In Proceedings of IEEE INFOCOM 2007.

When TCP Friendliness Becomes Harmful

Amit Mondal and Aleksandar Kuzmanovic
Northwestern University
{a-mondal, akuzma}@cs.northwestern.edu

***Abstract*— Short TCP flows may suffer significant response-time performance degradations during network congestion. Unfortunately, this creates an incentive for misbehavior by clients of *interactive applications* (*e.g.*, gaming, telnet, web): to send "dummy" packets into the network at a *TCP-fair* rate even when they have no data to send, thus improving their performance in moments when they do have data to send. Even though no "law" is violated in this way, a large-scale deployment of such an approach has the potential to seriously jeopardize one of the core Internet's principles — *statistical multiplexing*. We quantify, by means of analytical modeling and simulation, gains achievable by the above misbehavior. Further, we explore techniques that both misbehaving and regular clients can apply to optimize their performance. Our research indicates that easy-to-implement application-level techniques are capable of dramatically reducing incentives for conducting the above transgressions, still without compromising the idea of statistical multiplexing.**

I. Introduction

It is well known that short TCP flows may experience significant performance degradations when they multiplex with long-lived TCP flows [1]. The root of the problem is the lack of knowledge about the level of the underlying network congestion. In absence of the large number of packets characteristic for long-lived flows, even a single packet loss can force a short-lived TCP flow to experience long retransmission timeouts [2], which in turn significantly increase a client's perceived response time. While several solutions have been proposed to efficiently combat the problem, none has been deployed in the Internet, probably because they require non-negligible architectural changes [1], [3], [4].

However, one extremely relevant — and imminent — aspect of this problem is still unexplored. In essence, TCP-based *interactive applications* such as gaming [5], telnet, or persistent HTTP [6], which *share* the above problem common for short flows, have incentive to improve their performance; still, without waiting for any Internet-wide architectural changes. In particular, they can "upgrade" themselves from "mice" to "elephants" in a trivial way, simply by sending packets into the network at a *TCP-fair rate* even when they have nothing to send. In this way, they become capable of developing larger congestion windows, avoid "loosing memory" in moments of application-level data starvation [7], and improve their performance by avoiding long retransmission timeouts.

While it may appear that this is a minor problem, or even there is no problem at all (given that all flows are TCP friendly), this is far from being the case. A large-scale deployment of this approach has the potential to seriously jeopardize one of the core principles that today's Internet is built upon — *statistical multiplexing*. Indeed, if everybody would start taking their own fair bandwidth share, the network would soon become highly congested. While the *absolute* performance of *all* flows would necessarily degrade in such a case, a troubling observation is that *those applying the "padding" misbehavior would still benefit relative to the regular clients. Hence, the dangerous incentive remains.*

Unfortunately, upgrading an interactive to a fully-backlogged flow is easy to implement, both at the TCP and the application levels. Indeed, client-side *only* implementations could dramatically improve user-experienced response times, still without requiring *any* changes at servers. Moreover, inciting servers to send traffic at TCP-fair rates is not impossible [8]. In all scenarios, both network- and endpoint-based mechanisms that check for TCP-friendliness, *e.g.*, [8]–[10], are incapable of detecting any violation, simply because all flows are TCP friendly.

To understand all aspects of the above problem, we conduct an extensive modeling and simulation analysis. By combining and extending the modeling results of [11]–[14], we quantify the response-time gains that fully-backlogged flows achieve over the interactive ones. Our results show that the expected response times of fully-backlogged flows can be *two to three times* smaller than those of interactive ones. Likewise, gains achievable by fully-backlogged TCP flows are much more pronounced in the case of Random Early Drop (RED) queues. Even if a packet is dropped at a RED bottleneck in the network, the probability is high that at least three of the follow-up packets will trigger the triple-duplicate ACK mechanism, thus avoiding long retransmission timeouts. Because Drop Tail queues invoke *correlated* packet losses, the corresponding gain is smaller.

Next, our research indicates that there exists a "sweet spot" system state for misbehaving clients. It is the packet loss ratio for which the fully-backlogged clients maximize their response-time gain *relative* to interactive flows. While the optimal point is a function of various system parameters, such as round-trip time (RTT) and the queuing discipline at the bottleneck link, we explore ways that misbehaving clients can apply to drive the system to the desired state.

Further, we explore techniques that regular clients can apply to mitigate the problem. Given the inherent deployment issues with network-based solutions [1], [3], [4], we focus on *endpoint* based methods. We initially explore a TCP-level approach of reducing the retransmission timeout parameter by a half. Despite evident improvements, both our modeling and simulation results indicate that the method is incapable of removing the dangerous incentive for misbehavior.

We further explore two other endpoint techniques to address the problem: (i) short-term padding, and (ii) a diversity approach. In the first scenario, applications append a small number of small-sized packets to data bursts, thus increasing the probability to invoke the triple-duplicate ACK mechanism. In the second scenario, TCP endpoints repeat their packets: if at least one reaches the destination, the response time is small. Surprisingly, our modeling and simulation results indicate that not a single approach is superior, and that the queuing discipline (*e.g.*, RED vs. Drop Tail) again dominantly impacts the system performance.

Finally, our results clearly show that both endpoint techniques outperform the fully-backlogged approach, thus effectively *removing* the dangerous incentive for the greedy TCP-friendly behavior. While various sub-versions of the proposed application-level techniques could themselves become attractive options for misbehaving clients, this no longer poses a threat to the Internet. Indeed, we show that even if *all* interactive-application clients deploy one of the proposed approaches, the overall network performance does not change dramatically. Thus, the statistical-multiplexing benefits remain available to all network clients.

II. Problem Origins and Implications

A. Problem Origins

TCP congestion control operates at two timescales. On smaller time scales of the order of RTTs, TCP performs additive-increase multiplicative-decrease (AIMD) control with the objective of having each flow transmit at the fair rate of its bottleneck link. At times of severe congestion in which multiple losses occur, TCP operates on longer timescales of Retransmission Time Out (RTO). It provides two mechanisms for packet loss detection: Fast Retransmit and timeout.

TCP interprets receipt of three duplicate ACKs as an indication of a packet loss. It retransmits the lost packet immediately upon the receipt of the third duplicate ACK. This mechanism is called Fast Retransmit; it detects a packet loss and reacts to it on the order of a flow's RTT. Another mechanism to detect a packet loss is the timeout mechanism. TCP sender starts a retransmission timer when it sends a packet. In case it receives less than three duplicate ACKs and the timer expires, the sender retransmits the packet. The initial RTO value is set to three seconds [2]. It has been experimentally shown that TCP achieves near-maximal throughput if there exists a lower bound for RTO of one second [2], [15].

The main reasons for the response-time performance degradations experienced by short TCP flows is their poor knowledge about the actual level of congestion in the network. Indeed, given that such flows only have a few packets to send, in case a packet gets lost in the network, they have no other option but to wait for the RTO to expire. In other words, they are unable to resend the packet immediately after one RTT, because the three duplicate ACKs may never return; simply because the corresponding data packets were never sent by the sender. Given that RTTs are typically of the order of 10's to 100's of msec, each such event degrades the response time for approximately one to two *orders of magnitude*.

While the above effect has mainly been explored in the context of web traffic [16], [17], the same problem holds for *interactive* applications [4]. In such scenarios, a client typically sends a small burst of data, and then waits for a longer period of time (*e.g.*, a few seconds) before sending the next burst. One additional issue with interactive scenarios is that even if an application manages to develop large congestion windows during burst periods, it cannot "freeze" the window during times when no data is coming from the application, and reuse it afterwards. Indeed, because the network conditions may change quickly, TCP endpoints are required to reduce their congestion windows during periods of data starvation [7].

B. Proposed Solutions

Several solutions based on the idea of service differentiation and preferential treatment to short flows in the network are proposed to address the above problem. Guo and Matta [1] use different marking/dropping functions at the routers and a packet classifier at the network edge to distinguish between long- and short-lived TCP flows. In addition to requiring large changes to the existing network infrastructure, the solution appears to address the problem of short, but not the interactive flows. Noureddine and Tobagi [4] propose application- and TCP-level marking to give strict priority to interactive applications in the network. In addition to requiring per-user traffic policing at the network edge (tedious to deploy), the authors assume a widespread network support for multi-prioirty services in the Internet (to the best of our knowledge, not the case).

Le *et al.* [3] propose an AQM scheme which gives a strict priority to short flows, while it applies congestion control only to long flows. The key advantage over the above two schemes is that it requires no support from the endpoints; it distinguishes short from long flows by tracking the number of packets that have recently been seen from each flow at the router. In addition to provoking potential security and stability side effects (*e.g.*, see [17]), the proposed scheme requires to be implemented in the *network core*; unfortunately, no strong incentives for such a deployment exist. Similarly, it has been shown that marking, instead of dropping, TCP control packets using Explicit Congestion Notification (ECN) could significantly improve the performance of short flows [17]. Unfortunately, ECN is poorly deployed in today's Internet.

Endpoint-based approaches have also been proposed. To address the problem of low network observability by short flows, RFC 2414 [18] allows the initial congestion window of two segments, while RFC 3390 [19] further allows the use of four segments. If at least one of the packets returns to the sender, the connection will not suffer the initial default 3 second-long timeout penalty [2]. Yang and de Veciana [20] develop TCP/SAReno in which the AIMD parameters dynamically depend on the remaining file size, such that short flows become more aggressive. Finally, Savage *et al.* [21] and

Anderson *et al.* [22] have demonstrated that using history can be efficiently used to improve the performance of short flows.

Despite the fact that all of the above endpoint approaches enable protocol support for improving the performance of short or interactive flows, the key problem remains: the *application-level data starvation* can prevent clients from experiencing any benefits from the above designs. In particular, the burst periods of interactive flows are typically small enough to fit into a *single packet* [23], [24]. As a result, an increased congestion window, a more aggressive TCP, or a history-based approach *cannot* help. If a packet gets lost in the network, the sender must rely on the RTO mechanism before re-injecting the packet back into the network, thus experiencing significant performance degradations.

C. Implications

The unsolved status of the above problem creates a dangerous incentive for clients of interactive applications to quickly solve the problem without anybody's support. The logic is simple: if interactive flows experience performance degradations relative to long TCP flows, then why not upgrading interactive to long flows? Clients can simply send packets into the network even when they have no data to send *at a TCP fair rate*, thus improving their performance in moments when they do have data to send. Figure 1 depicts this approach. Whenever data packets are available, they are immediately sent (hence, strict priority); in times of application-level data starvation, "dummy" packets are sent into the network.

Incentives for clients to apply this approach are manifold. First, by sending dummy packets into the network, clients avoid loosing memory in moments of data starvation [7]. Larger congestion windows can help "jumpstart" an actual data burst arriving from the application. Second, "dummy" packets following data packets may significantly increase the probability that a potential packet loss will be detected via the triple-duplicate ACK mechanism rather than the RTO. Finally, clients can freely apply this approach, without any fear of "getting caught." This is because both network- and endpoint-based schemes designed to check for TCP-fairness compliance (*e.g.*, [8]–[10]) would detect no violations.

Unfortunatelly, even though no law is officially broken with the above approach, its wide-spread adoption has a strong potential to seriously jeopardize the overall Internet performance. Indeed, if interactive clients would start taking their bandwidth fair-share, the network would soon become highly congested. The packet-based Internet as we know it would soon become a "circuit-based" network; given the large number of short and interactive flows [16], [25], the bandwidth "dedicated" to each "TCP-friendly circuit" would soon converge to *zero* [26]. Still, our research indicates that *even in such scenarios, misbehaving clients would outperform the behaving ones.* Thus, the dangerous incentive remains.

Finally, implementing the approach of Figure 1 is not particularly challenging. Client-side *only* implementations, both at the TCP and the application levels are straight forward. Such designs could improve the times required to "push" packets

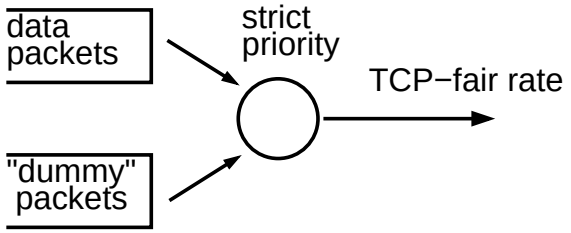


Fig. 1. Padding misbehavior: Upgrading mice to elephants.

to servers, a feature of particular interest to online gaming players. (Many online games require reliable transport, and hence use TCP ports [5]). While slightly more challenging, provoking servers to send at TCP-friendly rates is not impossible. One example is a recently proposed mobile TCP code method [8]. It enables clients to deploy a desired TCP version at servers. Given that it only checks for TCP friendliness, the approach of Figure 1 would not be qualified as a violation.

III. Padding-Induced Response-Time Gains: Modeling and Simulation

Here, we quantify the gain a misbehaving client is able to achieve by applying the padding approach. The key performance metric is the *response time*, defined as the time that elapses between sending a data packet into the network and receiving a corresponding acknowledgement. To establish a baseline for comparisons, we initially model the performance of pure interactive flows. Next, we model the response times achievable by fully-backlogged flows, assuming both random and correlated packet losses in the network. Finally, we verify our modeling results via simulation.

A. Modeling Response Times of Application-Limited TCP Flows

Interactive applications are characterized by two parameters: the data burst size, and the inter-burst arrival time. Both parameters are dependent on human behavior and activities, such as the user think times or the typing speed. The burst sizes are typically small, and they easily fit into a single packet [23], [24]. The inter-burst arrival times differ from application to application. They are typically modeled by the exponential distribution, with the mean of several hundreds of milliseconds (*e.g.*, for gaming [23]) to several seconds (*e.g.*, telnet [24]). In any case, as long as the inter-packet arrival times are longer than one third of the RTO, a potential packet loss will *not* trigger the triple-duplicate ACK mechanism, but will rather be detected via the RTO.

Thus, assuming single-packet-long data bursts and the RTO-based packet-loss detection, we proceed as follows. Denote by p the packet loss probability. Let $P_h(i)$ be the probability that a packet experiences exactly i failure transmission attempts, followed by one successful try. Then,

$$P_h(i) = p^i(1-p). \tag{1}$$

After the timeout expires, the client doubles the current value of RTO; thus, after i consecutive packet losses, the RTO value is set to $2^i RTO$. Denote by $L(i)$ the corresponding

latency experienced by the client after i failure transmission attempts. $L(i)$ can be expressed as

$$L(i) = \sum_{k=0}^{i-1} 2^k RTO + RTT = \left(2^i - 1\right) RTO + RTT. \quad (2)$$

Thus, for $p < 0.5$, the *expected value* of the response-time latency becomes

$$E[L] = \sum_{i=0}^{\infty} P_h(i) L(i) = RTO\left(\frac{(1-p)}{(1-2p)} - 1\right) + RTT. \quad (3)$$

B. Modeling Response Times of Fully-Backlogged TCP Flows

Here, we model the response times of fully-backlogged network-limited TCP flows. By establishing this result, we become capable of understanding gains that a misbehaving client can achieve by applying the padding approach. We exploit the sophisticated modeling results of [11], [14], and further extend them to obtain the desired response-time characteristics. In our analysis, we consider both correlated and random packet losses, typical for FIFO and RED routers, respectively.

1) Correlated Packet Losses: Padhye *et al.* [14] develop the well-known TCP throughput model for fully-backlogged TCP flows, which we exploit to obtain the response-time characteristic. We use the same notation and preserve all relevant assumptions of [14]. From our perspective, the most important is the *correlated packet loss* assumption. It says that if a packet is lost, so are all the following packets within the same RTT round. Indeed, when the bottleneck router applies FIFO (DropTail) queuing, this is likely the case.

Denote by b the number of packets acknowledged by each ACK. Denote by w the TCP congestion window size, and by $E[w]$ its expected value. Then, according to [14], $E[w]$ becomes

$$E[w] = \frac{2+b}{3b} + \sqrt{\frac{8(1-p)}{3bp} + \left(\frac{2+b}{3b}\right)^2}. \quad (4)$$

Next, for a given w, denote by $\hat{Q}(w)$ the probability that a loss is indicated via a timeout. According to [14],

$$\hat{Q}(w) = min\left(1, \frac{(1-(1-p)^3)(1+(1-p)^3(1-(1-p)^{(w-3)}))}{1-(1-p)^w}\right). \quad (5)$$

Q, the probability that a loss indication is a timeout is,

$$Q = \sum_{w=1}^{\infty} \hat{Q}(w) P(W = w) = \hat{Q}\left(E[w]\right). \quad (6)$$

Consequently, the probability that the sender detects a packet loss via triple duplicate ACKs is given by $1 - Q$.

2) Random Packet Losses: Brosh *et al.* [11] show that the above model underestimates the fast retransmit and fast recovery TCP features when routers deploy RED. Because in such scenarios packet losses are random, rather than correlated, the loss recovery probability increases and the subsequent loss recovery latency decreases. Thus, by adopting the Bernoulli loss model, the assumption is that each packet in a round is dropped with probability p, *independently* of other packets. Let $B(w,k) = \binom{w}{k} p^{w-k}(1-p)^k$. Then, according to [11], for a given w, and for $w > 3$, the probability that a loss indication is a timeout is given by

$$\hat{Q}(w) \leq \frac{\sum_{k=0}^{2} B(w,k)(1+(1-p)^k(-1+(2-p)^w))}{1-(1-p)^w}. \quad (7)$$

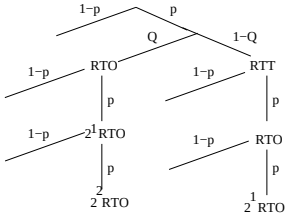


Fig. 2. Decision Tree

Next, considering a uniform distribution of the TCP congestion window W, on the discrete interval $[0, w_{max}]$; the probability that a loss indication is a timeout becomes

$$Q = \sum_{w=1}^{w_{max}} \hat{Q}(w) P[W = w] \approx min(1, \frac{1}{w_{max}}(6 + 96p - 32p^2 + o(p^3))). \quad (8)$$

Again, the probability that the sender detects a packet loss via triple duplicate ACKs is given by $1 - Q$.

3) Response Times: Finally, we compute the response times for both of the above scenarios. One important issue here is that TCP *always* evokes an RTO if a retransmitted packet is lost again [13]. All versions of TCP, including NewReno and SACK, cannot recover from a retransmission loss without a retransmission timeout. Figure 2 depicts this effect. Once a packet is lost (with probability p), the triple-duplicate ACK mechanism will be invoked with probability $1 - Q$. However, if the packet is lost more than once, the RTO is inevitable. While it may appear that computing Q is not that essential (given that it appears only once in the decision tree), this is not the case. Given that the Q branch is close to the root of the tree, it does impact the response times in a non-trivial way, as we demonstrate below.

For a fully backlogged TCP connection, denote by $L'(i)$ the latency experienced by the client after exactly i failure transmission attempts of a packet, followed by a successful transmission. Using the decision tree of Figure 2, we derive $L'(i)$ as

$$L'(i) = \begin{cases} RTT & \text{for } i = 0, \\ RTT + Q(2^i - 1)RTO + & \\ (1-Q)(RTT + (2^{i-1} - 1)RTO) & \text{for } i \geq 1. \end{cases} \quad (9)$$

Consequently, the expected value of the response-time latency becomes

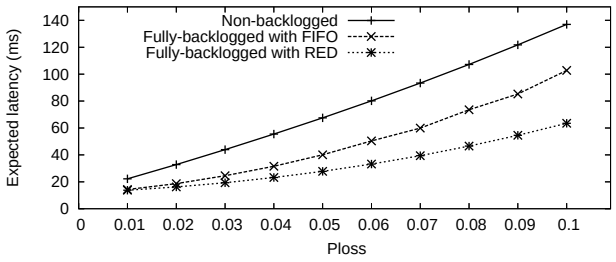


Fig. 3. Modeling: Expected latency as a function of packet loss prob.

$$E[L'] = \sum_{i=0}^{\infty} P_h(i)L'(i)$$
$$= Q\left(\frac{1-p}{1-2p} - 1\right) RTO + RTT$$
$$+ p(1-Q)\left(\left(\frac{1-p}{1-2p} - 1\right) RTO + RTT\right). \quad (10)$$

Finally, we define the response time gain, G, as the ratio between the expected response times for an interactive and a fully-backlogged TCP, $G = E[L]/E[L']$.

C. Modeling Results

Figure 3 depicts the expected latency as a function of the packet loss probability for application-limited as well as fully-backlogged flows (both for random and correlated packet losses). Naturally, in all scenarios, the expected latency increases as the packet loss probability increases. However, the key point is that for a given packet loss rate, the fully-backlogged flows *always* outperform interactive ones. In other words, clients promoting their flows from mice to elephants *always* experience better performance than pure interactive flows. Unfortunately, this means that the incentive for conducting the misbehavior is always present.

Figure 3 further shows that the padding misbehavior pays off better for RED-based bottlenecks. Because packet losses are random, avoiding RTOs is more likely in such scenarios. In particular, if a packet is lost, the probability that the following packets from the same RTT round will make it to the destination (and the corresponding ACKs back to the source) is not small. As a result, the triple duplicate ACK probability $(1 - Q)$ is larger for random packet losses than for correlated ones. Figure 3 demonstrates that there still exists gain of fully-backlogged flows with FIFO over the pure interactive scenario; this is despite the correlated packet loss assumption (if a packet is lost in a round — then all packets that follow in the same round are dropped). If at least three packets from a RTT round make it to the destination *before* the concerned packet is lost, they may still trigger the triple-duplicate ACK mechanism in the following RTT round (see reference [14] for details).

Figure 4 depicts the response-time gains achievable by the padding misbehavior as a function of the packet loss ratio. Such a measure is of particular importance for misbehaving clients trying to maximize their performance gains, an issue we discuss in more depth below. All curves in Figure 4 show a similar shape. Initially, the gain is relatively small for very small packet loss ratios. Indeed, even if packet losses are detected via the RTO, such events are rare, and thus the impact on the *expected latency* is negligible. However, as the packet loss ratio increases, so does the gain. Interactive flows suffer more and more, while fully backlogged flows manage to improve their performance by relying on the triple-duplicate ACK mechanism. Finally, the gain starts to decrease as the packet loss ratio keeps increasing. In such environments, the TCP congestion window starts reducing, Q starts converging to 1, and padding is not as beneficial any more.

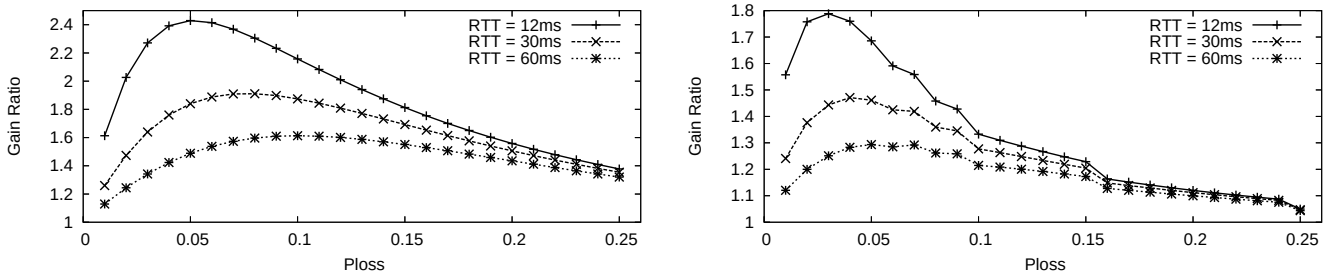
Figure 4 further shows that the gain is a function of RTT; the higher the RTT value, the smaller the RTO/RTT ratio, and the smaller the gain. Also, as RTT increases, the maximum gains are achieved for larger packet loss ratios. Indeed, as the RTO/RTT ratio decreases, it must be compensated by its factor $((1-p)/(1-2p) - 1)$ (Equations (3), (10)) to keep a balance, meaning that p increases. Finally, for the reasons explained above, RED's gain is larger than FIFO's.

D. Simulation

To verify our modeling results, we conduct extensive simulation experiments. The topology consists of a client and a server pool that are interconnected by a pair of routers and a bottleneck link. The effective round trip time fluctuates in the range from 10 to 100 ms; likewise, we vary the bottleneck link capacity from 1.5 to 10 Mbps. By generating the background cross traffic of appropriate intensity, we control the packet loss ratio at the bottleneck. We use *ns-2*'s TCP/FullTcpAgent, which is an implementation of a TCP Reno version. For each data sample, we run the simulation for a thousand seconds repeatedly and report averages.

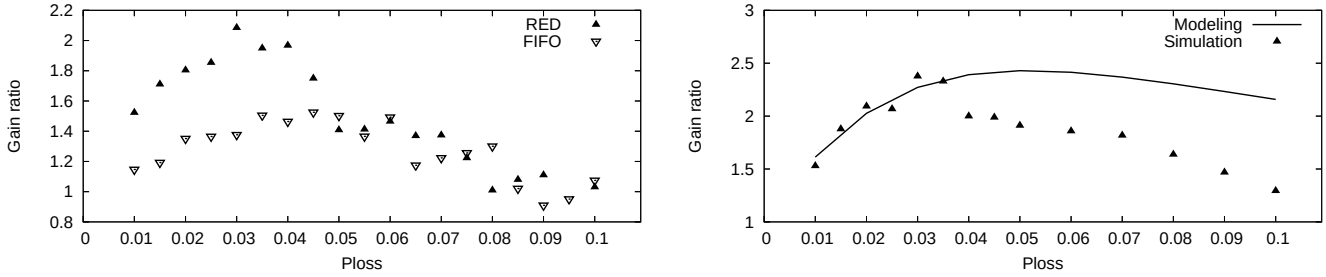
For interactive traffic, we open a telnet connection. The telnet client generates packets using an exponential distribution with average inter-arrival time of 1 second. For fully-backlogged TCP connections, we open FTP connections between a pair nodes, one each from the server and the client pool. To accurately emulate an interactive connection converted to a fully-backlogged connection, we mark packets randomly using the same exponential distribution as in the telnet scenario. For the analysis of the simulation results, we consider the statistics for those marked packets only.

Figure 5(a) plots the simulation results for the gain ratio (the y-axis in the figure) as a function of the packet loss rate (the x-axis in the figure) for RED and FIFO queues. In this particular scenario, we set the bottleneck bandwidth to 5 Mbps, and the round-trip propagation delay is 12 ms. The bottleneck router buffer is 40 kB; in the RED case, we use the default *ns-2* RED parameters. The shape of both curves in the figure is as predicted by modeling. Likewise, simulations confirm that gains are larger in the case of RED than with FIFO. However, due to varying queuing delay in simulations, and because the effective RTT increases with the packet loss ratio, we are unable to directly compare the modeling and simulation results in this scenario.



(a) RED (b) FIFO

Fig. 4. Modeling: Gain ratio as a function of packet loss prob.



(a) (b)

Fig. 5. Simulation: Gain ratio as a function of packet loss prob.

Thus, in order to perform a comparison, we proceed as follows. By applying the *ns-2*'s artificial random packet loss module, we manage to effectively control the packet loss ratio while keeping the RTT value relatively constant. Figure 5(b) shows the results. For packet loss ratios of up to 3.5%, the model and simulations match well. For larger packet loss ratios, the modeling results over-estimate simulations. This is because we assumed that the initial RTO is set to minRTO of 1 second [2]. However, when the packet loss ratio is high, this is not necessarily the case. For example, due to multiple packet losses in a single RTT round, a future packet may "inherit" a longer initial RTO, an effect that is *not* captured in our modeling. Still, the gain in both scenarios remains in favor of fully-backlogged flows.

E. Optimizing a Misbehaving Client's Performance

The above experiments indicate that there exists a "sweet spot," *e.g.*, a packet loss rate for which the misbehaving clients can maximize their performance gain. While the optimal point is a function of RTT and the queuing discipline at the bottleneck, misbehaving clients may be tempted to "drive" the system into the desired state. Our research (not shown due to space constraints) indicates that for reasonable bandwidth-delay products, it is possible for a client to maximize its gain simply by launching a moderate number of additional TCP connections.

IV. Sustainable Countermeasures

In this section, we explore ways to enhance the performance of interactive applications *without* applying the fully-backlogged approach. In other words, the challenge is to make *sustainable* changes, which if applied globally, would (i) solve the problem, yet (ii) without compromising the idea of statistical multiplexing. Our primary goal is to increase the performance of legitimate users to a level which will demotivate misbehaving clients from converting their interactive flows into fully-backlogged TCP connections.

A. Approach-I: Differentiated minRTO

We initially focus on the RTO parameter. Selection of the timeout value requires a balance among two extremes: if set too low, spurious retransmissions will occur when packets are incorrectly assumed loss when in fact the data or ACKs are merely delayed. Similarly, if set too high, flows will wait unnecessarily long to infer and recover from congestion. Allman and Paxson [15] experimentally showed that TCP achieves near-maximal throughput in the Internet if there exists a lower bound for RTO of one second. The study found that *all flows* should have a time-out value of at least 1 second in order to ensure that congestion is cleared, thereby achieving the best performance.

One approach to reducing the performance degradations experienced by application-limited flows is reducing the min-RTO parameter *exclusively* for such flows. In particular, we explore an approach in which a TCP sender is allowed to use a lower value for minRTO, (*e.g., minRTO'*), when its *used* congestion window size is less than a *fraction* of the current congestion window size (we quantify the *minRTO'* and *fraction* parameters below). While it is arguable whether such an approach can cause a congestion collapse, one argument on its behalf is that interactive applications represent only a small fraction of the Byte-level Internet traffic. Thus, having a few spurious retransmissions will not degrade the network performance to any perceptible amount. Moreover, in the context of the congestion collapse problem, this approach can only be better than the fully-backlogged one. Nevertheless, the

results we present below make such a discussion obsolete.

B. Approach-II: Short-term padding with dummy packets

In this approach, the goal is to improve the performance of interactive applications by increasing the probability that a packet loss is detected via the fast retransmit mechanism; yet, *without* applying the "brute-force" fully-backlogged approach. In particular, this could be achieved by appending the application data packet with three "tiny" (*e.g.*, 20 Bytes each) "dummy" packets. Indeed, RFC 3390 [19] enables setting TCP's initial congestion window size to 4 packets when TCP starts a new connection or restarts a connection after a long idle period. Thus, the three additional tiny dummy packets should help the endpoints detect data packet losses via triple dummy-packet-initiated duplicate ACKs.

The unique characteristic of this approach is that in addition to being implementable at the TCP layer, it could be implemented at the *application level* as well. An application should make sure that it does not send packets back to back; otherwise, TCP will make a single 60-Byte packet and send it to the network. Anyhow, contrary to the approaches above and below, interactive applications could immediately deploy this approach without requiring any kernel-level TCP changes.

1) Modeling: Here, we derive the response-time formula for the short-term padding approach. Assume a general scenario in which the minimum congestion window size parameter is m, such that $m-1$ packets are appended to a data packet. A timeout is invoked if two or less dummy packets reach the receiver; more precisely, if the TCP sender gets back two or less duplicate ACKs. Thus, the probability that the loss indication for a *data packet* is a timeout is given by

$$Q(m) = \sum_{i=0}^{2} \binom{m-1}{i} p^{(m-1-i)}(1-p)^i. \qquad (11)$$

Again, the probability that a data packet loss is detected by the triple duplicate ACK mechanism is 1-Q. Also, as discussed above, in case a retransmitted packet is lost again, it evokes an RTO. Thus, by applying the same approach as in Section III-B, the expected latency becomes

$$E[L'] = Q\left(\frac{1-p}{1-2p} - 1\right) RTO + RTT + p(1-Q)\left(\left(\frac{1-p}{1-2p} - 1\right) RTO + RTT\right). \qquad (12)$$

Strictly speaking, Equation (12) applies *only* to the random loss scenario. Indeed, under the assumption that if a packet is lost, so are all the packets that follow in the same round, the proposed approach is ineffective. However, our simulations indicate that the above correlated packet loss assumption is "too strong," and that when a data packet is lost, the corresponding follow-up packets are not always dropped in FIFO routers. Hence, the proposed approach improves the performance even in such scenarios, as we show below.

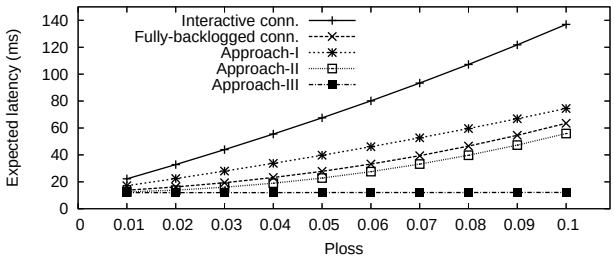


Fig. 6. Modeling: Expected latency as a function of packet loss prob.

C. Approach-III: A Diversity Approach

In this approach, we modify a TCP sender to send a packet k times; k is a small integer, $k > 1$. TCP sends k copies of a packet *without* violating TCP's congestion control mechanism. The key idea behind this approach is that the probability that at least one of the k copies of a packet will make it to the receiver is high. However, if all k packets are lost, TCP undergoes retransmission timeout and cuts down the congestion window to one. Hence, in the following retransmission rounds, it retransmits the packet only once.

This is a *TCP-only* approach; it cannot be deployed at the application layer. For example, if two copies of a packet are sent from the application to the TCP layer, TCP will treat them as two *different* packets. Thus, if the first packet is lost and the second one make it to the receiver, the second packet will only be *buffered* at the TCP layer; it will be "pushed" to the application layer only after the first packet is successfully retransmitted — which in this scenario happens after one RTO in the best case.

1) Modeling: Here, we derive the response-time formula for the diversity approach. Denote by k the number of copies of a packet a TCP sender generates. Then, the probability that *at least one copy of a packet* is transmitted successfully exactly after i failure rounds becomes

$$P_h(i) = \begin{cases} 1-p^k & \text{for } i = 0, \\ p^{k+i-1}(1-p) & \text{for } i \geq 1. \end{cases} \qquad (13)$$

In particular, the probability that at least one of the k packets in the first round successfully reaches the destination is given by $1-p^k$. For $i \geq 1$, $P_h(i)$ is given by the product of two probabilities: (i) the probability that all k copies are lost in the first round, and (ii) the probability that the single packet copy is lost in all subsequent $i-1$ rounds, followed by a successful transmission. Then, following the approach of Equation (3), it could be shown that for $p < 0.5$, the expected response-time latency becomes

$$E[L'] = p^{k-1} RTO\left(\frac{(1-p)}{(1-2p)} - 1\right) + RTT. \qquad (14)$$

Similarly to the above scenario, our modeling approach here strictly applies only to random packet losses. Still, simulations below indicate that the approach is viable in FIFO scenarios as well.

D. Evaluation

Here, we evaluate the effectiveness of the three approaches. For approach-I, we set the minRTO to 500 ms, and the *fraction* parameter to 1/4. Given that the minimum congestion window is four packets [19], this enables a client to always re-send a packet after an RTO of 500 ms. In approach-II, an application data packet is appended with three application-level dummy packets, each of the size of 20 Bytes. After TCP adds a 40-Byte-long header, the dummy packet size in the network becomes 60 Bytes. Finally, for approach-III, we set $k = 2$; thus, each packet is repeated twice.

Figure 6 plots the expected latency as a function of the packet loss ratio for the three approaches. The key observation is that the short-term padding and diversity approaches *outperform* the fully-backlogged approach. In this way, two goals are achieved: (i) The interactive-application clients no longer have incentives to generate fully-backlogged flows. Indeed, why converting to fully-backlogged when approaches-II and -III are better? (ii) Approaches-II and -III still preserve the idea of statistical multiplexing, as we demonstrate below.

Figure 6 also shows that despite the fact that we reduced the minRTO parameter by *a half*, the response time of approach-I is still higher than that of the fully-backlogged approach. Our evaluations (using both Equation (3) and simulations) indicate that reducing the minRTO parameter much more would help outperform the fully-backlogged approach. However, such an approach in essence converges to the approach-III (for $k = 2$), and hence we refrain from showing it further.

1) Simulations: Figure 7 plots the simulation results for the above scenarios, both for RED and FIFO routers. In simulations, we control the packet loss by varying the intensity of the cross traffic. In addition, we generate one of the flows indicated in the figure: an interactive, a fully-backlogged, an approach-II, and an approach-III flow. Figure 7(a) (the RED case) confirms general trends shown previously in Figure 6: approaches-II and -III outperform the fully-backlogged scenario. Moreover, as explained above (in Section III-D), due to inheriting longer than minRTO initial timeouts, the fully-backlogged flow experiences additional response-time degradations for larger packet-loss ratios.

Figure 7(b) plots simulation results for the FIFO case. While correlated packet losses, characteristic for drop-tail queues, do affect the overall performance, the key finding remains unchanged: both approaches-II and -III have lower response times than the fully-backlogged approach. For example, due to larger probability that both copies of a packet in approach-III will get dropped at the router, its performance is not as good as in the RED case. However, because the probability that both packets are lost concurrently does *not* equal one in reality, there still exists gain over the fully-backlogged approach. Also, contrary to the RED scenario, it is interesting that approach-II (padding) outperforms approach-III (diversity). Since padded dummy packets are smaller than data packets, the likelihood that they will get dropped at the Byte-based drop-tail queue is smaller.

2) Overhead and Sustainability: One final issue that we explore is overhead and sustainability. In essence, we explore scenarios in which a given approach is widely deployed, and evaluate (i) the performance gains over the greedy fully-backlogged approach, and (ii) performance reductions relative to the purely interactive, yet *unsustainable*, approach.

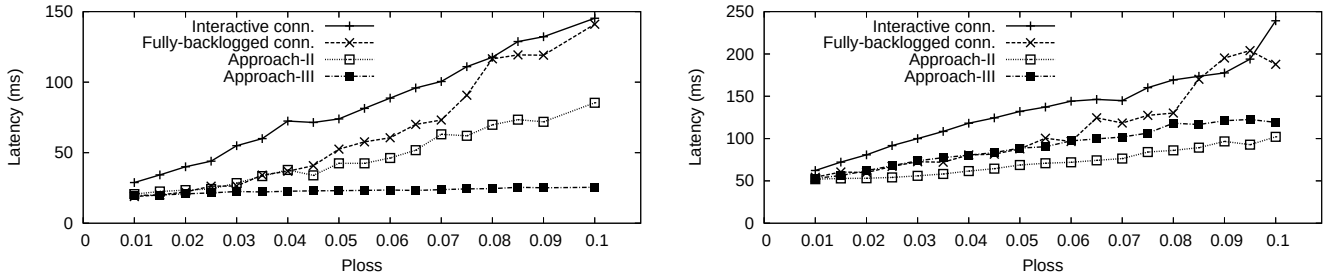
Figure 8 plots response times as a function of the number of flows in the network, when *all* clients apply a given approach indicated in the figure. Even for a moderate number of connections, the response times increase dramatically for the fully-backlogged approach. On the other extreme, the purely interactive approach can support more than 350 connections before the latency starts increasing. Unfortunately, as discussed above, this state is unstable in the sense that clients have incentives to improve their performance while still remaining TCP friendly. Finally, the figure shows that approaches-II and -III support a necessarily smaller number of connections relative to the interactive scenario. However, the key point is that both approaches provide (i) a sustainable solution that demotivates clients from moving the system into the fully-backlogged state; and (ii) a significantly "friendlier" environment relative to the fully-backlogged approach.

Figure 8 shows that in the case of approach-III, the latency starts increasing when the number of flows exceeds 175. Indeed, because clients send two copies of a packet by default, the "departure" point is approximately at one half of the number achievable by the purely interactive approach. Next, because the overhead for the approach-II is smaller (3*60 Bytes relative to 540 Bytes-long data packets), it can support a larger number of flows without increasing response times (the "departure" point is around 250 flows).

Also, while the performance for approach-II (padding) is approximately identical in the RED and FIFO scenarios, this is not the case with the diversity approach. Indeed, Figure 8(a) shows that RED's random packet dropping has a brilliant effect on approach-III, given that latency increases moderately with the number of flows. Not only that the approach dramatically outperform the fully-backlogged approach, but it even outperforms the pure interactive approach when there are many flows in the system. On the contrary, due to correlated packet losses, the latency slope is much steeper in the FIFO case.

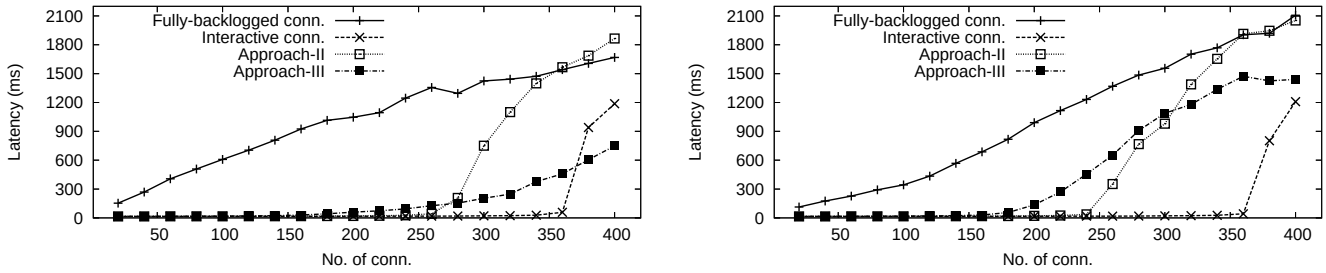
V. Conclusions

This paper revisited the well-known problem of unfairness between short- and long-lived TCP flows. Our first contribution lies in pointing out at an imminent and a serious implication of this problem: nothing stops clients of *interactive applications* to improve their response-time performance by generating traffic at a TCP-fair rate. The problem is imminent because the misbehavior is hard to detect, given that flows are TCP friendly. The problem is serious because it has the potential to jeopardize one of the core principles that today's Internet is built upon — statistical multiplexing. Second, we showed that interactive clients *always* have an incentive to send at a TCP-fair rate, because the corresponding response-time performance *always* outperforms the pure interactive



(a) RED (b) FIFO

Fig. 7. Simulation: Latency as a function of loss rate



(a) RED (b) FIFO

Fig. 8. Simulation: Number of flows vs latency: C = 1.5Mbps

approach. Moreover, we revealed that due to random packet losses, the gain is much larger for RED routers. Finally, we demonstrated that there exist simple, easy-to-deploy, and *sustainable* solutions that are capable of effectively demotivating clients from applying the greedy TCP-fair approach. In particular, we showed that a diversity method, accompanied with RED routers in the network, performs remarkably well. Still, the short-term padding approach appears even more attractive; it could be implemented at the *application layer*, without requiring *any* TCP-level modifications.

REFERENCES

[1] L. Guo and I. Matta, "The war between mice and elephants," in *Proceedings of IEEE ICNP '01*, Riverside, CA, Nov. 2001.

[2] V. Paxson and M. Allman, "Computing TCP's retransmission timer," Nov. 2000, Internet RFC 2988.

[3] L. Le, J. Aikat, K. Jeffay, and F. Smith, "Differential congestion notification: Taming the elephants," in *Proceedings of IEEE ICNP '04*, Berlin, Germany, Oct. 2004.

[4] W. Noureddine and F. Tobagi, "Improving the performance of interactive TCP applications using service differentiation," in *Proceedings of IEEE INFOCOM '02*, New York, NY, June 2002.

[5] "Which ports are used by computer games?" http://www.u.arizona.edu/~trw/games/ports.htm.

[6] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee, "Hypertext transfer protocol - HTTP/1.1," June 1999, Internet RFC 2616.

[7] M. Handley, J. Padhye, and S. Floyd, "TCP congestion window validation," June 2000, Internet RFC 2861.

[8] P. Patel, A. Whitaker, D. Wetherall, J. Lepreau, and T. Stack, "Upgrading transport protocols with untrusted mobile code," in *Proceedings of ACM SOSP '03*, Bolton Landing, NY, Oct. 2003.

[9] R. Mahajan, S. Floyd, and D. Wetherall, "Controlling high-bandwidth flows at the congested router," in *Proceedings of IEEE ICNP '01*, Riverside, CA, Nov. 2001.

[10] A. Kuzmanovic and E. Knightly, "A performance vs. trust perspective in the design of end-point congestion control protocols," in *Proceedings of IEEE ICNP '04*, Berlin, Germany, Oct. 2004.

[11] E. Brosh, G. Lubetzky-Sharon, and Y. Shavitt, "Spatial-temporal analysis of passive TCP measurements," in *Proceedings of IEEE INFOCOM '05*, Miami, FL, Mar. 2005.

[12] N. Cardwell, S. Savage, and T. Anderson, "Modeling TCP latency," in *Proceedings of IEEE INFOCOM '00*, Tel Aviv, Israel, Mar. 2000.

[13] B. Kim and J. Lee, "Retransmission loss recovery by duplicate acknowledgement counting," *IEEE Communications Letters*, vol. 8, no. 1, Jan. 2004.

[14] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose, "Modeling TCP Reno performance: A simple model and its empirical validation," *IEEE/ACM Transactions on Networking*, vol. 8, no. 2, pp. 133–145, Apr. 2000.

[15] M. Allman and V. Paxson, "On estimating end-to-end network path properties," in *Proceedings of ACM SIGCOMM '99*, Vancouver, British Columbia, Sept. 1999.

[16] L. Le, J. Aikat, K. Jeffay, and F. Smith, "The effects of active queue management on Web performance," in *Proceedings of ACM SIGCOMM '03*, Karlsruhe, Germany, Aug. 2003.

[17] A. Kuzmanovic, "The power of explicit congestion notification," in *Proceedings of ACM SIGCOMM '05*, Philadelphia, PA, Aug. 2005.

[18] M. Allman, S. Floyd, and C. Partridge, "Increasing TCP's initial window," 1998, Internet RFC 2414.

[19] ——, "Increasing TCP's initial window," Oct. 2002, Internet RFC 3390.

[20] S. Yang and G. de Veciana, "Size-based adaptive bandwidth allocation: Optimizing the average QoS for elastic flows," in *Proceedings of IEEE INFOCOM '02*, New York, NY, June 2002.

[21] S. Savage, N. Cardwell, and T. Anderson, "The case for informed transport protocols," in *Proceedings of HotOS '99*, Rio Rico, Arizona, Mar. 1999.

[22] T. Anderson, A. Collins, A. Krishnamurthy, and J. Zahorjan, "PCP: Efficient endpoint congestion control," in *Proceedings of NSDI '06*, San Jose, CA, May 2006.

[23] J. Farber, "Network game traffic modeling," in *Proceedings of NetGames '02*, Braunschweig, Germany, Apr. 2002.

[24] P. Danzig and S. Jamin, "tcplib: A library of internetwork traffic characteristics," *USC Technical Report, Computer Science Department*, 1991, Report CS-SYS-91-01.

[25] F. Smith, F. Campos, K. Jeffay, and D. Ott, "What TCP/IP protocol headers can tell us about the Web," in *Proceedings of ACM SIGMETRICS '01*, Cambridge, MA, June 2001.

[26] R. Morris, "TCP behavior with many flows," in *Proceedings of IEEE ICNP '97*, Atlanta, GA, Oct. 1997.