Elasticity Detection: A Building Block for Internet Congestion Control

Prateesh Goyal[1], Akshay Narayan[1], Frank Cangialosi[1], Srinivas Narayana[2], Mohammad Alizadeh[1], Hari Balakrishnan[1]

[1]MIT CSAIL, [2]Rutgers University

ABSTRACT

This paper introduces Nimbus, a robust technique to detect whether the cross traffic competing with a flow is "elastic", and shows that this elasticity detector improves congestion control. If cross traffic is inelastic, then a sender can control queueing delays while achieving high throughput, but in the presence of elastic traffic, it may lose throughput if it attempts to control packet delay. To estimate elasticity, Nimbus modulates the flow's sending rate with sinusoidal pulses that create small traffic fluctuations at the bottleneck link, and measures the frequency response of the rate of the cross traffic. Our results on emulated and real-world paths show that congestion control using elasticity detection achieves throughput comparable to Cubic, but with delays that are 50–70 ms lower when cross traffic is inelastic. Nimbus detects the nature of the cross traffic more accurately than Copa, and is usable as a building block by other end-to-end algorithms.

1 INTRODUCTION

Achieving high throughput and low delay has been a key goal of congestion control research for decades. To achieve these goals, researchers have proposed many *delay-controlling* algorithms. These schemes (e.g., Vegas [2], FAST [32], LEDBAT [27], Sprout [33], Copa [1]) reduce their rates as delays increase to control packet delays and avoid "bufferbloat" [11], unlike methods like Cubic [12], NewReno [14], and Compound [28] that must fill buffers to elicit congestion signals (packet losses or ECN).

There is, however, a major obstacle to deploying delay-controlling algorithms on the Internet: their throughput is dismal when competing against buffer-filling flows at a shared bottleneck. The reason is that buffer-filling senders steadily increase their rates, causing queuing delays to rise; in response to increasing delays, a competing delay-controlling flow will reduce its rate. The buffer-filling flow then grabs this freed-up bandwidth. The throughput of the delay-controlling flow plummets, but delays don't reduce. Because most traffic on the Internet today uses buffer-filling algorithms, it is hard to justify deploying a delay-controlling scheme.

Is it possible to achieve the benefits of delay-controlling algorithms while ensuring that throughput does not degrade in the presence of buffer-filling schemes? We believe that a rigorous answer to this question requires the sender to understand the *nature* of the cross traffic. The salient aspect of this nature is whether the cross traffic is buffer-filling or not. That, however, is beyond our current abilities, but we contribute in this paper a new, rigorous algorithm to characterize whether cross traffic is *elastic* or not. We also show that elasticity is a good metric for determining whether the sender should attempt to control delays.

We define a flow to be elastic at a given bottleneck if it *increases its rate when it senses that more bandwidth is available there, and decreases it otherwise*. All other flows are *inelastic*. Examples of elastic flows include backlogged flows using either buffer-filling schemes like Cubic and NewReno, or delay-based schemes like Vegas, Copa, and BBR [4]. By contrast, constant bit-rate (CBR) flows, short TCP connections, application-limited flows, and flows bottlenecked at a different link are all inelastic. In general, traffic could have both elastic and inelastic flows; we define traffic to be elastic if it contains any elastic flows, and inelastic otherwise.

We have developed an *elasticity detector* called *Nimbus*, which any sender can use to make its congestion-control decisions. When Nimbus deems cross traffic to be inelastic, the sender can use a delay-controlling algorithm to achieve low delays, but when cross traffic is elastic, an algorithm that competes fairly with the cross traffic without necessarily attempting to control delay is required.

Because all buffer-filling flows are elastic, this approach guarantees that a sender using Nimbus will not lose throughput by using a delay-controlling method when competing with such flows. It may, however, miss out on opportunities to control delays when cross traffic is both elastic and itself delay-controlling (note that this is really no different from the status quo, where a Cubic, BBR, or even Copa flow will crush Vegas, for example). Competing fairly with delay-controlling cross traffic while achieving low delay requires a method to determine not only the nature, but also the type of congestion control algorithm used by the cross traffic.

Elasticity detection. Nimbus uses only end-to-end RTT measurements to monitor the cross traffic to determine if the cross traffic is elastic. The sender continuously modulates its rate with sinusoidal pulses to create small traffic fluctuations at the bottleneck at a specific frequency (e.g., 5 Hz). It concurrently estimates the rate of the cross traffic based on the its own send and receive rates, and monitors its frequency response (FFT) to determine if the cross traffic's rate oscillates at the same frequency. If it does, then the sender concludes that the cross traffic contains elastic flows; otherwise, it is inelastic.

This technique relies on two assumptions. First, the sender must be able to create sufficient pulses and observe the impact on cross traffic over a period of time. Thus it is best suited for

large data transfers. Fortunately, it is for such transfers that delay-controlling schemes are useful, because short flows are unlikely to cause significant queueing delay [11].

Second, pulsing is most effective when the elastic flows react on a timescale of a few RTTs. If an elastic flow is slower to react, it can go undetected with short pulses. On the other hand, using longer pulses to detect such "sluggish" elastic flows could cause congestion. The majority of traffic on the Internet reacts on RTT timescales (e.g., ACK-clocked TCP flows). Nimbus is targeted at detecting ACK-clocked flows, but we have found that it also correctly classifies fast-reacting rate-based flows as elastic.

In our experiments, we find that Nimbus is robust to a variety of cross traffic conditions, achieving at least 85% detection accuracy even when cross traffic is a combination of varying number of elastic flows and highly-varying inelastic short flows, or when cross traffic is composed of multiple elastic flows with different RTTs. These results hold across a wide range of network characteristics: buffer sizes, RTTs, bottleneck link rates, active queue management schemes, and fraction of traffic controlled by Nimbus.

NimbusCC is a congestion control system that uses elasticity detection to switch between TCP-competitive and delay-controlling modes. NimbusCC can support various algorithms in each mode. We report results with Vegas, Copa's default mode, and a simple new method that uses our cross-traffic rate estimator, as examples of delay-controlling algorithms, and Cubic and Reno as examples of TCP-competitive algorithms.

Key results: We have implemented NimbusCC in Linux using CCP [23]. Our experimental results show that:

(1) NimbusCC achieves throughput within 10% of the fair share against elastic traffic made up of a variable number of TCP flows, whereas Copa is 54% lower. NimbusCC also achieves 60 ms lower mean delay than Cubic against Poisson-distributed inelastic cross traffic.

(2) When cross traffic is modeled from a flow-size distribution measured at a WAN link [3], NimbusCC achieves throughput comparable to Cubic and BBR, but with 50 ms lower median delay. Copa has similar median throughput, but its 10th percentile of throughput is 60% lower than fair share because it performs a lot worse than NimbusCC against elastic cross traffic. By contrast, both NimbusCC and Cubic (which NimbusCC emulates) have a 10th percentile only 30% lower than fair share.

(3) On 25 different Internet paths, NimbusCC achieved a throughput at least as high as Cubic, exceeding Cubic on paths with policers, with lower delays on 60% of the paths and similar delays on the other 40%. Compared to BBR, NimbusCC's throughput was 10% lower, but the mean packet delay was 40–50 ms lower.

Our principal contribution is the idea that the nature of cross traffic, quantified as elasticity, is a useful building block for congestion control. We envision it being used to solve other problems in the future. For example, in tools like speedtest or iperf to inform users not only of the rate, but also whether the rate is lower than expected due to elastic cross traffic. Such a tool can shed light on traffic behavior and may also help guide the deployment of active queue management (AQM) schemes.

2 RELATED WORK

Copa [1] aims to maintain a bounded number of packets in the bottleneck queue. Copa induces a periodic pattern of sending rate that nearly empties the queue once every 5 RTTs. This helps Copa flows obtain an accurate estimate of the minimum RTT and the queuing delay. In addition, Copa uses this pattern to detect the presence of non-Copa flows: Copa expects the queue to be nearly empty at least once every 5 RTTs, provided only Copa flows with similar RTTs share the bottleneck link. If the estimated queuing delay does not drop below a threshold in 5 RTTs, Copa switches to a TCP-competitive mode.

Unlike Copa, Nimbus does not look for a pattern in the RTTs caused by its transmission pattern. Instead, it estimates the rate of the cross traffic and observes how the cross traffic reacts to the rate fluctuations it induces over a period of time. Thus, Nimbus directly estimates the elasticity of cross traffic. Although elasticity detection takes a few seconds, our experiments show that it is more robust than Copa's method. We show that:

(1) Copa incorrectly and frequently switches modes, losing throughput against elastic cross-traffic, e.g., by 54% than its fair share (§5 and §8.1)). The reason is that Copa uses instantaneous measurements for mode-switching, making it vulnerable to variations in the cross traffic.

(2) Copa misclassifies cross traffic when the inelastic traffic rate is high, or when elastic flows have high RTTs (§8.2).

Moreover, since Nimbus does not rely on properties of any specific control algorithm (e.g., emptying queues every 5 RTTs), it applies to any combination of TCP-competitive and delay-controlling schemes and can be used as a building block.

BBR [4] estimates the bottleneck bandwidth (b) and minimum RTT (d). It paces traffic at a rate b while capping the number of in-flight packets to $2 \times b \times d$. To estimate the bottleneck, BBR periodically increases its rate over b for about one RTT and then reduces it for the following RTT. BBR uses this sending-rate pattern to obtain estimates of b; specifically, it tests if the bottleneck rate exceeds the current estimate b in the rate-increase phase. However, BBR doesn't use these pulses to infer the nature of cross traffic.

PCC-Vivace [7] uses an online learning algorithm to adapt its sending rate to maximize a utility function that incorporates the achieved rate, delay, and loss rate. Our experiments (§5, §8.1) show that Vivace cannot achieve both low delay with inelastic cross traffic and compete fairly with elastic TCP flows. Compound TCP [30] maintains both a loss-based window and a delay-based window, and transmits data based on the sum of the two windows. Compound does not attempt to switch between two modes, and therefore it incurs high queuing delays due to its loss-based window.

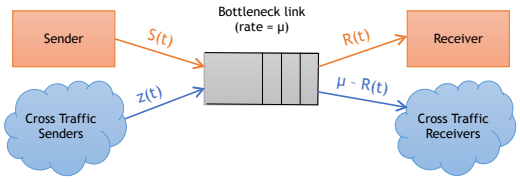


Figure 1: Network model. The time-varying total rate of cross traffic is $z(t)$. The bottleneck link rate is μ. The sender's transmission rate is $S(t)$, and the rate of traffic received by the receiver is $R(t)$.

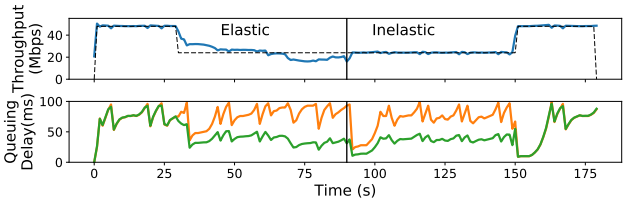


Figure 2: Instantaneous delay measurements do not reveal elasticity. The bottom plot shows the total queueing delay (orange) and the self-inflicted delay (green). The experiment contains one background Cubic flow in the elastic region (30–90 s) and CBR cross traffic in the inelastic region (90–150 s).

3 CROSS-TRAFFIC ESTIMATION

We present a simple new method to estimate the total rate of cross traffic at the sender (§3.1). Then, we show how to detect whether the cross traffic contains *any* ACK-clocked elastic flows, describing the key principles (§3.2) and a practical method (§3.3).

Figure 1 shows our network model and introduces some notation. A sender communicates with a receiver over a single bottleneck link of rate μ. The bottleneck link is shared with cross traffic, consisting of an unknown number of flows, each of which is either elastic or inelastic. $S(t)$ and $R(t)$ denote the time-varying sending and receiving rates, respectively, while $z(t)$ is the total rate of the cross traffic. We assume that the sender knows μ, and can use prior work to estimate it (§4.2).

3.1 Estimating the Rate of Cross Traffic

In Fig. 1, the total traffic into the bottleneck queue is $S(t)+z(t)$, of which the receiver sees $R(t)$. As long as the bottleneck link is busy (i.e., its queue is not empty), and the router treats all traffic the same way, the ratio of $R(t)$ to μ must be equal to the ratio of $S(t)$ and the total incoming traffic, $S(t)+z(t)$. Using this property, we propose a new estimator for $z(t)$:

$$\hat{z}(t) = \mu \frac{S(t)}{R(t)} - S(t). \tag{1}$$

We estimate $S(t)$ and $R(t)$ by considering n packets at a time:

$$S_{i,i+n} = \frac{n_{bytes}}{s_{i+n} - s_i}, \qquad R_{i,i+n} = \frac{n_{bytes}}{r_{i+n} - r_i}, \tag{2}$$

where n_{bytes} is the number of bytes in the n packets, s_k is the time at which the sender sends packet k, r_k is the time at which the sender receives the ACK for packet k, and the units of the rates are bytes per second. Note that $S(t)$ and $R(t)$ must be measured over the *same* n packets.

We have conducted several tests with various patterns of cross traffic to evaluate the effectiveness of this $z(t)$ estimator. The overall error is small: the 50th and 95th percentiles of the relative error are 1.3% and 7.5%, respectively. Unlike prior work on estimating cross-traffic rate [16, 18, 29], our method is in-band and does not use any probe packets; it relies on the property that the sender is persistently backlogged.

3.2 Elasticity Detection: Principles

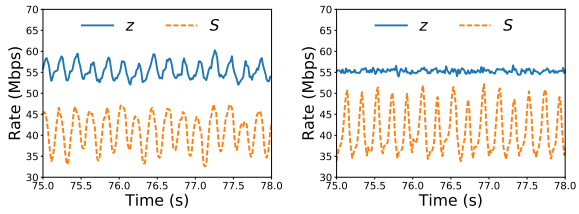
We now turn to designing an online estimator for a sender to determine if the cross traffic includes *any* elastic flows.[1] A strawman approach might attempt to detect elastic flows by estimating the contribution of the cross traffic to queueing delay. For example, the sender can estimate its own contribution to the queueing delay—i.e., the "self-inflicted" delay—and if the total delay is significantly higher than the self-inflicted delay, conclude that the cross traffic is elastic.

This scheme does not work. To see why, consider the experiment in Figure 2, where a Cubic flow shares a link with elastic and inelastic traffic in two separate time periods. The self-inflicted queueing delay for the Cubic flow (green, bottom figure) looks the same in the elastic and inelastic phases. The reason is that a flow's share of the queue occupancy is proportional to its throughput, which is roughly the same in the two phases (top figure). Because the Cubic flow gets 50% of the bottleneck link, its self-inflicted delay is roughly half of the total queueing delay always (orange, bottom figure). This example suggests that instantaneous measurements cannot be used to distinguish between elastic and inelastic cross traffic.

To detect elasticity, tickle the cross traffic! Our method detects elasticity by monitoring how the cross traffic responds to induced traffic variations at the bottleneck link over a period of time. The key observation is that elastic flows react in a predictable way to rate fluctuations at the bottleneck. Consider, for example, long-running Cubic or Reno flows, which are ACK-clocked. For these flows, if an ACK is delayed by a time duration δ, then the next packet transmission will also be delayed by δ. Therefore changes in the rate of packet arrivals at the receiver cause similar changes in the sending rate after one RTT via the ACKs. By contrast, the sending rate of inelastic flows does not depend on the receive rate.

We induce changes in the inter-packet spacing of cross traffic at the bottleneck link by sending packets in *pulses*. We take the desired sending rate, $S(t)$, and alternate between sending at rates higher and rates lower than $S(t)$, ensuring that the mean rate is $S(t)$. Sending in such pulses (e.g., modulated on a sinusoid) changes the inter-packet spacing of the cross traffic departing the bottleneck link in a controlled manner.

[1] Receiver participation will improve accuracy by avoiding the need to estimate $R(t)$ from ACKs at the sender, but would be a little harder to deploy.



(a) Elastic cross traffic **(b) Inelastic cross traffic**

Figure 3: Cross traffic's reaction to pulses. The pulses change the inter packet spacing for cross traffic. Elastic traffic reacts to these changes after a RTT. Inelastic cross traffic is agnostic to these changes.

If the cross traffic contains elastic flows, then because of the induced changes in the ACK clocks of those flows, their rates will react to our pulses. When we increase our rate, the elastic cross traffic will reduce its rate in the next RTT, and conversely. If enough of the cross traffic is elastic, then our sender can measure and detect these fluctuations in the cross traffic rate.

Fig. 3a and Fig. 3b compare the responses of elastic (Cubic) and inelastic (constant bit rate) cross traffic when the sender transmits packets in sinusoisal pulses at frequency $f_p = 5$ Hz. $S(t)$ is the sender's rate and $z(t)$ is the estimated cross traffic rate computed using Eq. (1). The path has a minimum RTT of 50 ms and a buffer size of 100 ms (2× the bandwidth-delay product). The elastic flow's sending rate after one RTT is inversely correlated with the pulses in the sending rate, while the inelastic flow's sending rate is unaffected.

3.3 Elasticity Detection: Practice

To produce a practical method to detect cross traffic using this idea, we must address three challenges:

(1) Pulses in the sending rate must induce a measurable change in z, but not congest the bottleneck link.
(2) Because there is natural variation in cross traffic, and noise in $\hat{z}$, it is not easy to perform a robust comparison between the predicted change in z and the measured z.
(3) Because the sender does not know the RTTs of cross-traffic flows, it does not know when to look for the predicted response in the cross-traffic rate.

The first method we developed to solve these problems measured the *cross-correlation* between $S(t)$ and $z(t)$. A cross-correlation near zero would be considered inelastic cross traffic, whereas a significant non-zero value would indicate elastic cross traffic. We found that this approach works well (with square-wave pulses) if the cross traffic is substantially elastic and has a similar RTT to the flow trying to detect elasticity, but not otherwise. The trouble is that because elastic cross traffic will react after *its* RTT, $S(t)$ and $z(t)$ must be aligned using the cross traffic's RTT, which is not easy to infer. Moreover, the elastic flows in the cross traffic may have different RTTs, making the alignment even more challenging.

From time to frequency domain. We have developed a method, Nimbus, that overcomes the three challenges stated above. It uses two ideas. First, the sender modulates its packet transmissions using *sinusoidal pulses* at a known frequency f_p, with amplitude equal to a modest fraction (e.g., 25%) of the bottleneck link rate. These pulses induce a noticeable change in inter-packet times at the link without causing congestion, because the queues created in one part of the pulse are drained in the subsequent part, and the period of the pulses is short (e.g., $f_p = 5$ Hz). By using short pulses, we ensure that the total burst of data sent in a pulse is a small fraction of the typical bottleneck queue size.

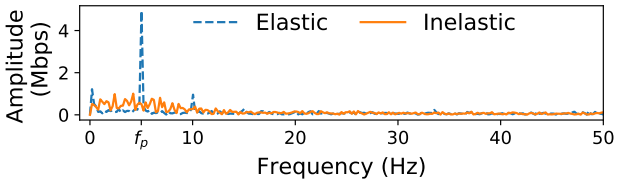


Figure 4: Cross traffic FFT for elastic and inelastic traffic. Only the FFT for elastic traffic has a pronounced peak at f_p (5 Hz).

Second, the sender looks for periodicity in the cross traffic rate at frequency f_p, using a frequency domain representation of the cross-traffic rates. We use the Fast Fourier Transform (FFT) of the time series of the cross traffic estimate $\hat{z}(t)$ over a short time interval (e.g., 5 seconds). Detecting periodicity in the frequency domain is more robust than the time-domain, for the same reason that frequency modulation provides better signal-to-noise ratio than amplitude modulation [26]: it is less affected by variations in the cross traffic rate and measurement noise. Further, observing the cross traffic's response at a known frequency, f_p, yields a method that is robust to the presence of multiple ACK-clocked flows with different RTTs. All the elastic flows in the cross traffic, irrespective of their RTTs and congestion control protocol, will exhibit rate oscillations at the frequency f_p. As a result, there will be an overall response at frequency f_p in the cross traffic, equal to superposition of the responses of the individual elastic flows at frequency f_p.[2]

Fig. 4 shows the FFT of the $\hat{z}(t)$ time-series produced using Eq. (1) for examples of elastic and inelastic cross traffic, respectively. Elastic cross traffic exhibits a pronounced peak at f_p compared to the neighboring frequencies, while for inelastic traffic the FFT magnitude is spread across many frequencies. The magnitude of the peak depends on how much of the cross traffic is elastic; the more elastic the cross traffic, the sharper the peak at f_p. Therefore, rather than compare the peak at f_p to a pre-determined threshold, we compare it to the magnitude of the nearby frequencies.

We define the *elasticity metric*, η, as follows:

$$\eta = \frac{|FFT_z(f_p)|}{\max_{f \in (f_p, 2f_p)} |FFT_z(f)|} \tag{3}$$

Eq. (3) compares the magnitude of the FFT at frequency f_p to the peak magnitude in the range from just above f_p to just below $2f_p$. If η is less than a threshold $\eta_{thresh} (\geq 1)$, then the cross traffic is deemed inelastic; otherwise, it is elastic.

[2] In theory, the response of flows with different RTTs may cancel each other out, but this is very unlikely since it requires specific combinations of RTTs. We have not seen this problem occur in our experiments (§8.2).

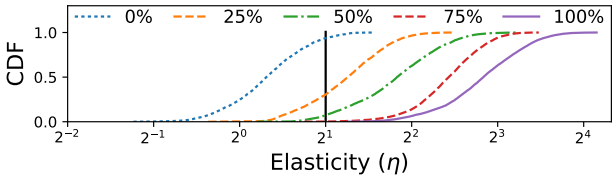


Figure 5: Distribution of elasticity with varying elastic fraction of cross traffic. The cross traffic consists of an elastic Cubic flow and inelastic Poisson-distributed traffic with different rates. Completely inelastic cross traffic has η close to zero, while completely elastic cross traffic exhibits η. Cross traffic with some elastic fraction also exhibits high elasticity ($\eta > 2$).

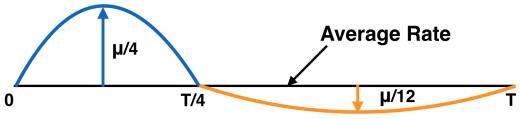


Figure 6: Asymmetric sinusoidal pulse. The pulse has period $T = 1/f_p$. The positive half-sine lasts for $T/4$ with amplitude $\mu/4$, and the negative half-sine lasts for the remaining duration, with amplitude $\mu/12$. The two half-sines cancel out each other over one period.

3.4 Setting Parameters for Elasticity Detection

Detection threshold. In practice, cross traffic is likely to be a mix of elastic and inelastic flows. In such scenarios, we want our detector to be sensitive to the presence of any elastic flows, since even one elastic flow can eventually grab all the link bandwidth from a delay-controlling flow. A large value of η_{thresh} will ensure that purely inelastic traffic will always be classified correctly, but cross traffic with small elastic components will be misclassified. Fig. 5 shows the CDF of elasticity (η) as the fraction of bytes belonging to elastic flows in the cross traffic varies. The median values range from $\eta = 1$ for purely inelastic traffic to $\eta = 10$ for purely elastic traffic. We choose a fixed threshold $\eta_{thresh} = 2$, which corresponds to classifying 25% elastic cross traffic correctly 75% of the time.

FFT duration. Computing FFTs over a small duration allows quick responses to changes in cross traffic, but it increases errors due to noise. Variations in the inelastic cross traffic over small periods can cause false peaks at f_p in the FFT, causing that traffic to be incorrectly classified. We choose an FFT duration of 5 seconds to balance these concerns.

Pulse shaping. Rather than a pure sinusoid, we use an *asymmetric* sinusoidal pulse, as shown in Fig. 6. In the first one-quarter of the pulse cycle, the sender adds a half-sine of a certain amplitude (e.g., $\mu/4$) to $S(t)$; in the remaining three-quarters of the cycle, it subtracts a half-sine with one-third of the amplitude used in the first quarter of the cycle (e.g., $\mu/12$). The reason for this asymmetric pulse is that it enables senders with low sending rates, $S(t)$, to generate pulses. For example, for a peak amplitude of $\mu/4$, a sender with $S(t)$ as low as $\mu/12$ can generate the asymmetric pulse shown in Fig. 6; a symmetric pulse with the same peak rate would require $S(t) > \mu/4$.

Our pulses produce an observable pattern in the FFT when the cross traffic is elastic. Using asymmetric sinusoidal pulses creates harmonics at multiples of the pulse frequency f_p. However, these harmonics do not affect η (see Eq. (3)), which only uses the FFT in the frequency band $[f_p, 2f_p)$.

Pulse duration. What should the duration, T, of the pulse be? The answer depends on two factors: first, the interval over which S and R are measured (with which the sender computes $\hat{z}$), and second, the amount of data we are able to send in excess of the mean rate without causing congestion. If T were smaller than the measurement interval of S and R, the perturbation to the cross traffic rate during one part of the pulse will be averaged out during the rest of the pulse, resulting in no impact on $\hat{z}(t)$. But T cannot be too large because the sender transmits in excess of the mean rate $S(t)$ for $T/4$. In particular, the size of the burst sent in a pulse is $\frac{2}{\pi}\frac{\mu}{4}\frac{T}{4} = \frac{T\mu}{8\pi} \approx 0.04\mu T$. If T is equal to the RTT, this is 4% of the bandwidth-delay product (BDP).

We set T to a large RTT value observed on the Internet, for example $T = 200$ ms, with the rationale that router buffers are typically provisioned to avoid packet losses for one such RTT, and because our implementation measures S and R over one RTT. We measure rates over one RTT because sub-RTT measurements are confounded by burstiness in packet transmissions (e.g., caused by ACK compression [19]).

If the cross traffic reacts slower than the pulse duration, Nimbus might misclassify those flows. A longer pulse duration, corresponding to the response timescale of the elastic traffic, could detect such flows. However, longer pulses would also send more traffic into the network and might cause congestion. We evaluate this alternative for detecting PCC-Vivace, a rate-based scheme (not ACK-clocked), in Appendix F.

4 NIMBUSCC

NimbusCC is a congestion control system that uses mode switching. It has a TCP-competitive mode in which the sender transmits using a TCP-competitive congestion control algorithm (e.g., Cubic), and a delay-control mode that uses a delay-controlling algorithm (e.g., Copa). NimbusCC switches between the two modes using our elasticity detector, Nimbus.

4.1 Mode Switching

At any given time, NimbusCC transmits data at the time-varying rate dictated by the congestion control algorithm running at that time. It modulates this rate with asymmetric sinusoidal pulses (Fig. 6). NimbusCC uses the pulsing parameters described in §3.4, calculating S and R over one window's worth of packets. It computes the FFT for the z measurements reported in the last 5 seconds to calculate elasticity (η) using Eq. (3), and it picks the mode by comparing η to $\eta_{thresh} = 2$ (§3.4).

We support Cubic and NewReno for the TCP-competitive mode and Copa's default mode and Vegas for the delay-control mode. We also implemented a basic delay-controlling algorithm, BasicDelay, using our cross traffic rate estimator.

Let S be the sending rate and $\hat{z}$ be the estimated cross-traffic rate, both measured over the last window of packets. Also, let x be the current RTT, and x_{min} be the minimum observed RTT. Upon receiving an ACK, BasicDelay sets its current rate to:

$$\text{Rate} \leftarrow S + \alpha(\mu - S - \hat{z}) + \beta\frac{\mu}{x}(x_{\min} + d_t - x), \qquad (4)$$

Scheme	Throughput Δ Elastic	Throughput Δ Inelastic	QDelay Inelastic
NimbusCC Cubic+BasicDelay	−10%	0%	12 ms
NimbusCC Cubic+Copa	−15%	−1%	14 ms
Cubic	+12%	0%	78 ms
BBR	+61%	−2%	56 ms
Vegas	−79%	−15%	3 ms
Compound	−25%	0%	45 ms
Copa	−54%	−19%	18 ms
PCC-Vivace	+61%	−2%	27 ms

Table 1: Average queuing delay (in ms) in the inelastic region, and deviation from fairshare throughput in elastic and inelastic regions from Fig. 7. NimbusCC is the only scheme to achieve close to fair-share throughput and low delays.

where α and β are constants smaller than 1, and d_t is a target queuing delay. The term $(\mu - S - z)$ is the sender's estimate of the spare capacity in the last RTT. By adding an α-fraction of the spare capacity to $S(t)$, BasicDelay tries to get closer to the ideal rate. The second term in the above rule seeks to maintain a specified queuing delay, d_t, to prevent the queue from both growing too large or going empty. Recall that our cross traffic estimator, Eq. (1), requires a non-empty queue to estimate z.

NimbusCC takes special care in initializing the rate when switching to TCP-competitive mode. NimbusCC sets the rate (and equivalent window) to the rate that was used 5 seconds ago because the elasticity detector takes 5 seconds (FFT Duration) to detect elastic cross traffic. During this time, the elastic traffic could cause a reduction in the delay-control mode's rate. Hence, NimbusCC resets its rate to the rate at the beginning of the 5-second detection period.

4.2 Implementation

We implemented NimbusCC using CCP [23], which provides a convenient way to express the signal processing operations in user-space code. It uses estimates of S, R, the RTT, and packet losses from the Linux kernel every 10 ms.

Calculating $\hat{z}$ requires an estimate of the bottleneck link rate (μ). There has been much prior work [8, 9, 15, 17, 20–22] in estimating μ, which NimbusCC could use. We use the maximum received rate as the estimate, taking care to avoid incorrect estimates due to ACK compression. We evaluate the impact of errors in estimating μ on elasticity detection in §8.2.

5 VISUALIZING NIMBUSCC

We illustrate NimbusCC on a synthetic workload with time-varying cross traffic. We emulate a bottleneck link in Mahimahi [24], a link emulator. The network has a bottleneck rate of 96 Mbit/s, a minimum RTT of 50 ms, and 100 ms (2 BDP) of buffering. We compare two mode-switching protocols, NimbusCC (Cubic+BasicDelay) and NimbusCC (Cubic+Copa), with Cubic, BBR, Vegas, and PCC-Vivace (all from Linux), Copa (from Copa's authors), and Compound atop CCP (written by us).

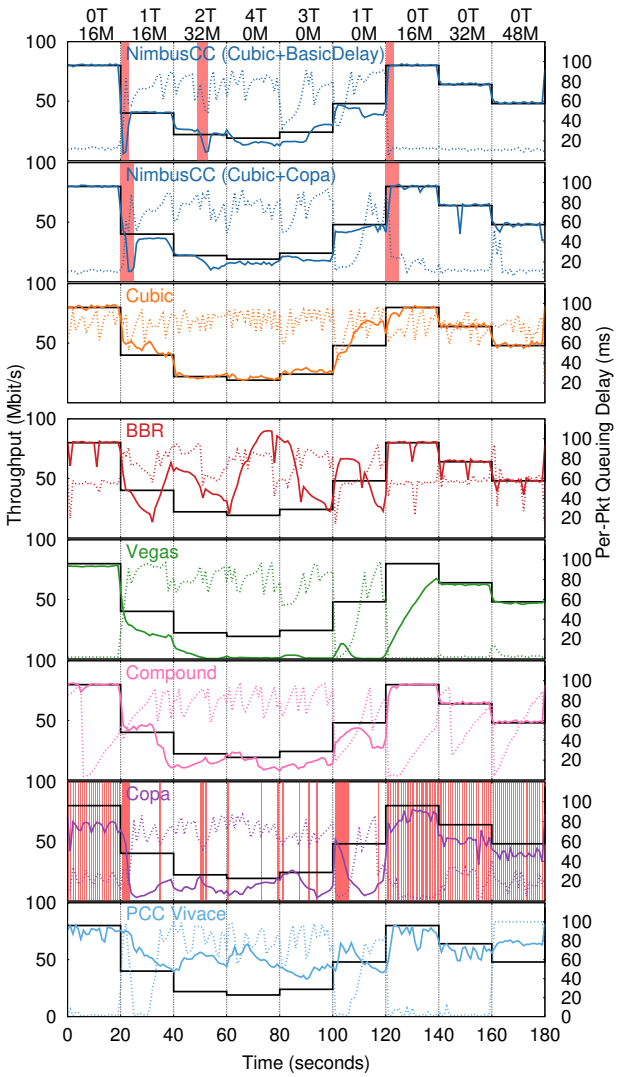


Figure 7: Performance on a 96 Mbit/s Mahimahi link with 50 ms delay and 2 BDP of buffering while varying the rate and type of cross traffic as denoted at the top of the graph. xM denotes x Mbit/s of inelastic Poisson cross-traffic. yT denotes y long-running Cubic cross-flows. The solid black line indicates the correct time-varying fair-share rate that the protocol should achieve given the cross-traffic. For each scheme, the solid line shows throughput and the dotted line shows queuing delay. The cross-traffic contains elastic flows from 20–120 s. For Nimbus and Copa, the red shaded regions indicate times spent in the wrong mode (e.g., delay-controlling with elastic cross traffic).

The cross traffic varies over time between elastic, inelastic, and a mix of the two. We generate inelastic cross-traffic using Poisson packet arrivals at the specified mean rate. Elastic cross-traffic uses Cubic, via `iperf` [31].

Fig. 7 shows the throughput and queuing delays for the various protocols, as well as the correct fair-share rate. Table 1 summarizes the deviation from fair-share throughput in the elastic (20–120 s) and inelastic (0–20 and 120–180 s) regions, and the mean queuing delay in the inelastic region. The delay in the elastic region is similar for all schemes.

Throughout the experiment, both NimbusCC variants achieve throughput close to the fair-share rate and low ($\leq$15 ms) queuing delays in the presence of inelastic cross traffic. With elastic cross traffic, both variants switch to TCP-competitive mode within 5 seconds and achieve close to their fair share. The delays during this period approach the buffer size because the competing traffic is buffer-filling; the delays return to their previous low value (15 ms) within 5 seconds after the elastic flows complete. NimbusCC stays in the correct mode throughout the experiment, except for one interval in the elastic period. The deviation from fair-share in the elastic region is because Cubic is not perfectly fair to itself over short time periods.

Cubic achieves it's fair-share rate but experiences high delays (80 ms) throughout. BBR's throughput is often much higher than its fair share with high delays even against inelastic cross-traffic, which prior work has also observed [1, 13].

Vegas suffers from low throughput in the presence of elastic cross-traffic as it reacts to packet delays. Compound ramps up its rate quickly when it detects low delays, but behaves like TCP Reno otherwise. Hence, it attains slightly lower than its fair-share rate in the presence of Cubic flows, and suffers from high delays even with inelastic cross-traffic.

Unlike Nimbus, Copa uses instantaneous measurements for mode-switching, and is vulnerable to the variations in cross traffic. As a result, while Copa generally uses the correct mode it frequently switches mode unnecessarily; Copa makes 28 switching errors in the elastic region, while NimbusCC only switches once. In the elastic period, Copa's frequent mode switches lower its throughput (14 Mbit/s) compared to NimbusCC (27.5 Mbit/s) and fair-share rate (e.g., see 100–120 s). Further, by draining queues periodically, Copa incurs minor underutilization against inelastic traffic (e.g., 140–160 s).

Vivace competes unfairly with elastic traffic. At times, Vivace fails to maintain low delays against inelastic cross traffic and incurs heavy packet loss (e.g., 160–180s).

6 MULTIPLE NIMBUSCC FLOWS

What happens when a bottleneck is shared by multiple NimbusCC flows? Ideally, we want all the NimbusCC flows to remain in delay-control mode when there is no elastic cross traffic, and compete well with elastic cross traffic otherwise.

One approach is for the NimbusCC flows to all pulse at the same frequency. However, in this case, they will all detect a peak in the FFT at the oscillation frequency. They will all then stay in TCP-competitive mode and won't be able to maintain low delays, even when there is no elastic cross traffic. A second approach is for different NimbusCC flows to pulse at different frequencies. But this approach cannot scale to more than a few flows, because the set of distinguishable frequencies is limited (recall that the pulse period T cannot be too small).

The pulser and the watchers. We propose a third approach. One of the NimbusCC flows assumes the role of the *pulser*, while the others are *watchers*. They coordinate with no explicit communication; in fact, each NimbusCC flow is unaware of the identities, or even existence, of the others.

The pulser sends data by modulating its rate with asymmetric sinusoids. The pulser uses two different frequencies, f_{pc} in TCP-competitive mode, and f_{pd} in delay-control mode. The values of these frequencies are fixed and agreed upon beforehand; we use $f_{pc} = 5$ Hz and $f_{pd} = 6$ Hz in our experiments.[3]

A watcher infers whether the pulser is pulsing at frequency f_{pc} or frequency f_{pd} by computing the FFT of its receive rate, R, at these two frequencies. It then picks the mode corresponding to the larger peak to match the pulser's mode. Note that since a watcher is not pulsing, it can detect the pulser's pulses in its own receive rate, R; i.e., it does not even need to estimate z. The pulser, on the other hand, cannot look at its own R to detect pulses in the cross traffic, since it will end up detecting its own pulses.

For multiple NimbusCC flows to maintain low delays during times when there is no elastic cross traffic on the link, the pulser must classify watcher traffic as inelastic. Note that from the pulser's perspective, the watcher flows are part of the cross traffic; thus, to avoid confusing the pulser, the rate of watchers must not react to the pulses of the pulser. To achieve this goal, a watcher applies an exponentially weighted moving average (EWMA) filter to its transmission rate before sending data. The EWMA filter cuts off all frequencies in the sending rate that exceed $\min(f_{pc}, f_{pd})$.

Pulser election. A distributed and randomized election decides which flow is the pulser and which are watchers. If a NimbusCC flow determines that there is no pulser (by seeing that there is no peak in the FFT at the two potential pulsing frequencies), then it decides to become a pulser with a probability proportional to its transmission rate:

$$p_i = \frac{\kappa\tau}{\text{FFT Duration}} \times \frac{R_i}{\mu}. \tag{5}$$

Each flow makes decisions periodically, e.g., every $\tau = 10$ ms, κ is a constant, and R_i is the receive rate of the i^{th} flow. This rule ensures that the expected number of flows that become pulsers over the FFT duration is at most κ. To see why, note that the expected number of pulsers is equal to the sum of the probabilities in Eq. (5) over all the decisions made by all flows in the FFT duration. Since $\sum_i R_i \leq \mu$ and each flow makes (FFT Duration/τ) decisions, these probabilities sum up to at most κ.

It is also not difficult to show that the number of pulsers within an FFT duration has approximately a Poisson distribution with a mean of κ [10]. Thus the probability that after one flow becomes a pulser, a second flow also becomes a pulser before it can detect the pulses of the first flow in its FFT measurements is $1-e^{-\kappa}$. Therefore, κ involves a tradeoff: a smaller κ will lead to fewer conflicts but will take longer to elect a pulser.

For any value of κ, there is a non-zero probability of more than one concurrent pulser. If there are multiple pulsers, then each pulser will observe that the cross traffic has more variation than the variations it creates with its pulses. This can be detected by comparing the magnitude of the FFT of the cross traffic $z(t)$ at f_p with the FFT of the pulser's receive rate

[3] These values are in accordance with bounds on T and f described in §4.

Cross Traffic	Elastic	ACK-Clocked	Classification
Cubic	Yes	Yes	Elastic
Reno	Yes	Yes	Elastic
Copa	Yes	Yes	Elastic
Vegas	Yes	Yes	Elastic
BBR	Yes	If CWND-limited	Elastic*
PCC-Vivace	Yes	No	Inelastic*
Fixed window	Yes	Yes	Elastic
App. limited	No	No	Inelastic
Const. stream	No	No	Inelastic

Table 2: Classification by Nimbus.

$R(t)$ at f_p. If the cross traffic's FFT has a larger magnitude at f_p, the NimbusCC pulser concludes that there must be multiple pulsers and switches to a watcher with a fixed probability.

7 LIMITATIONS

Table 2 summarizes how Nimbus classifies different types of cross traffic. Recall that our method relies on the cross traffic responding to variations induced by pulses on an RTT timescale. This is true of all ACK-clocked protocols, which are classifed as elastic.

Nimbus does not always classify BBR cross traffic as elastic. When the buffer is large, BBR becomes ACK-clocked and is elastic. However, when the buffer is small, BBR responds on timescales longer than an RTT; here, Nimbus classifies it as inelastic. Nonetheless, we find that NimbusCC (with Cubic as the TCP-competitive protocol) achieves similar throughput to Cubic when competing against BBR (Appendix §C).

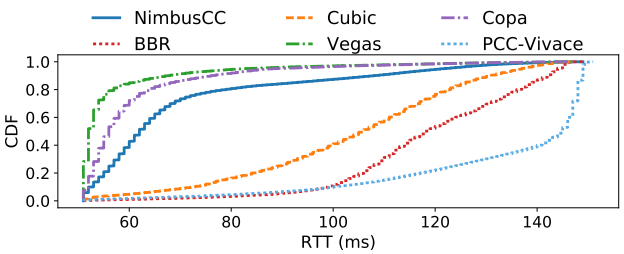
Rate-based protocols, (e.g., PCC-Vivace) may not react on RTT timescales. For example, Nimbus in its default configuration classifies PCC-Vivace as inelastic because it does not react quickly enough to Nimbus's pulses. Increasing the pulse duration helps Nimbus to correctly classify such flows as elastic (Appendix F). Increasing the pulse duration might of course also increase queuing delays. Since most elastic traffic today is ACK-clocked, we use a small pulse duration by default. In the future, if rate-based protocols become widely deployed, the pulse duration could be adjusted accordingly.

The detector also assumes that the flow has a single bottleneck. Multiple bottlenecks can add noise to Nimbus's rate measurements, preventing accurate cross-traffic estimation. The challenge is that the spacing of packets at one bottleneck is not preserved when traversing the second bottleneck.

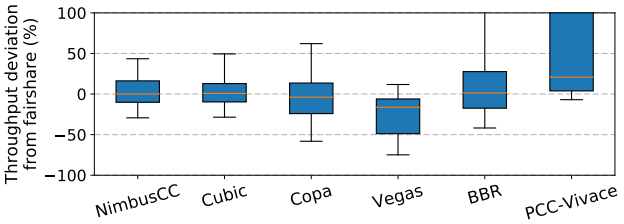
While Nimbus can detect presence of elastic flows, it cannot detect the specific congestion protocol used by competing flows. If the TCP-competitive protocol NimbusCC uses is different from that of the cross traffic, there could be unfairness. Further, if elastic cross traffic is using a delay-controlling scheme like Vegas, then Nimbus could miss out on an opportunity to control delays if it uses a buffer-filling TCP-competitive algorithm. Detecting the congestion control protocols used by competing elastic flows remains an open question.

8 EVALUATION

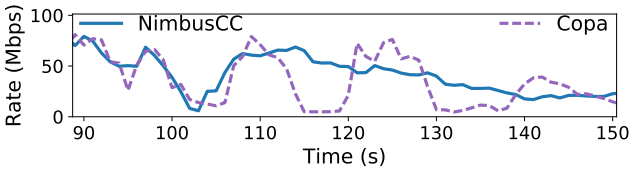
We evaluate our elasticity detection method, Nimbus, and a specific protocol using NimbusCC: Cubic+BasicDelay, as in §4.1. We use the Mahimahi emulator and measure the performance benefits (§8.1), robustness (§8.2), and fairness (§8.3) of elasticity detection with realistic traffic workloads. We also evaluate the performance of NimbusCC on real Internet paths (§8.4). All experiments use our Linux implementation (§4.2).



(a) NimbusCC reduces delay relative to Cubic, BBR and PCC-Vivace. It has higher delays than Copa and Vegas, but those two schemes have lower throughput than the fair share against elastic cross-traffic (see figure below).



(b) Deviation in throughput from fair-share. NimbusCC and Cubic achieve highest fairness. Vegas and Copa deviate from fair-share at lower percentiles and lose throughput; BBR and PCC-Vivace are significantly higher than fair share. Midline is the median, the box edges are the 25%ile and 75%ile, the whisker notches are 10%ile and 90%ile.



(c) Copa incorrectly switches to its default delay-control mode even when competing against elastic traffic, unlike NimbusCC.

Figure 8: Performance of NimbusCC on a cross-traffic workload derived from a packet trace collected at a WAN router.

8.1 NimbusCC Benefits from Elasticity Detection

We evaluate the delay and throughput benefits of mode switching using trace-driven emulation. We generate cross traffic from an empirical distribution of flow sizes derived from a wide-area packet trace from CAIDA [3]. This packet trace was collected at an Internet backbone router on January 21, 2016 and contains over 30 million packets recorded over 60 seconds. We generate Cubic cross-flows with flow sizes drawn from this data, with flow arrival times generated by a Poisson process to offer a fixed average load to fill 50% of the link (48 Mbit/s). Since the flow size distribution is heavy-tailed, the traffic trace consists of periods with *a mix of elastic and inelastic cross-traffic,* along with periods with only inelastic cross-flows.

One backlogged flow running a fixed algorithm (NimbusCC, Cubic, Copa, Vegas, PCC-Vivace or BBR) and the cross-traffic flows share a 96 Mbit/s Mahimahi bottleneck link with a propagation RTT of 50 ms and a buffer size of 100 ms. For BasicDelay we used $\alpha = 0.8$, $\beta = 0.5$ and $d_t = 12.5$ ms.

NimbusCC reduces delays while achieving fair-share throughput. Fig. 8a shows the distribution of per-packet RTT and Fig. 8b shows the deviation from fair-share throughput (over 5-second intervals) for various schemes. High deviation, i.e., unfairness with cross traffic, can harm application performance. For example, for a video application, temporary throughput drops can cause stalls, hurting user experience.

NimbusCC and Cubic achieve the lowest deviation from fair share among these schemes. NimbusCC's deviation profile is comparable to Cubic (note that both NimbusCC and Cubic deviate from the fair share since Cubic is not perfectly fair to itself over short time periods). The reason is that NimbusCC correctly switches to Cubic mode in the presence of elastic flows. Additionally, by switching to delay-controlling mode in the absence of elastic flows, NimbusCC achieves lower RTTs, with a median delay only 10 ms higher than Vegas and >50 ms lower than Cubic and BBR.

Cost of incorrect mode-switching. Copa has a slightly lower median delay than Nimbus, but at a high cost: its throughput deviates significantly from the fair-share at the 10th and 25th percentiles. Fig. 8c demonstrates why, comparing NimbusCC and Copa during a 60-second interval. Copa often incorrectly operates in its default delay-controlling mode against elastic cross-traffic (e.g., 115–120, 130–140 s).

Note that in this experiment we would expect the mean throughput for delay-controlling schemes to be the fair-share, since the total number of bytes in the cross traffic is fixed. The cross-traffic flow sizes are fixed, and a flow can last for different time durations depending on its throughput. Since the flow sizes are fixed, fair co-existence of NimbusCC with elastic cross-fows increases the lifetimes of those flows relative to Copa. This results in NimbusCC achieving lower (but fairer) throughput than Copa during periods (e.g., 120–130 s) when an elastic flow has completed in Copa, but not in NimbusCC. Moreover, since elastic flows last longer in NimbusCC, the delay is higher than Copa at the tail.

NimbusCC helps cross traffic. The 95th percentile flow completion time (FCT) of cross-traffic flows reduces by 3-4× compared to BBR, and 1.3× compared to Cubic for short (≤ 15 KB) flows (Appendix B). In contrast, PCC-Vivace is unfair to the background flows (positive deviation from fairshare). It grabs significantly more bandwidth than all the other schemes and keeps the buffer near-full more than half the time. The result is that many background flows do not complete, and their completion times are over 100× worse than with other schemes. PCC-Vivace also shows higher delays that any other scheme; the median delay is 90 ms higher than NimbusCC.

We repeated the above experiment with cross-traffic BBR flows instead of Cubic. Again, NimbusCC achieves a

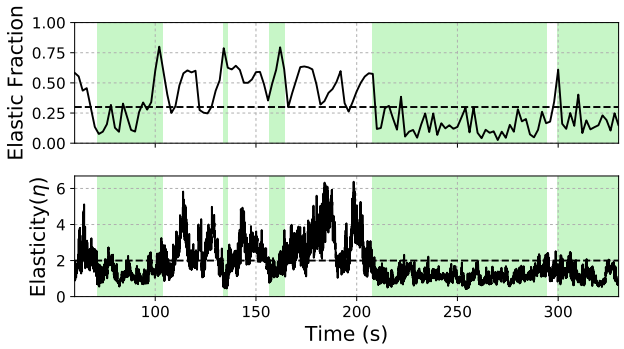


Figure 9: The elasticity metric closely tracks elastic cross-traffic (ground truth measured independently from the rate of ACK-clocked flows). Green-shaded regions indicate inelastic periods.

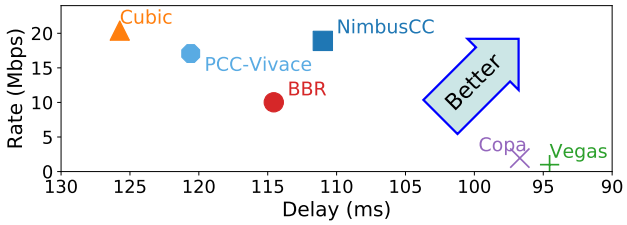


Figure 10: Throughput and mean delay (lower delay on the right) with video cross traffic. NimbusCC achieves similar throughput as Cubic but reduces delays and performs better than the other schemes. Copa and Vegas achieve low throughput.

throughput profile similar to Cubic while reducing delays (Appendix C).

Elasticity detection is accurate compared to ground truth. To define ground truth, we note that short flows (< 10 packets) transmit all data at once, without any rate adjustments. We thus classify a cross-flow as elastic if it is larger than the initial congestion window of 10 packets, finishing in greater than a RTT.

The top chart in Fig. 9 shows the fraction of bytes belonging to elastic flows as a function of time. The bottom chart shows the output of the elasticity detector with the dashed threshold line at $\eta = 2$. The shading corresponds to periods when NimbusCC is in delay-control mode. Shaded regions correlate well with the periods when the true fraction of elastic traffic is low (e.g., < 0.3), while white regions correlate well with periods when the elastic fraction is high. Unlike Copa, our elasticity detector observes fluctuations in cross traffic *over a period of time in the frequency domain*, and the accuracy is less susceptible to variations in the cross traffic rate. Despite the churn in cross-traffic flows, the overall accuracy of our elasticity detector is over 90%.

Performance with video cross traffic. Video streams, a large fraction of Internet traffic [5], can be application-limited (inelastic) or network-limited (elastic) at different points in time. We compare the performance of congestion-control algorithms running against cross traffic consisting of a 4k

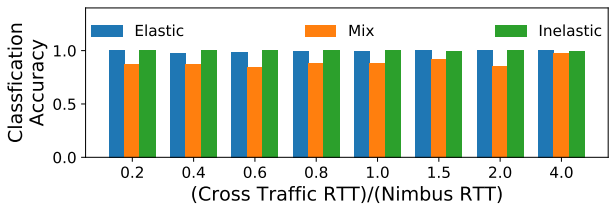


Figure 11: Nimbus classifies purely elastic and inelastic traffic with accuracy greater than 98%. For a mix of elastic and inelastic traffic, the average accuracy is greater than 85% in all cases.

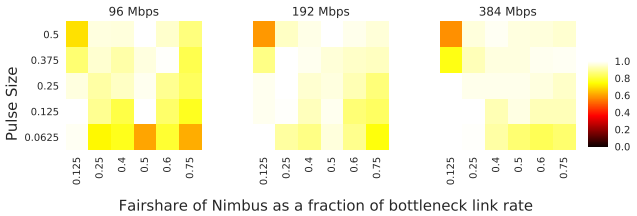


Figure 12: Nimbus is robust to variations in link bandwidth and fraction of traffic controlled by it. The accuracy is high even when the fraction of traffic under control is small. Increasing pulse size increases robustness.

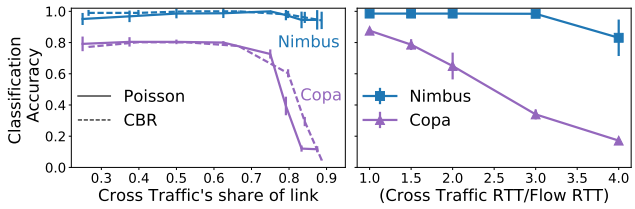


Figure 13: Nimbus's classification accuracy is better than Copa when (i) inelastic cross traffic occupies a large fraction of the link (left); (ii) elastic cross traffic has higher RTT than the flow's RTT (right).

DASH [6] video stream using Cubic on a 48 Mbit/s link with 50 ms propagation RTT. Fig. 10 shows the throughout and delay of the various schemes. Because of effective mode switching, NimbusCC achieves similar throughput as Cubic at 15 ms lower delay. Nimbus recognizes application-limited video traffic as inelastic, allowing the sender to control delays in those cases; it rarely recognizes network-limited elastic traffic as inelastic, so does not wrongly reduce its rate as Copa does.

8.2 Robustness of Elasticity Detection

We evaluate the robustness of Nimbus under a variety of network and traffic conditions. Unless specified otherwise, we run NimbusCC as a backlogged flow on a 96 Mbit/s bottleneck link with a 50 ms propagation RTT and a 100 ms drop-tail buffer (2 BDP). We consider three categories of synthetic cross-traffic sharing the link with NimbusCC: (i) inelastic Poisson-distributed traffic; (ii) fully elastic traffic (backlogged NewReno flows); and (iii) an equal mix of inelastic and elastic traffic. The duration of each experiment is 120 seconds. We evaluate *accuracy*: the fraction of time Nimbus correctly detects the presence of elastic cross-traffic. For each experiment, we report the mean accuracy of the detector across 5 runs.

Impact of cross-traffic RTT. We vary the cross traffic's minimum RTT from 10 ms to 200 ms (0.2 – 4× NimbusCC's RTT). Fig. 11 shows the mean detection accuracy for each of the three classes of cross traffic. We find that varying cross-traffic RTT does not reduce accuracy. Regardless of the cross-traffic RTT, the elastic flows respond to fluctuations created by Nimbus, generating a peak in the cross-traffic FFT at the oscillation frequency. The cross traffic's RTT *affects the phase, but not the amplitude* of the peak in the FFT.

For purely inelastic and purely elastic traffic, the accuracy is more than 98% in all cases, while for mixed traffic, the accuracy is more than 85% in all cases (a random guess would have only achieved 50%). The accuracy for mixed traffic is lower because only half the cross traffic oscillates, and the FFT peak is smaller.

A mix of RTTs in the cross traffic. We vary the number of elastic cross traffic flows from 1 to 5, where the RTT of n^{th} flow in $20 \cdot n$ ms. In case the cross-traffic contains elastic flows, all the elastic flows oscillate at Nimbus's pulse frequency. As a result, the sum of the rates of these elastic flows also oscillates,[4] and the traffic is correctly classified as elastic. For purely elastic and inelastic traffic, Nimbus achieves an average accuracy of 98% across 5 runs, while for mixed traffic, the mean accuracy is greater than 90% in all cases. In other words, heterogeneity in RTTs of cross-flows does not degrade the accuracy of elasticity detection.

Pulse size, link rate, and offered cross-traffic load. We perform a multi-factor experiment varying Nimbus's pulse size from 1/16 to 1/2 the link rate, the fair share of the bottleneck link rate from 12.5%—75% (by varying the cross-traffic load), bottleneck link rates set to 96, 192, and 384 Mbit/s. The accuracy for purely elastic cross-traffic is always higher than 95%. while the average accuracy over all the points for the other two traffic mixes is more than 90%. Fig. 12 shows the average detection accuracy over the other two categories of cross-traffic (mix + purely inelastic). The classification accuracy is not sensitive to cross traffic load. Nimbus's *use of asymmetric pulses* enables a sender to create fluctuations in the cross traffic even when the sending rate is low. As a result, the detection accuracy remains high under high cross-traffic load.

In general, increasing the pulse sizes improves accuracy because the elasticity detector can create a more easily observable change in the cross-traffic sending rates. An increase in the link rate results in higher accuracy for a given pulse size and Nimbus link share because the variance in the rates of inelastic Poisson cross-traffic reduces with increasing cross-traffic sending rate, reducing the number of false peaks in the cross-traffic FFT. However, at low link rates, the elasticity detector has low accuracy (~60%) when it uses high pulse sizes and controls a low fraction of the link rate. We believe that this is due to a quirk in the way the Linux networking stack reports round-trip time measurements under sudden sending rate changes.

[4]Since the RTTs are different, the elastic flows' oscillations will differ in phase and the oscillations could in theory cancel each other out leading to mis-classification, but it requires specific combinations of RTT and is unlikely.

Impact of errors in link rate estimation. We explicitly supply an incorrect link rate estimate to Nimbus. We vary the error in the link rate estimate from −50% to +50% of the real link rate value. The classification accuracy is high (> 80%) for all traffic classes when the error is low (≤ 12.5%). When the error rate is higher, all traffic is classified as elastic.

To understand why, define $\hat{z}(t)$ as the estimate of the cross traffic rate, $z^*(t)$ as the real cross traffic rate, $\hat{\mu}$ as the estimate of the link rate and μ^* as the real link rate. Then, from Equation 1:

$$\hat{z}(t) = \hat{\mu}\frac{S(t)}{R(t)} - S(t), \qquad z^*(t) = \mu^*\frac{S(t)}{R(t)} - S(t) \tag{6}$$

Combining the equations above, we get

$$\hat{z}(t) = \frac{\hat{\mu}}{\mu^*}z^*(t) + \left(\frac{\hat{\mu}}{\mu^*} - 1\right)S(t) \tag{7}$$

When the link estimate is inaccurate, the cross traffic estimate is a linear combination of the real cross traffic rate and the sending rate. As the error increases, the contribution of the sending rate to the cross traffic estimate increases. Since the sending rate oscillates at the pulse frequency, the cross traffic estimate also oscillates, and all cross traffic (regardless of its nature) is classified as elastic.

This implies that when the error in link rate estimate is high, NimbusCC always uses the TCP-competitive mode, losing the benefits of delay control in the presence of inelastic traffic. But its throughput will not suffer.

Buffer size, RTT, and Active Queue Management(AQM). Nimbus is robust to these settings (Appendix E).

Comparison with Copa. We now compare the classification accuracy of Nimbus with Copa. First, we generate inelastic cross traffic at different rates and measure the accuracy.We use a 96 Mbit/s bottleneck link with a 50 ms propagation delay and a 100 ms drop-tail buffer (2 BDP). We consider both constant-bit-rate (CBR) and Poisson cross traffic.

Fig. 13 (left) shows that Nimbus has high accuracy in all cases, but Copa's accuracy drops sharply when the cross traffic occupies over 80% of the link. This result highlights a pitfall of Copa's approach: setting an operating mode based on the absolute value of queueing delays is problematic. With a high inelastic cross-traffic load, Copa is unable to drain the queue quickly enough (i.e., every 5 RTTs), which throws off its detector. In contrast, the elasticity detector estimates elasticity through delay variations caused by its pulses, and is more robust.

Next, we ran a backlogged NimbusCC or Copa flow competing against a backlogged NewReno flow. We vary the RTT of the NewReno flow between $1-4\times$ the RTT of the NimbusCC/Copa flow. Fig. 13 (right) shows that Copa's accuracy degrades as the RTT of the cross traffic increases; Nimbus's accuracy is much higher, dropping only slightly when the cross traffic RTT is $4\times$ larger than NimbusCC.

An elastic cross-flow with a large RTT increases its rate slowly enough to evade detection by Copa. Therefore, Copa drains the queue as it expects and concludes the absence of non-Copa cross-traffic. This behavior continues until the cross-flow has grown to offer a load close to the link rate, when it starts interfering with Copa's queue draining. By contrast,

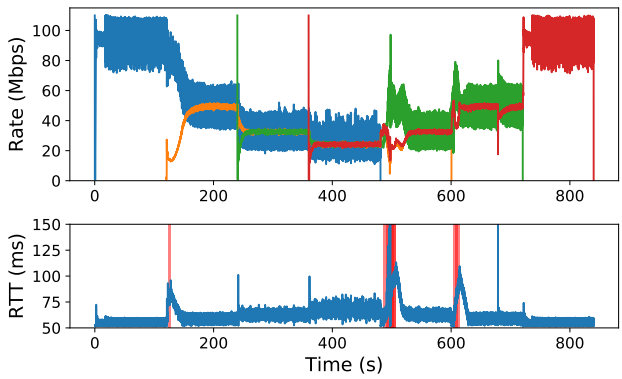


Figure 14: Multiple competing NimbusCC flows. Multiple NimbusCC flows achieve fair sharing of a bottleneck link (top graph). There is at most one pulser flow at any time; identified by its rate variations. Together, the flows achieve low delays by staying in delay mode for most of the duration (bottom graph). The red background shading shows when a NimbusCC flow was (incorrectly) in competitive mode

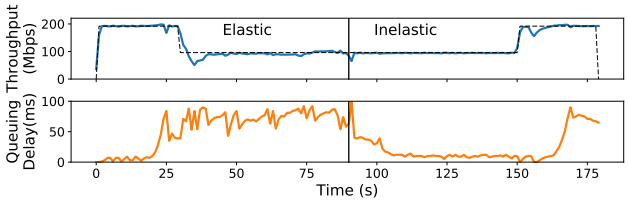


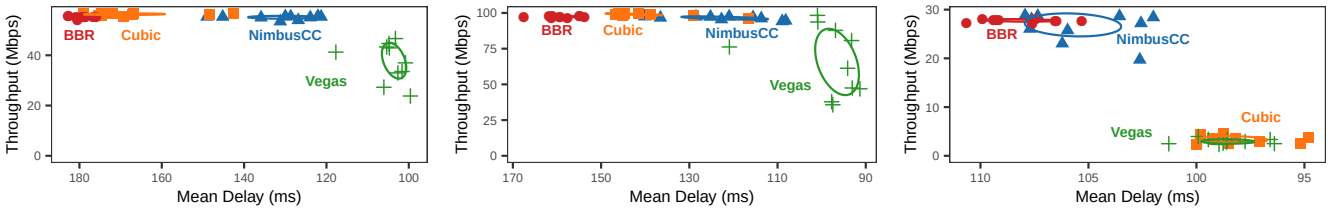
Figure 15: Multiple NimbusCC flows and other cross-traffic. There are 3 NimbusCC flows throughout. Cross traffic in 30-90 s is elastic and made up of three Cubic flows. Cross traffic in 90-150 s is inelastic and made up of a 96 Mbit/s constant bit-rate stream. NimbusCC flows achieve their fair share rate (top) while maintaining low delays in the absence of elastic cross traffic (bottom).

Nimbus is more robust since it is based on the time series of variations of the cross traffic rate. Moreover, even when the classification accuracy for Copa is higher, it makes frequent mode-switches and is suspectible to lose throughput against elastic traffic. Appendix D shows the throughput and queueing delay dynamics of Copa and NimbusCC.

8.3 Fairness With Elasticity Detection

Can multiple flows run elasticity detection and share a bottleneck link fairly with each other and with cross-traffic?

We run NimbusCC with Vegas as its delay-control algorithm. Fig. 14 demonstrates how NimbusCC flows react as other NimbusCC flows arrive and leave (there is no other cross-traffic). Four flows arrive at a link with rate 96 Mbit/s and round-trip time 50 ms. Each flow begins 120 s after the last one began, and lasts for 480 s. The top half shows the rates achieved by the four flows over time. Each new flow begins as a watcher. If the new flow detects a pulser ($t = 120,240,360$ s), it remains a watcher. If the pulser goes away or a new flow fails to detect a pulser, one of the watchers becomes a pulser ($t = 480,720$ s). The pulser can be identified visually by its rate variations.



(a) EC2 California to Host A **(b) EC2 Ireland to Host B** **(c) EC2 London to Host C**

Figure 16: Performance on three example Internet paths. The x axis is inverted; better performance is up and to the right. On paths with buffering and no drops, ((a) and (b)), NimbusCC achieves the same throughput as BBR and Cubic but reduces delays significantly. On paths with significant packet drops (c), Cubic suffers but NimbusCC achieves high throughput.

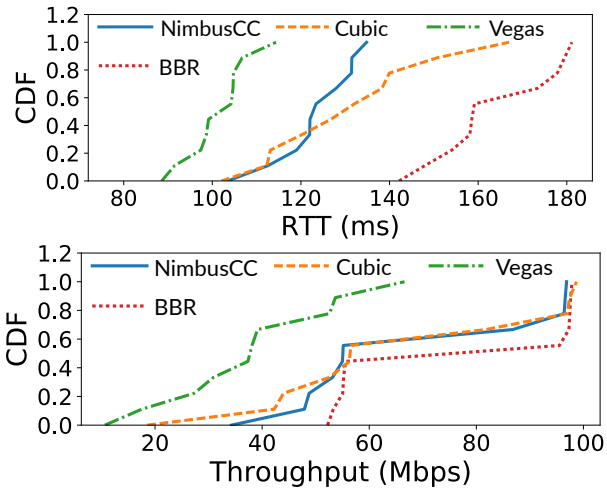


Figure 17: Paths with queuing. NimbusCC reduces the RTT compared to Cubic and BBR (40-50 ms lower), at similar throughput.

The flows share the link rate equally. The bottom half of the figure shows the achieved delays with red background shading to indicate when one of the flows is (incorrectly) in competitive-mode. The flows maintain low RTTs and stay in delay-mode for most of the time.

Fig. 15 demonstrates multiple NimbusCC flows switching in the presence of cross-traffic. We run three NimbusCC flows on an emulated 192 Mbit/s link with a propagation delay of 50 ms. In the first 90 s, the cross-traffic is elastic (three Cubic flows), and for the rest of the experiment, the cross-traffic is inelastic (96 Mbit/s constant bit-rate). The top graph shows the total rate of the three NimbusCC flows, along with a reference line for the fair-share rate of the aggregate. The graph at the bottom shows the measured queuing delays. NimbusCC shares the link fairly with other cross-traffic, and achieves low delays by staying in the delay mode in the absence of elastic cross-traffic.

8.4 Performance on Internet Paths

We ran NimbusCC on Internet paths on 25 paths between five senders and five receivers. The servers were Amazon EC2 instances located in California, London, Frankfurt, Ireland, and Paris, all with 10 Gbit/s links. The receivers were five residential hosts in different autonomous systems. We verified that the bottleneck in each case was not the server's Internet link.

To understand the nature of cross traffic on these paths, we ran experiments with NimbusCC delay-control algorithm (without mode-switching) and Cubic each performing bulk transfers over a three-day period. The results showed that scenarios where cross traffic is predominantly inelastic are common. This indicates that delay-control algorithms can be effective on the Internet (see Appendix A for details).

To understand the nature of cross-traffic on these paths, we initiated bulk data transfers using NimbusCC, Cubic, BBR, and Vegas. We ran one-minute experiments over five hours on each path, and measured the achieved mean throughput and mean delay. Fig. 16 shows throughput and delays over three of the paths. The x (delay) axis is inverted; better performance is up and to the right. NimbusCC achieves high throughput comparable to BBR in all cases, at significantly lower delays. Cubic attains high throughput on paths with deep buffers (Fig. 16a and Fig. 16b), but not on paths with packet drops or policers (Fig. 16c). Vegas attains poor throughput on these paths because it does not keep the bottleneck link busy and is unable to compete with elastic cross-traffic. These trends show the utility of elasticity detection on Internet paths: it is possible to achieve high throughput and low delays over the Internet using delay-control algorithms with the ability to switch to a different competitive mode when required.

Fig. 17 summarizes the results on the paths with queueing. NimbusCC's throughput is similar to Cubic and 10% lower than BBR but at much lower delay (40–50 ms lower than BBR).

9 CONCLUSION

This paper's key contribution is the idea that characterizing the *nature* of cross traffic is a useful signal and building block for congestion control. It introduced a method for detecting and quantifying the *elasticity* of cross traffic. The detection technique uses a carefully constructed asymmetric sinusoidal pulse and observes the frequency response of cross traffic rates at a sender, taking advantage of the property that elastic cross traffic can be made to oscillate at a pulsing frequency set by sender. We presented several experiments to demonstrate the robustness and accuracy of our proposed method. We also showed that elasticity detection enables transport protocols to combine the best aspects of delay-control methods while being competitive with buffer-filling flows when necessary. We found that our proposed methods are beneficial not only on a variety of emulated conditions that model realistic workloads, but also on a collection of 25 real-world Internet paths.

REFERENCES

[1] V. Arun and H. Balakrishnan. Copa: Practical Delay-Based Congestion Control for the Internet. In *NSDI*, 2018.

[2] L. S. Brakmo, S. W. O'Malley, and L. L. Peterson. TCP Vegas: New Techniques for Congestion Detection and Avoidance. In *SIGCOMM*, 1994.

[3] CAIDA. The CAIDA Anonymized Internet Traces 2016 Dataset - 2016-01-21. http://www.caida.org/data/passive/passive_2016_dataset.xml, 2016.

[4] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, and V. Jacobson. BBR: Congestion-Based Congestion Control. *ACM Queue*, 14(5):50:20–50:53, Oct. 2016.

[5] Cisco. Cisco visual networking index: Forecast and methodology, 2016-2021. https://www.cisco.com/c/dam/en/us/solutions/collateral/service-provider/visual-networking-index-vni/complete-white-paper-c11-481360.pdf. June 6, 2017.

[6] DASH Industry Forum. Dynamic Adaptive Streaming over HTTP. https://github.com/Dash-Industry-Forum/dash.js, 2019.

[7] M. Dong, T. Meng, D. Zarchy, E. Arslan, Y. Gilad, B. Godfrey, and M. Schapira. PCC Vivace: Online-Learning Congestion Control. In *NSDI*, 2018.

[8] C. Dovrolis, P. Ramanathan, and D. Moore. What do Packet Dispersion Techniques Measure? In *INFOCOM*. IEEE, 2001.

[9] A. B. Downey. Using pathchar to Estimate Internet Link Characteristics. In *ACM SIGCOMM Computer Communication Review*, volume 29, pages 241–250. ACM, 1999.

[10] W. Feller. *An Introduction to Probability Theory and its Applications*, volume 2. John Wiley & Sons, 2008.

[11] J. Gettys and K. Nichols. Bufferbloat: Dark Buffers in the Internet. *ACM Queue*, 9(11):40, 2011.

[12] S. Ha, I. Rhee, and L. Xu. CUBIC: A New TCP-Friendly High-Speed TCP Variant. *ACM SIGOPS Operating System Review*, 42(5):64–74, July 2008.

[13] M. Hock, R. Bless, and M. Zitterbart. Experimental Evaluation of BBR Congestion Control. In *ICNP*, 2017.

[14] J. C. Hoe. Improving the Start-up Behavior of a Congestion Control Scheme for TCP. In *SIGCOMM*, 1996.

[15] N. Hu and P. Steenkiste. Estimating available bandwidth using packet pair probing. Technical report, DTIC Document, 2002.

[16] N. Hu and P. Steenkiste. Evaluation and Characterization of Available Bandwidth Probing Techniques. *IEEE JSAC*, 21(6):879–894, 2003.

[17] V. Jacobson. Pathchar: A Tool to Infer Characteristics of Internet Paths, 1997.

[18] M. Jain and C. Dovrolis. Pathload: A measurement tool for end-to-end available bandwidth. In *Passive and Active Measurements (PAM) Workshop*, 2002.

[19] H. Jiang and C. Dovrolis. Source-level IP Packet Bursts: Causes and Effects. In *IMC*, 2003.

[20] K. Lai and M. Baker. Measuring Link Bandwidths Using a Deterministic Model of Packet Delay. In *ACM SIGCOMM Computer Communication Review*, volume 30, pages 283–294. ACM, 2000.

[21] K. Lai and M. Baker. Nettimer: A tool for measuring bottleneck link bandwidth. In *USITS*, volume 1, pages 11–11, 2001.

[22] B. Mar. pchar: A tool for measuring internet path characteristics. *http://www. employees. org/˜ bmah/Software/pchar/*, 2000.

[23] A. Narayan, F. Cangialosi, D. Raghavan, P. Goyal, S. Narayana, R. Mittal, M. Alizadeh, and H. Balakrishnan. Restructuring Endpoint Congestion Control. In *SIGCOMM*, 2018.

[24] R. Netravali, A. Sivaraman, S. Das, A. Goyal, K. Winstein, J. Mickens, and H. Balakrishnan. Mahimahi: Accurate Record-and-Replay for HTTP. In *USENIX ATC*, 2015.

[25] R. Pan, P. Natarajan, C. Piglione, M. Prabhu, V. Subramanian, F. Baker, and B. VerSteeg. PIE: A Lightweight Control Scheme to Address the Bufferbloat Problem. In *Intl. Conf. on High Performance Switching and Routing (HPSR)*, 2013.

[26] T. S. Rappaport et al. *Wireless Communications: Principles and Practice*, volume 2. Prentice Hall, 1996.

[27] D. Rossi, C. Testa, S. Valenti, and L. Muscariello. LEDBAT: The New BitTorrent Congestion Control Protocol. In *ICCCN*, 2010.

[28] M. Sridharan, K. Tan, D. Bansal, and D. Thaler. Compound TCP: A New TCP congestion control for high-speed and long distance networks. Technical report, Internet-draft draft-sridharan-tcpm-ctcp-02, 2008.

[29] J. Strauss, D. Katabi, and F. Kaashoek. A Measurement Study of Available Bandwidth Estimation Tools. In *Internet Measurement Conf.*, 2003.

[30] K. Tan, J. Song, Q. Zhang, and M. Sridharan. A Compound TCP Approach for High-speed and Long Distance Networks. In *INFOCOM*, 2006.

[31] A. Tirumala, F. Qin, J. Dugan, J. Ferguson, and K. Gibbs. Iperf: The TCP/UDP bandwidth measurement tool. http://dast.nlanr.net/Projects, 2005.

[32] D. Wei, C. Jin, S. Low, and S. Hegde. FAST TCP: Motivation, Architecture, Algorithms, Performance. *IEEE/ACM Trans. on Networking*, 14(6):1246–1259, 2006.

[33] K. Winstein, A. Sivaraman, and H. Balakrishnan. Stochastic Forecasts Achieve High Throughput and Low Delay over Cellular Networks. In *NSDI*, 2013.

A CROSS TRAFFIC IS OFTEN INELASTIC

Our experiments on 25 Internet paths show that scenarios where cross traffic is predominantly inelastic are common. Figure 18 shows the average throughput and delay for 100 runs of buffer-filling Cubic compared to BasicDelay, a delay-controlling method, on one of these paths. The delay-controlling scheme generally achieves much lower delays than Cubic, with similar throughput. This shows that there is an opportunity to significantly improve delays using delay-controlling algorithms, provided we can detect the presence of elastic cross-traffic flows and compete with them fairly when needed.

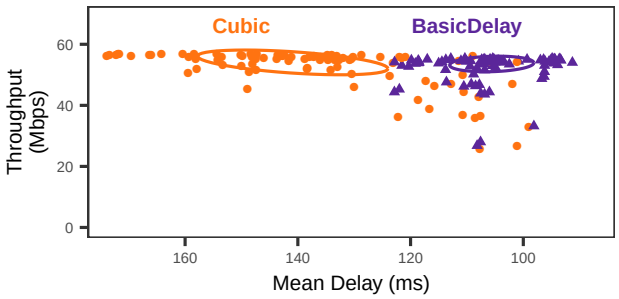


Figure 18: Mean throughput and delay for 100 one-minute data transfers with Cubic and BasicDelay. The experiments were run between an AWS EC2 server in California and a receiver on the US east coast. The ellipses show one standard deviation. BasicDelay achieves the same throughput as Cubic in many runs, signifying an absence of elastic cross traffic in these cases.

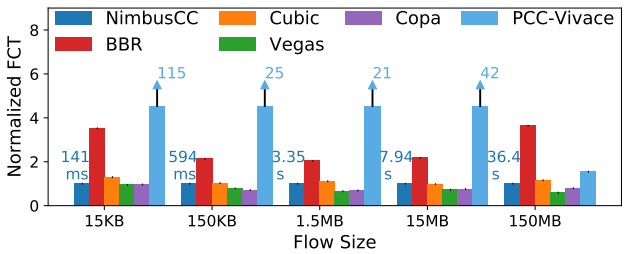


Figure 19: Using NimbusCC reduces the p95 FCT of cross-flows relative to BBR at all flow sizes, and relative to Cubic for short flows. Vegas provides low cross-flow FCT, but its own rate is low.

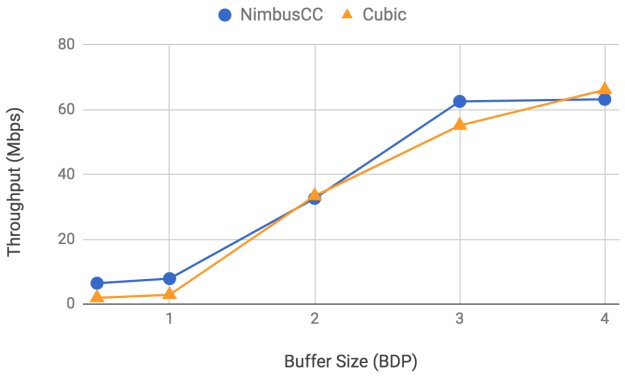


Figure 20: NimbusCC's performance against BBR is similar to that of Cubic. Both NimbusCC and Cubic compete against 1 BBR flow on a 96 Mbit/s link. For various buffer sizes, NimbusCC achieves the same throughput as Cubic.

B NIMBUS HELPS CROSS TRAFFIC

In the setup from §8.1, we measure the flow completion time (FCT) of cross-traffic flows. Fig. 19 compares the 95th percentile (p95) FCT for flows of different sizes. The FCTs are normalized by the corresponding value for NimbusCC at each flow size (i.e., NimbusCC is always 1).

BBR and PCC-Vivace exhibits much higher FCT at all cross-traffic flow sizes compared to the other protocols, consistent with the unfairness seen in the experiment in §5.

For small flows (≤15 KB), the p95 FCT with NimbusCC and Copa are comparable to Vegas and lower than Cubic. With NimbusCC, p95 FCT of cross traffic at higher flow sizes are slightly lower than Cubic because of small delays in switching to TCP-competitive mode. At all flow sizes, Vegas provides the best cross-traffic flow FCTs, but its own flow rate is dismal; Copa is more aggressive than Vegas but less than NimbusCC, but at the expense of its own throughput (§8.1).

C NIMBUSCC & CUBIC V. BBR

We now evaluate how well a NimbusCC (Cubic + BasicDelay) flow competes with a BBR flow. In this experiment, the cross traffic is 1 BBR flow and the bottleneck link bandwidth is 96 Mbit/s. We vary the buffer size from 0.5 BDP to 4 BDP. Fig. 9 shows the mean throughput of NimbusCC and Cubic flows while competing with BBR over a 2-minute experiment. NimbusCC achieves the same throughput as Cubic for all buffer sizes.

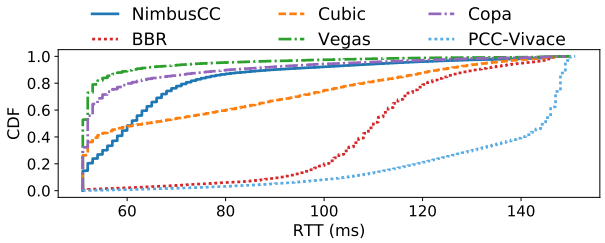
In this experiment, when the buffer size is ≤ 1 BDP, BBR is not ACK-clocked, and the elasticity detector classifies it as inelastic traffic. As a result, NimbusCC gets a relatively small fraction of the link bandwidth. In this scenario, Cubic also gets a small fraction of the link, because BBR sends traffic at its estimate of bottleneck link and is too aggressive.

When the buffer size is ≥1 BDP, BBR becomes ACK-clocked because of the cap on its congestion window. The elasticity detector now classifies BBR as elastic traffic. NimbusCC stays in competitive mode, so NimbusCC behaves like Cubic.

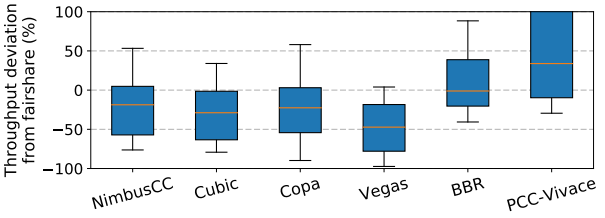
WAN-cross traffic: We repeated the experiment in Fig. 7, with cross traffic flows running BBR. Fig. 21 shows the performance of the various schemes. NimbusCC's deviation profile is similar to that of Cubic, but NimbusCC achieves lower delays. However, both NimbusCC and Cubic achieve lower throughput than the fairshare, this is because Cubic doesn't compete will with BBR cross traffic. Copa and Vegas have similar delays as NimbusCC, but their tail throughput is lower. PCC-Vivace is unfair to the cross traffic, and sends more than its fair-share.

D COPA'S MODE-SWITCHING ERRORS

We explore the dynamics of NimbusCC and Copa's mode switching in experiments from the scenarios in §8.2.



(a) Per-packet RTT



(b) Deviation from fairshare throughput

Figure 21: WAN cross-traffic. Cross traffic flows are running BBR. The deviation profile of NimbusCC is similar to that of Cubic, however, NimbusCC reduces delays.

D.1 CBR Cross Traffic

Fig. 22 shows throughput and delay profile for Copa and NimbusCC while competing against inelastic CBR traffic. We consider two scenarios: (i) CBR occupies a small fraction of the link (24 Mbits/s, 25%) and (ii) CBR occupies majority of the link (80 Mbit/s, 83%). When the CBR traffic is low (Fig. 22 a and Fig. 22 b), both Copa and Nimbus identify it as non-buffer-filling and inelastic, respectively, and achieve low queuing delays.

When the CBR's share of the link is high (Fig. 22 c), Copa incorrectly classifies the cross traffic as buffer-filling and stays in competitive mode, leading to high queuing delays. Copa relies on a pattern of emptying queues to detect whether the cross traffic is buffer-filling or not. However, when the rate of cross traffic is z, the fastest possible rate at which the queue can drain is $\mu - z$, even if Copa reduces its rate to zero. If the cross traffic occupies x fraction of the link (i.e., $z = x\mu$), then

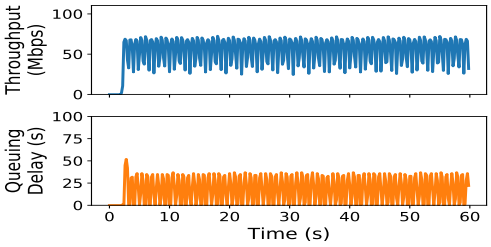
$$\max(-\frac{dQ}{dt}) = \mu - z = (1-x)\mu = (1-x)\frac{BDP}{RTT}. \quad (8)$$

Hence, if the queue size exceeds $5\times(1-x)BDP$, Copa won't be able to drain the queue in 5 RTTs, and it will mis-classify the cross traffic as buffer-filling. The queue size can grow large due to a transient burst or if Copa incorrectly switches to competitive mode. Once Copa is in competitive mode, it will drive the queues higher, and may get stuck in that mode.

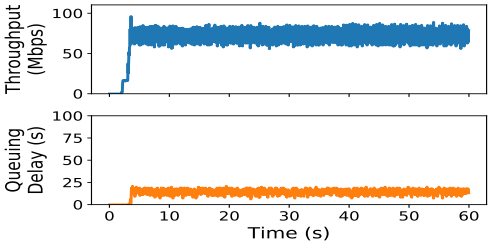
Nimbus doesn't rely on emptying queues and correctly classifies cross traffic as inelastic, achieving low delays (Fig. 22 d).

D.2 Elastic cross traffic

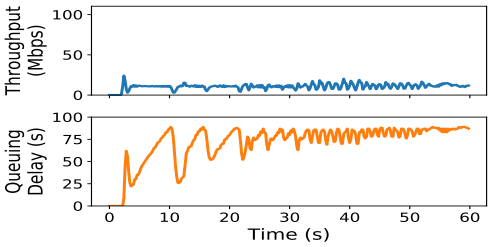
Fig. 23 shows throughput and delay over time for Copa and NimbusCC while competing against an elastic NewReno flow. We consider two scenarios: (1) both flows have the same propagation RTT, and (2) the cross traffic's propagation RTT is 4× higher than the Copa or NimbusCC flow. When the RTTs



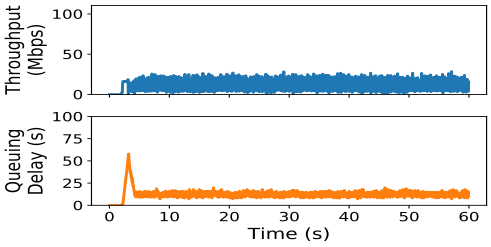
(a) Copa: 24 Mbit/s CBR



(b) NimbusCC: 24 Mbit/s CBR



(c) Copa: 80 Mbit/s CBR

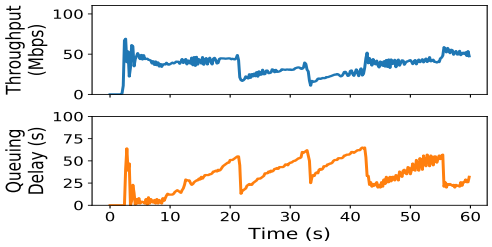


(d) NimbusCC: 80 Mbit/s CBR

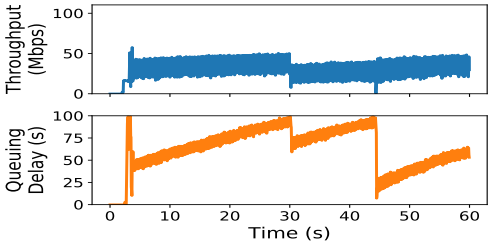
Figure 22: When the CBR traffic is low (a), Copa classifies the traffic as non buffer-filling and is able to achieve low queuing delays. But when the CBR traffic occupies a high fraction (c), Copa incorrectly classifies the traffic as buffer-filling, resulting in higher queuing delays. In both the situations (b and d), the elasticity detector correctly classifies the traffic as inelastic and NimbusCC achieves low queuing delays.

are the same (Fig. 23 a and Fig. 23 b), both Copa and Nimbus correctly classify the cross traffic, achieving their fair share.

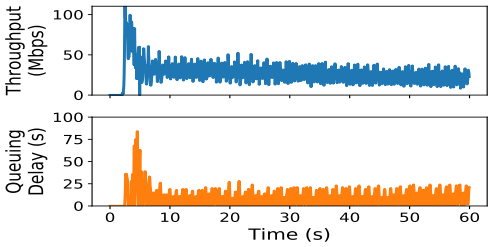
When the cross traffic RTT is higher (Fig. 23 c), NewReno ramps up its rate slowly, causing Copa to mis-classify the traffic and achieve less than its fair share. Here, Copa achieves 27 Mbit/s but its fair share is at least 48 Mbit/s (in fact, 77 Mbit/s considering the RTT bias). In contrast, (Fig. 23 d), Nimbus correctly classifies the cross traffic as elastic, and NimbusCC achieves its RTT-biased share of throughput.



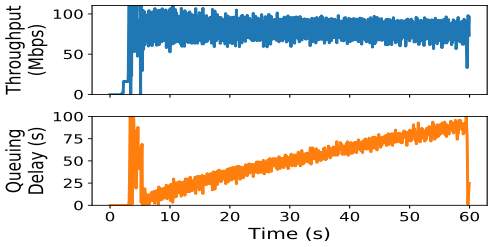
(a) Copa: Cross Traffic RTT = 1 × Flow RTT



(b) NimbusCC: Cross Traffic RTT = 1 × Flow RTT



(c) Copa: Cross Traffic RTT = 4 × Flow RTT



(d) NimbusCC: Cross Traffic RTT = 4 × Flow RTT

Figure 23: Queuing delay and throughput dynamics for elastic cross traffic. When the elastic cross traffic increases fast enough (a), Copa classifies it as buffer-filling and is able to achieve its fair share. But when the elastic cross traffic increases slowly (c), Copa incorrectly classifies the traffic as non-buffer-filling, achieving less than its fair share. In both the situations (b and d), Nimbus correctly classifies the traffic as elastic and NimbusCC achieve its fair share.

E BUFFER SIZE, RTT, AND AQM

We vary the bottleneck drop-tail buffer size from 0.25 BDP to 4 BDP for three categories of cross traffic as in the earlier experiments, with propagation delays of 25 ms, 50 ms, and 75 ms. We also measured classification accuracy when the bottleneck link implements PIE [25] at two target delays (0.25 BDP and 1 BDP) with a propagation delay of 50 ms. With purely elastic or inelastic traffic, Nimbus has a mean accuracy (across five runs) of 98% or more in all cases but two, while with mixed traffic, the accuracy is always 85% or more. In all cases (including low accuracy ones), NimbusCC achieves its fair-share throughput and low delays.

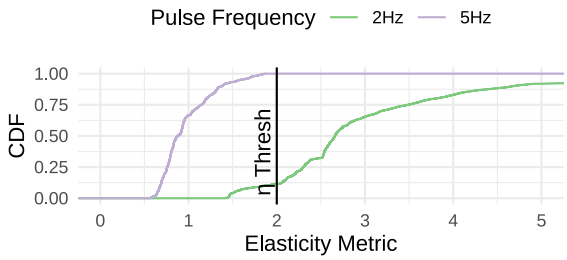


Figure 24: By modifying the pulse frequency, Nimbus correctly classifies PCC-Vivace, a rate-based elastic protocol, as elastic.

Now we discuss the cases with low classification accuracy. First, with shallow buffers of size less than the product of the delay threshold x_t and the bottleneck link rate (e.g., 0.25 BDP when the round-trip time is 50 ms), Nimbus classifies all traffic as elastic. Second, with the bottleneck link implementing PIE with small target delay (e.g., corresponding to 0.25 BDP), Nimbus classifies all traffic as elastic. In both cases, NimbusCC can incur heavy losses in delay-control mode as NimbusCC's target queuing delay of 0.25 BDP is comparable to the drop-tail buffer size or target delay of PIE. These losses interfere with the cross-traffic estimator leading to classification errors (in delay-control mode). However, low accuracy does not impact the performance of NimbusCC as it achieves its fair-share throughput and low delays (bounded by the small buffer size for a drop-tail queue and the delay control threshold of PIE). Further, classification accuracy decreases when Nimbus's RTT exceeds its pulse period. Since Nimbus's measurements of rates are over one RTT, any oscillations over a smaller period cannot be observed.

F ELASTIC FLOWS, NO ACK CLOCKING

Nimbus aims to detect ACK-clocked elastic flows that react quickly to changes in available bandwidth on RTT timescales. This experiment demonstrates Nimbus's ability to also detect slow-reacting elastic cross traffic by tuning the pulse frequency. We ran a NimbusCC flow against a PCC-Vivace flow on a 96 Mbit/s link with 100ms of buffering. Fig. 24 shows the CDF of the elasticity metric, η, for two different pulse frequencies, f_p. PCC-Vivace is not ACK-clocked and does not react to Nimbus's pulses at $f_p = 5$ Hz. As a result η is below the threshold most of the time. Reducing the pulse frequency to 2 Hz creates pulses with a longer duration. PCC-Vivace reacts to these slower variations in available bandwidth, and is correctly classified as elastic ($\eta > \eta_{thresh}$).

Changing the pulse frequency involves a trade-off. Increasing the pulse duration will increase queuing delays and congestion. But if slowly-reacting elastic protocols become widely deployed, competing with them using Nimbus for delay-control opportunities will require an increase in pulse duration.