

# Removing Exponential Backoff from TCP

Amit Mondal  
a-mondal@cs.northwestern.edu

Aleksandar Kuzmanovic  
akuzma@cs.northwestern.edu

Department of Electrical Engineering and Computer Science  
Northwestern University  
Evanston, IL 60208, USA

## ABSTRACT

The well-accepted wisdom is that TCP’s exponential backoff mechanism, introduced by Jacobson 20 years ago, is essential for preserving the stability of the Internet. In this paper, we show that removing exponential backoff from TCP altogether can be done without inducing any stability side-effects. We introduce the *implicit packet conservation principle* and show that as long as the endpoints uphold this principle, they can only improve their end-to-end performance relative to the exponential backoff case.

By conducting large-scale simulations, modeling, and network experiments in Emulab and the Internet using a kernel-level FreeBSD TCP implementation, realistic traffic distributions, and complex network topologies, we demonstrate that TCP’s binary exponential backoff mechanism can be safely removed. Moreover, we show that insuitability of TCP’s exponential backoff is fundamental, *i.e.*, independent from the currently-dominant Internet traffic properties or bottleneck capacities. Surprisingly, our results indicate that a path to incrementally deploying the change does exist.

## Categories and Subject Descriptors

C.2.2 [Computer Communication Networks]: Network Protocols

## General Terms

Algorithms, Experimentation, Measurement, Performance

## Keywords

TCP, exponential backoff algorithm, congestion collapse, implicit packet conservation principle

## 1. INTRODUCTION

One of the main goals of a congestion control algorithm is to protect the network against *congestion collapse* (*e.g.*, [16, 26, 36]) — an armageddon scenario in which endpoints send packets at a high rate into the network, majority of which gets dropped never reaching the destination. As a result, the network stays highly congested and useless, since the effective network throughput converges to zero.

The congestion collapse phenomenon was first introduced by Nagle in 1984 [36], and one of the first incarnations of such a scenario was reported in 1986, when the data throughput from Lawrence Berkeley Laboratory to UC Berkeley dropped from 32 kbps to 40 bps [20]. Reacting to the crisis, Jacobson proposed and implemented a set of algorithms that fundamentally helped resolve the problem. The basic idea was to make the endpoints more conservative. The key

innovations were slow-start, dynamic window sizing, round-trip-time variance estimation, and exponential retransmit timer backoff. These approaches are considered the main barrier that saved the Internet from the congestion collapse to date. Consequently, these algorithms are ubiquitously deployed literally in all TCP variants.

Twenty years have passed since the above events and the Internet has become a much different place. The penetration of high-speed broadband access technologies and highly-skewed traffic distributions are example factors that already significantly impacted a number of end-point and network protocol designs. Large bottleneck capacities enabled more aggressive end-point start-up behavior [5, 15] and control [14, 21, 25, 31]; moderate flow sizes inspired novel probe-based congestion control protocols [7]; and skewed flow-size distributions significantly impacted router buffer sizing [8] and enabled scalable fair queuing algorithms [27].

In this paper, we focus on and challenge the need for *TCP’s binary exponential backoff mechanism* that has never been modified since originally introduced 20 years ago [20]. This is despite the well-known side-effects that this algorithm can cause. For example, web clients can frequently trigger the algorithm and experience orders of magnitude response time degradations (*e.g.*, [18, 28, 29]), even when no collapse is on its way. Nevertheless, the common wisdom is that the exponential backoff algorithm is essential for preserving the stability of the Internet [20].<sup>1</sup>

Protocol implementers and people more closely dealing with the above problems are well aware of the above trade-offs. On the one hand, they are tempted to overcome the above performance degradations; at the same time, they do not want to be blamed for causing the collapse of the Internet. The result is quite intriguing: the exponential backoff algorithm is preserved, yet certain TCP implementations essentially defer entering the algorithm. As an example, we found that FreeBSD versions 4.0 and above enter the binary backoff algorithm only after *five* consecutive TCP SYN packets get dropped, thus directly violating the behavior standardized by RFC 2988 [37].

The key contributions of this paper are as follows. (*i*) We effectively ‘untangle’ the retransmit timer backoff mechanism from other TCP mechanisms, and systematically evaluate this fundamental piece of congestion control in diversified network scenarios. (*ii*) We show that *removing the exponential backoff mechanism altogether from TCP* could be done without causing any collapse-like scenarios. We prove this result for single-bottleneck scenarios, and perform ex-

<sup>1</sup>“an unstable system (a network subject to random load shocks and prone to congestive collapse) can be stabilized by adding some exponential damping (exponential timer backoff) to its primary excitation (senders, traffic sources)” [20].

tensive large-scale experiments to show that this holds for multiple bottlenecks as well. (iii) We demonstrate that removing the exponential backoff algorithm from TCP is an incrementally deployable two-step task. Most importantly, (iv) we show that the inappropriateness of TCP’s exponential backoff is fundamental, *i.e.*, independent from the currently-dominant Internet traffic properties or bottleneck capacities.

In the broader context, the key implication of our work is that it opens the doors for evaluating other well-accepted pieces of TCP congestion control. The growing sense is that a number of these pieces were introduced in 1986 in a ‘rush’ to solve a very pressing problem [31]. Hence, it is viable to revisit each of these algorithms independently and understand what is really needed and what is not.

## 2. BACKGROUND AND MOTIVATION

When a packet is sent into the network, there are no guarantees it will make it to the destination. An important question thus becomes: how long should a TCP endpoint wait before retransmitting the packet, *i.e.*, how to set retransmission timeouts? Another question is how to behave in scenarios when a packet must be retransmitted more than once, *i.e.*, how to set the value of the new retransmission timeout once the previous one has expired? Below, we provide a brief background standing behind the current practice in the Internet, and then motivate our research.

### 2.1 Retransmission Timeouts and Exponential Backoff: Origins

#### 2.1.1 Retransmission Timeouts

TCP uses a retransmission timer to ensure data delivery in the absence of any feedback from the remote receiver. The duration of this timer is referred to as the retransmission timer (RTO). RFC 2988 [37] specifies that a host TCP must implement Jacobson’s algorithm [20] and Karn-Partridge’s algorithm [23] for computing the RTO. Jacobson’s algorithm for computing the RTO incorporates a simple measure of smoothed RTT,  $SRTT$ , and RTT variance,  $var(RTT)$ , *i.e.*,  $RTO = SRTT + 4 * var(RTT)$ . Karn-Partridge’s algorithm for selecting RTT measurements ensures that ambiguous round-trip times will not corrupt the calculation of the smoothed round-trip time.

Allman and Paxson studied the fundamental tradeoff in setting the RTO value [6]. The more aggressive the value is the less time a connection spends waiting for needed RTOs. But, at the same time, it experiences more spurious RTOs. RFC 2988 [37] specifies that whenever RTO is computed, if it is less than one second then the RTO should be rounded up to one second. The authors acknowledge that at some future point, research may show that a smaller RTO is acceptable or superior. To the best of our knowledge, the current practice in most operating systems is setting the minimum RTO parameter to 200 msec [40].

#### 2.1.2 Backoff Mechanisms

Another question is how to help the network during the period of severe congestion. In particular, how to set the value of the new retransmission timeout once the previous one has expired? The original TCP recommendation [2] simply reuses the existing RTO value without any changes. Jacobson first introduced a retransmit timer *backoff* mechanism in TCP [20]. The backoff mechanism controls how

the retransmits should be spaced, if a packet has to be retransmitted more than once. Jacobson adopted the backoff mechanism from the classical (shared-medium) Ethernet protocol of Metcalfe and Boggs [34], justifying the choice by the observation that an IP gateway has essentially the same behavior as the ‘ether’ in a shared-medium Ethernet network.

The Ethernet backoff protocol is called *binary exponential backoff* and it is used for the control of a multiple-access broadcast channel. A single channel allows several geographically dispersed stations to communicate with each other by broadcasting packets onto the channel. If two or more stations simultaneously attempt to broadcast a packet, then the transmission interferes and each fails. There is no central control, so the protocol resolves collisions using the backoff algorithm. A station waits a random time  $T$  after its  $k^{th}$  unsuccessful attempt at transmitting a given packet and then reattempts, where  $E(T) = 2^k$  time slots.

An extensive theoretical effort stands behind the choice of using exponential backoff in the Ethernet protocol. In the *queuing* model of the backoff algorithm, it is assumed that there are  $N$  users, and user  $i$  ( $1 \leq i \leq N$ ) generates a message independently in each step with probability  $\lambda_i$ . The total mean arrival rate  $\lambda = \sum \lambda_i$ .

The quality of a backoff protocol is measured by its *stability* and *capacity*. A backoff protocol is defined to be stable, *for a given arrival rate*  $\lambda$ , if the expected number of waiting messages over time is finite. A protocol is said to achieve throughput  $\lambda$  if, when it runs with an input rate  $\lambda$ , the average success rate is  $\lambda$ . The capacity of a protocol is the maximum throughput it can achieve. Goldberg *et al.* [17] prove the upper capacity bound for a shared-medium Ethernet backoff protocol. Kelly [24] shows no collision backoff slower than exponential guarantees stability on an Ethernet. Moreover, Aldous [4] shows that with infinite user population even exponential backoff will not guarantee stability for any positive arrival rate.

### 2.2 Rationale for Revisions

Here, we outline the key reasons motivating us to revisit the design, and more fundamentally, suitability of the binary exponential backoff algorithm for TCP.

First, despite a longer-than-a-decade-long research effort in the area of admission control (*e.g.*, [9]), no such algorithm is implemented in today’s Internet. Indeed, packets are forwarded on the best-effort basis, and new flows can enter the system despite potentially heavy congestion in the network. Moreover, there is no bound on the number of active flows sharing a single bottleneck link in the Internet. Thus, for a transport endpoint embedded in a network of unknown topology and with an unknown, unknowable and constantly changing population of competing flows, whatever stability results exist for the backoff algorithm with bounded arrivals simply cannot be applied in the Internet.

Second, Jacobson’s observation that an IP gateway has essentially the same behavior as the ‘ether’ in a classical Ethernet network [20] is discussable. A classical Ethernet is a shared medium, where only one node can transmit at a time. Any simultaneous transmissions result in a collision and the time slot is wasted, making it possible for the throughput to drop to *zero* in overloaded scenarios. This is *not* the case with routers in the Internet. Indeed, assume a router with an outgoing link capacity  $C$ ;<sup>2</sup> even in the most congested scenarios in which new incoming packets are get-

<sup>2</sup>We assume that the traffic flow is bottlenecked at an out-

ting dropped, the throughput does not drop to zero.

Moreover, finite flow sizes, dynamic flow arrivals and departures, skewed traffic distributions, a large number of short flows and increased bottleneck capacities in today’s Internet arising due to penetration of high-speed broadband technologies are all factors that even further challenge the need for TCP’s exponential backoff algorithm. While there are still areas in the world where the penetration of high-speed technologies is not high, we show that insuitability of TCP’s exponential backoff is fundamental, *i.e.*, independent from the currently dominant Internet bottleneck capacities.

### 2.3 Implicit Packet Conservation Principle

Despite the lack of admission control, the mismatch between shared and non-shared mediums, and potential benefits one would be able to achieve by removing TCP’s exponential backoff, it appears that this mechanism would still be highly beneficial in highly congested scenarios when traffic arrivals are systematically above an IP gateway’s service rate. Our key result is that this is *not* the case.

Consider a single congested gateway. Given that the service rate of an outgoing gateway’s link is independent of the packet loss ratio in such scenarios, the implication is that as long as  $RTO > RTT$  for each of the endpoints, the *end-to-end* performance experienced by clients will *not* degrade even if we remove the exponential backoff algorithm altogether. If a copy of a packet is sent into the network only after the previous copy has been dropped (*i.e.*,  $RTO > RTT$ ), so that no unnecessary congestion is induced by the endpoints, *the congested gateway will serve each packet only once*, and thus the end-to-end performance will not suffer.

We call the  $RTO > RTT$  condition the *implicit packet conservation principle*. It is implicit because an endpoint has no feedback from the network that the packet has been lost, and thus it has to rely upon the RTO value to *implicitly* determine that the packet has been lost. Indeed, this question has been explored in depth in [20, 23].

**Theorem 2.1** *In a single-bottleneck scenario where end-hosts satisfy the implicit packet conservation principle, the average file response time will not degrade after the removal of TCP’s exponential backoff mechanism.*

**Proof:** Consider  $n$  TCP flows with heterogeneous RTTs are sharing a single bottleneck link. Denote by  $RTT_i(t)$  the round-trip time of the  $i$ -th TCP flow at time  $t$ ,  $i = 1, \dots, n$ . Also, denote by  $RTO_i(t)$ , for  $i = 1, \dots, n$ , their corresponding retransmission timeouts at time  $t$ . Assume that for all  $t$  the implicit packet conservation principle holds for each connection. Thus, by removing the parameter  $t$ , it holds that  $RTO_i = SRTT_i + 4 * var(RTT_i) > RTT_i$ . Denote by  $C$  the capacity of the bottleneck link (in packets per unit time). Assume a congested scenario in which the packet arrival rate exceeds service rate  $C$ , *i.e.*,  $\sum_{i=1}^n 1/RTO_i > C$ , and endpoints operate in the exponential backoff mode.

If flows do not apply the exponential backoff algorithm, then the packet loss ratio  $p$  at the bottleneck becomes

$$p = \frac{\sum_{i=1}^n 1/RTO_i - C}{\sum_{i=1}^n 1/RTO_i}. \quad (1)$$

Thus, a flow of size  $L_i$  has to send  $L_i/(1-p)$  packets on average to successfully transfer all its  $L_i$  packets. Hence, its

going link. Even if the bottleneck exists elsewhere in the router, our analysis below still holds. The key point is that the router bottleneck does not fully starve the traffic flow, which can happen in the classical Ethernet scenario.

average response time  $r_i$  becomes

$$r_i = \frac{L_i \cdot RTO_i}{1-p}. \quad (2)$$

Combining Eq. (1) and (2), it follows that

$$r_i = \frac{L_i \cdot RTO_i \cdot \sum_{i=1}^n 1/RTO_i}{C}. \quad (3)$$

Next, assume that flows apply the binary exponential backoff algorithm to avoid packet losses such that each flow sends a single packet per  $2^k \cdot RTO_i$  seconds. Then  $k$  should be such that

$$\sum_{i=1}^n \frac{1}{2^k \cdot RTO_i} \leq C. \quad (4)$$

Hence, it follows that

$$k \geq \log_2 \left( \frac{\sum_{i=1}^n 1/RTO_i}{C} \right). \quad (5)$$

Thus, a flow of size  $L_i$  has the average response time  $r'_i$  equal to  $L_i \cdot 2^k \cdot RTO_i$ . From Eq. (5) it follows that

$$r'_i \geq \frac{L_i \cdot RTO_i \cdot \sum_{i=1}^n 1/RTO_i}{C}. \quad (6)$$

Finally, from Eq. (3) and (6), it follows that  $r'_i \geq r_i$ .  $\square$

While certainly insightful and illustrative, the theorem above makes several assumptions which we outline below, and extensively evaluate further in the paper.

1. *All flows in the exponential-backoff state?* We assume a highly congested case in which *all* endpoints operate in the exponential backoff mode. The key point is that if the exponential backoff is unnecessary even in this extreme scenario (as the theorem implies), then it is unnecessary in other scenarios in which a subset of flows operate in the non-exponential-backoff mode. Nevertheless, the above proof does not formally apply to such cases. Hence, we extensively evaluate such scenarios in a network testbed with state-of-the-art TCP implementations in Sections 4.1 - 4.5 below.
2. *Implicit packet conservation principle always fulfilled?* The TCP’s RTO estimator  $RTO = SRTT + 4 * var(RTT)$  is sufficiently conservative to fulfill the implicit packet conservation principle in most scenarios, *i.e.*,  $RTO > RTT$ . However, it does not *guarantee* that a given RTO estimate is always greater than a given RTT sample. For example, due to flash crowds, it is possible that RTT dramatically increases in a short period of time such that RTO becomes less than RTT. We explore such scenarios in Section 4.6 below. We use the same experiment to explore the performance of the algorithm in extremely congested scenarios, *e.g.*, packet loss rate at the order of 45%.
3. *Multiple-bottlenecks?* The theorem above holds for a single bottleneck scenario. The question is what happens in multiple-bottlenecked scenarios? We address this question in depth in Section 5 below.

## 2.4 How Often Does Exponential Backoff Happen in the Internet?

While our main objective is to address a fundamental TCP design issue — should TCP apply the exponential backoff algorithm or not — the performance repercussions

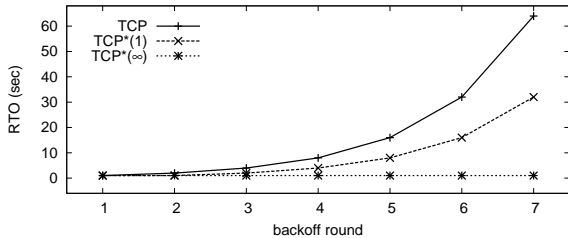


Figure 1: RTO as a function of the backoff round

of the proposed change are not negligible. Here, we explore how frequently does a TCP connection experience retransmission timeouts and enter the exponential backoff in today’s Internet.

**Tbit-like experiments.** Ideally, monitoring TCP’s behavior at a number of web servers would give a representative answer to this question. Since we do not have access to any of the popular servers, we develop a tool that infers timeout events at the server side from client-side-only measurements. In particular, our tool passively monitors packet arrivals and departures at the client side and estimates a connection’s roundtrip times. Then, it compares inter-packet arrival times to detect RTO and backoff events<sup>3</sup>. In our experiment, we select a well-connected machine at a university campus network as the client. The machine sends requests for index pages to 75,000 popular web servers around the world, from the Tbit web server list [33].<sup>4</sup>

Analyzing the measurement data from the above experiment, we find that approximately 8% of the connections experience timeout at least once during their entire transfer time. Furthermore, approximately 52% of the connections that experience timeout also experience a consecutive timeout, *i.e.*, TCP’s exponential backoff is invoked. It is important to understand that our experiments are conducted from a well-connected endpoint at a university campus. Also, to ensure that our experiment does not induce congestion at the receiver access network, the client connects only to a single server at a time. Moreover, index pages that we download are in general short in size. We believe that our analysis here provides only a *lower bound* on the percentage of connections that experience exponential backoff. In scenarios where congestion also happens at the receiver side, *e.g.*, when a client is behind a residential access network, or when a connection is longer-lived, the exponential backoff mechanism could be invoked even more frequently.

### 3. PRELIMINARIES

#### 3.1 “Design” Space

Our key goal to test the hypothesis that we can live without the exponential backoff algorithm. Still, one important issue, which we explore in more depth in Section 6, is *incremental deployability* of the proposed change in the current TCP stack and its coexistence with the state-of-the-art TCP variants. For that reason, we explore a class of *sub-exponential* backoff algorithms which should be easier to sub-exponentially deploy in today’s Internet, simply because their deviation from the pure exponential backoff algorithm is smaller than is the case with the backoff-less algorithm.

<sup>3</sup>We infer timeout if inter-packet arrival time is greater than 200 ms and 1.2 times the estimated RTT value.

<sup>4</sup><http://www.icir.org/tbit/URLListFeb2004.txt>

In particular, we define a class of TCP\*(n) sub exponential backoff algorithms, where the endpoints do *not* backoff the first *n* consecutive timeouts. For example, as shown in Figure 1, regular TCP’s backoff pattern is purely exponential,  $RTO^*(1,2,4,8,16,32,64)$ . On the other hand TCP\*(1) disregards the first consecutive timeout such that its backoff pattern becomes  $RTO^*(1,1,2,4,8,16,32)$ . Finally, the last algorithm in the chain is TCP\*(6), which we call TCP\*(∞), given that there is no backoff anymore. It simply applies the RTO parameter (computed using Jacobson’s and Karn-Partridge’s algorithms) consistently as shown in the figure.<sup>5</sup>

#### 3.2 Experimental Methodology

**Testbed.** The experimental testbed consists of a cluster of 64-bit Intel Xeon machines running FreeBSD 6.1 kernel. We modify the TCP implementation in the FreeBSD kernel to realize TCP\*(n) stack. The single-bottleneck scenario is a dumbbell-shaped topology with a pool of Apache servers at one side, and a pool of clients connected on the other side of the bottleneck link. We use dummynet to distribute the RTT between the clients and the servers in the range from 10ms to 200ms on a per-flow basis in order to emulate a wide-area network environment. We limit the bandwidth of the bottleneck link to 50 Mbps, which represents an uncongested scenario, and 10 Mbps, which represents a congested scenario. The bottleneck router is equipped with a RED queue by default; we also evaluate DropTail queues later in the paper. We use TCP Sack for all our experiments unless otherwise specified. We set the queue size to two times the bandwidth-delay product.

In the experiments, we first limit the network capacity between the client and server pools to 50 Mbps. Next, we choose the new connection arrival rate parameter such that the resulting average network throughput, in the direction from servers to clients, becomes between 10 Mbps and 15 Mbps. Finally, we limit the rate between the two pools to 10 Mbps, which enables us to explore congested network scenarios. The key performance measure of interest is flow completion time [13].

**Workloads.** For our experimental workload generation, we leverage the traffic model developed in [10]. It prototypes synthetic HTTP traffic based on recent Internet traffic measurements. While certainly representing traffic patterns characteristic for web-based activities in today’s Internet, HTTP has become widely used for other applications as well, including popular peer-to-peer file-sharing systems [22]. As a result, the traces we deal with are *not* web-limited. Moreover, we enrich the traffic characteristic even further by changing the given traffic distribution as we explain in detail below.

The model generates synthetic traffic based on the empirical heavy-tailed distribution reported in [29]. While the majority of the flows are very short (the mean file size is around 7 kBytes), GByte-long file sizes are also generated. Next, to generate a trace dominated by short flows, we truncate each file-size sample that is larger than 10 kB, to 10 kB, thus moving the average flow size towards smaller values. Likewise, to skew the distribution towards longer-flow sizes, we increase the percent of long-lived flows as follows. Starting from the baseline distribution [29], whenever a generated flow-size sample is less than 5 kB, we randomly (with probability 0.01) alternate this value to a longer flow size,

<sup>5</sup>The RTO parameter stops doubling when a connection experiences seven or more consecutive timeouts [37].

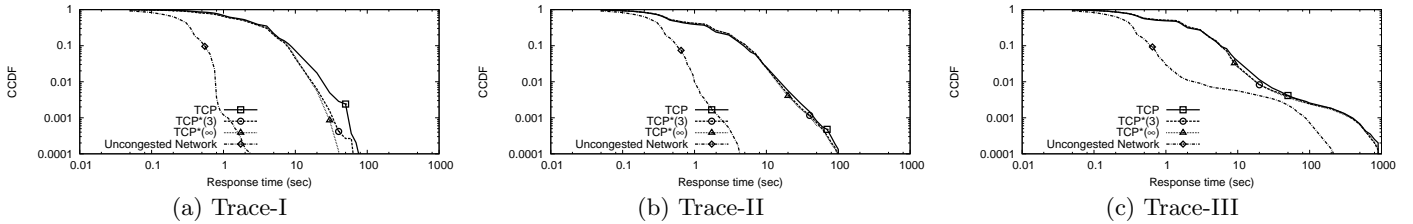


Figure 2: CCDF of response times with different backoff mechanisms (minRTO=1.0s, initRTO=3.0s)

according to the exponential distribution of the long-term average of 10 MB. Henceforth, we denote by Trace-I the flow size distribution skewed towards short file sizes, by Trace-II the baseline distribution of [29], and by Trace-III the one skewed towards longer file sizes. For the scenarios where generating Emulab-based experiments is either highly non-trivial or infeasible, we conduct ns-2 based simulations.

## 4. EVALUATION

### 4.1 No Admission Control

Here, we explore the problem of the lack of admission control in the Internet, which has important implications on system performance. As discussed above, stability in the theoretical sense can be guaranteed only in scenarios where the number of arriving flows can be controlled. While no such mechanism explicitly exists in today’s Internet, we explore whether the fact that most operating systems adopt a policy where a TCP connection is aborted if it experiences three (or five for Fedora/Linux) consecutive SYN packet drops may be considered as a way of *implicit* admission control. Below, we explore if, and under which circumstances, does this implicit admission control become functional.

Denote by  $p$  the packet loss probability at a congested router. Then, assuming connection abortions after three consecutive SYN drops, the probability that the connection will be implicitly rejected becomes  $p^3$ . For example, if the packet loss rate is as high as 10%, the probability to implicitly reject the connection is 0.001. Nevertheless, we run simulation experiments in a single bottleneck scenario to understand how the offered load on the congested link and a queuing algorithm at a router (*e.g.*, FIFO or RED) affects the probability a connection will be rejected from the system. Our findings are the following.

In both RED and FIFO routers operating in the Byte mode we were unable to log the case where a connection was rejected due to triple SYN drop, despite extremely heavy load placed on the congested routers. This is not a surprise. Because SYN packets are small in size (40 Bytes), they are almost certainly admitted and served by a Byte-based router. For the packet-based routers, our experiments confirm the above modeling results. Indeed, even under close-to catastrophic congestion levels at a router (*e.g.*,  $p=0.5$ ) where each second packet gets dropped, the probability to implicitly reject a connection is slightly above 10% for kernels that abort a connection after 3 SYN attempts, and about 3% for those that abort a connection after 5 consecutive tries.

### 4.2 Sub-Exponential Backoff

Here, we explore the performance of a class of sub-exponential backoff algorithms (including the backoff-less one, termed TCP\*( $\infty$ )) explained in Section 3.1 above. We set the retransmission timers following the recommendation from RFC 2988 [37], *i.e.*, minRTO is 1 sec, and initRTO is 3 sec. Later

in the paper, we explore other minRTO values (*i.e.*, 200 ms) as well. Figures 2(a), 2(b), and 2(c) depict the CCDF functions of response times for three traffic traces. Trace-I is dominated by short flows, Trace-II is the baseline distribution of [29], and Trace-III is skewed towards longer file sizes.

Figure 2(a) indicates that the tail of the backoff-less algorithm reduces relative to other backoff schemes. This is because the backoff-less algorithm manages to utilize all potential “gaps” in service that arise due to highly bursty arrivals, despite the fact the average arrival rate is above the service rate. By being more aggressive, the backoff free scheme manages to utilize the link more efficiently, and reduce the CCDF tail.

Figures 2(b) and 2(c) show that almost no service gaps exist when the bottleneck is exposed to Traces-II and -III at a high rate. As the flow-size distribution becomes more and more heavy tailed, the response times necessarily increase relative to the Trace-I scenario. Likewise, as the number of long-lived TCP connections increases, the number of competing active connections present in the network becomes large, which in turn reduces the gain of using sub-exponential backoff schemes. We will later show other less-congested scenarios in which the gain can be as high as an order of magnitude.

**Key observation.** However, no matter how monotonic Figures 2(a), 2(b), and 2(c) may appear, they convey the most important message of this paper: the end-to-end performance does *not* degrade when removing the exponential backoff algorithm from TCP. Indeed, as long as the endpoints uphold the implicit packet conservation principle, the end-to-end performance will not suffer. This is particularly the case in this scenario when relatively conservative minRTO (1 sec) and initRTO (3 sec) values of RFC 2988 are applied.

One other important implication of the implicit packet conservation principle is that the *aggregate* goodput (defined as the number of “unique” packets served at the congested router) does *not* degrade in this scenario, despite increased packet loss ratio.

While RFC 2988 [37] recommends setting the minRTO to 1 sec, the current practice is setting minRTO to 0.2 sec [40]. Hence, we repeat the experiment with minRTO 0.2 sec and initRTO 3.0 sec, and we obtain results consistent to those shown in Figure 2. The experiments show that our findings are independent from RTO parameter setups. In the following experiments, we stick with these currently dominant parameters, *i.e.*, minRTT 0.2 sec and initRTO 3.0 sec.

### 4.3 Bottleneck Capacity

In all experiments thus far, we experimented with a 10 Mbps bottleneck link. Here, we change the bottleneck capacity to 50 Mbps. Our intention is to understand how the response-time profiles change relative to the 10 Mbps case. We experiment with TCP and TCP\*( $\infty$ ) stacks, and the working

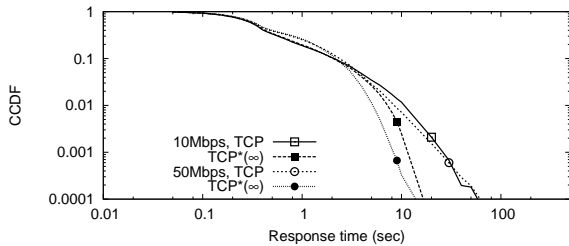


Figure 3: Impact of bottleneck capacity

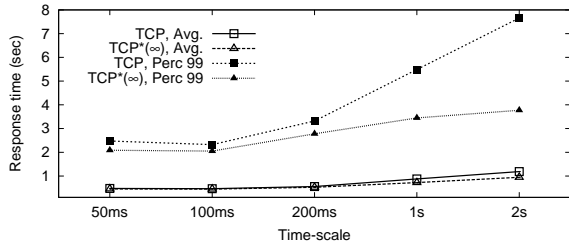


Figure 4: Average and 99<sup>th</sup> percentile of response times

trace is again Trace-I. While the file-size distribution is unchanged, we scale the arrival rate so that the effective load placed in both cases is equivalent.

Figure 3 shows the CCDF response-time profiles for the two bottleneck capacities. The key point is that the TCP\*( $\infty$ ) outperforms the classical TCP stack dramatically, independent of the capacity change. Likewise, for Traces-II and -III (not shown), the gain slightly changes for these overloaded scenarios, yet it is more moderate than in the Trace-I case.

#### 4.4 Dynamic Environments

Here, we explore how the backoff-less algorithm performs in much more dynamic environments, characterized by *bursty* traffic arrivals. In particular, we keep the overall long-term average load constant, but vary the duration of *on* and *off* arrival periods. Connections are arriving only during *on* periods. To keep the overall load high, we double the arrival rate during *on* periods. While not representative of an actual or realistic scenario, our goal here is to understand the impact that inter-burst arrival time scales, which we vary from 50 *ms* to 2 *s*, have on the response times.

Figure 4 depicts average as well as 99<sup>th</sup> percentile of response times for TCP and TCP\*( $\infty$ ) stacks as a function of *on* (burst) and *off* (no burst) time scales for Trace-I. Expectedly, the difference between 99<sup>th</sup> percentile of response times (tails of CCDF distributions) is much larger than it is the case with averages. This is because during periods of persistent congestion many TCP flows enter long backoff periods. As a result, new flows are being admitted into the system while already admitted ones are not served. On the other hand, the backoff-less TCP\*( $\infty$ ) stack effectively serves admitted flows without wasting time in unnecessary backoffs, thus reducing the number of active flows in the system. Exactly because of this, the tail of the CCDF curve for TCP is approximately twice as larger than it is for TCP\*( $\infty$ ).

Both average and 99<sup>th</sup> percentile of response times show an increasing trend as the burst time-scales increase. This is because the opportunities (and gains) of being more aggressive decrease with longer bursts than TCP.

#### 4.5 TCP Variants and Queuing Disciplines

In this section, we evaluate other TCP variants, *i.e.*, TCP

Reno and Tahoe, as well as their backoff-less variants, and compare them to TCP Sack which we used thus far. In addition, we explore the performance of these stacks on RED and DropTail routers. Figure 5(a) depicts the CCDF of response times for RED for Trace-I. While the figure again shows that the backoff-less variants improve the response-time profiles over their original stacks for all TCP versions, the most interesting detail in the figure is that there is no significant difference among various TCP stacks. This is despite the fact that all these TCP versions differ in their loss recovery phase: Reno improves Tahoe by less conservatively reacting to loss events; Sack further improves Reno in its fast recovery phase when multiple packets are dropped from a single window.

While TCP Sack slightly outperforms the other two versions, the difference is not so dramatic. This is not a surprise. Given that a congestion window remains small due to persistent congestion, selective acknowledgement has very little impact on individual response times and goodput of the bottleneck link. Because windows are small, all versions in a way operate in an almost single-packet “selective acknowledgement” mode, and hence there is no big difference. Figure 5(b) depicts similar results for the DropTail queue.

#### 4.6 Flash Crowds

The retransmission timer can expire spuriously and cause unnecessary retransmissions when no segments have been lost [41]. This can happen due to a temporary delay spike or a more permanent but sudden delay increase due to a flash crowd in the TCP data or ACK path. Below we explore the impact of removing exponential backoff from TCP on end-to-end performance in such scenarios.

In our experiments, the bottleneck router capacity is set to 10 Mbps, and it uses a DropTail queue. In order to explore the worst-case scenario, we experiment with Trace-I which is dominated by short flows. We then create (in two separate experiments) both TCP and UDP flash crowds to evaluate their impact on end-to-end performance. In both cases, we set the link utilization to around 80% of the bottleneck capacity in absence of the flash crowd. Then, we create pareto ON/OFF flash crowd traffic with average burst period of 16.0 sec and idle period of 8.0 sec. During ON periods new TCP flows arrive at a high rate (200 connections a second for TCP flash crowd) creating a permanent packet loss rate of about 25%; in the UDP case, the burst suddenly congests the bottleneck router and creates permanent packet loss rate of about 45% during ON periods.

Figure 6(a) and 6(b) show the flow response times in presence of UDP and TCP flash crowds. We indeed confirm that TCP’s RTO estimator may lag behind RTT in such extreme scenarios, *i.e.*,  $RTO < RTT$ . This is particularly the case at a flash crowd’s early stage when the RTT quickly jumps. Still, Figures 6(a) and 6(b) show no negative impact on response times relative to the TCP case. Indeed, as soon as new RTT samples are obtained, the RTO successfully adjusts to the new environment, despite extremely heavy congestion. Moreover, due to more aggressive behavior, TCP( $\infty$ ) substantially outperforms TCP in such cases.

### 5. MULTIPLE BOTTLENECKS AND COMPLEX TOPOLOGIES

Independent of where a flow may be bottlenecked on an end-to-end path, *i.e.*, upstream or downstream, we showed that as long as there exists a single bottleneck on an end-to-

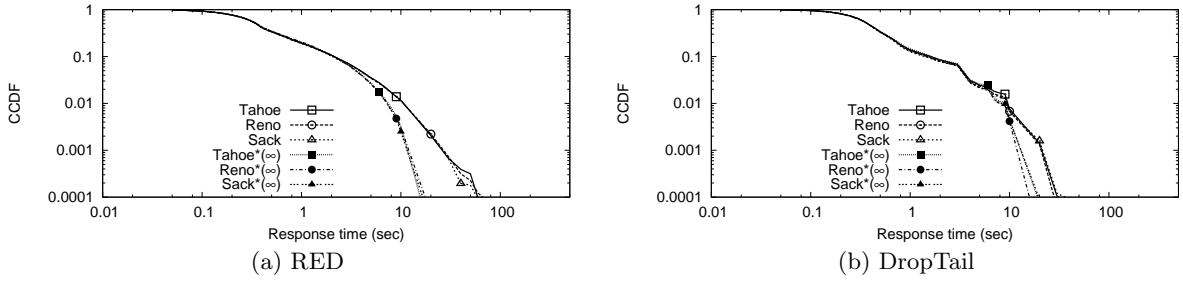


Figure 5: Effect of queuing disciplines and TCP variants on response time distributions

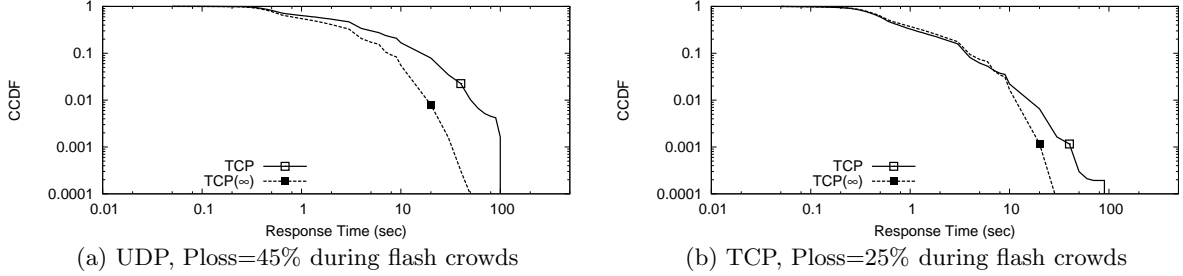


Figure 6: Effect of flash crowd

end path, the backoff-less approach can only improve end-to-end performance, irrespective of the TCP versions and queuing disciplines.

**Dead packets.** One concern is that when a bottleneck is located downstream, *i.e.*, closer to the destination, a more aggressive endpoint can generate a larger number of so-called “dead packets;” *e.g.*, those that exhaust network resources upstream, but are then dropped downstream.

There are two issues with respect to this problem. First, even if an upstream router’s bandwidth is wasted by “dead packets,” this does not necessarily mean that this bandwidth would have otherwise been used by other flows. Indeed, as long as there exists a single bottleneck and endpoints adhere to the implicit packet conservation principle, no other flow in the network will suffer due to the dead packet issue. Second, if there exist multiple bottlenecks on an end-to-end path, then there is indeed a chance that dead packets impact the performance of flows sharing the upstream bottleneck. We explore such scenarios in depth below.

### 5.1 The Impact of Dead Packets on Network Efficiency

Here, we develop a simple model to understand the impact of dead packets on network efficiency in multiple bottleneck scenarios. For simplicity, we stick with a two-bottleneck topology (Figure 7). Still, our analysis is extensible to scenarios involving an arbitrary number of bottlenecks.

Denote by  $p_1$  and  $p_2$  the packet drop ratios at the bottleneck links  $R1 - R2$  and  $R3 - R4$ , respectively. Denote by  $L_0$  the load at link  $R1 - R2$  generated by the  $S0 - C0$  flows. Similarly, denote by  $L_1$  and  $L_2$  loads at links  $R1 - R2$  and  $R3 - R4$  generated by  $S1 - C1$  and  $S2 - C2$  flows, respectively. Denote by  $\alpha$  the fraction of  $R1 - R2$  link capacity occupied by dead packets belonging to  $S0 - C0$  flows. Assuming random drops, it could be shown that

$$\alpha = \left( \frac{L_0}{L_0 + L_1} \right) (1 - p_1)p_2. \quad (7)$$

Eq. (7) shows that the fraction of dead packets is a function

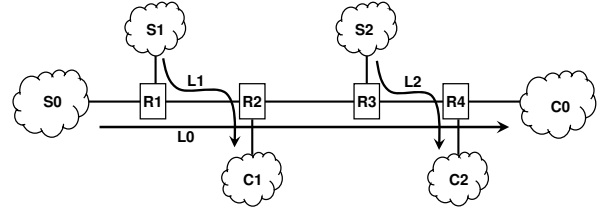


Figure 7: A typical two-bottleneck scenario

of packet loss ratios at the two bottlenecks, and a function of  $L_0/(L_0 + L_1)$ , which represents the fraction of flows experiencing multiple bottlenecks on their end-to-end paths.

Figure 8 plots  $\alpha$  as a function of  $L_0/(L_0 + L_1)$  with  $(p_1, p_2)$  as parameters. We show three scenarios: (i) (1%, 1%), which represents a lightly congested scenario at both bottlenecks; (ii) (5%, 5%), which is a highly congested scenario; and (iii) (1%, 5%), in which packet loss rate at the downstream bottleneck (5%) is higher than the one at the upstream bottleneck (1%). Expectedly, Figure 8 shows that dead packets have the maximum impact in scenarios when the downstream bottleneck is heavily congested. The figure also shows that the percent of dead packets increases as the fraction of multiple-bottlenecked flows,  $L_0/(L_0 + L_1)$ , increases. Thus, to realistically understand the impact of dead packets, it is necessary to have an estimate of the fraction of multiple-bottlenecked flows in today’s Internet. Because this question is beyond the direct scope of our work here, we rely upon existing research results.

Measurements studies have shown that bottlenecks typically reside at Internet edges, either at access networks [12] or at lower-tier ASes [3, 19]. A recent study indicates that less than 5% of flows experience multiple bottlenecks on an end-to-end path at time-scales of minutes [11]. Projecting this result to our analysis (point 0.05 on the x-axis in Figure 8), it follows that the percent of dead packets that can impact upstream flows is indeed marginal in today’s Internet. As an example, even in the worst-case scenario from Figure 8, the fraction of such packets is very small, 0.002475.

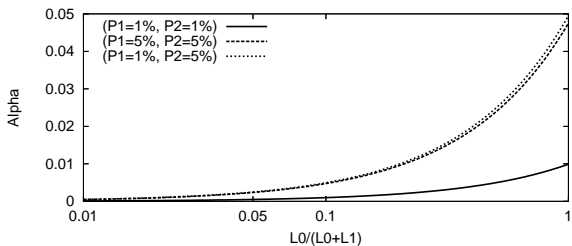


Figure 8: Fraction of dead packets as a function of the fraction of multiple-bottlenecked flows

## 5.2 The Impact of Dead Packets on End-to-end Performance

Here, we answer the following two questions. First, what would happen if the percent of multiple-bottlenecked flows would increase dramatically above the level measured in today’s Internet? And second, what would be the impact of the backoff-less TCP approach on end-to-end performance in such scenarios? To answer these two questions, we rely upon Emulab experiments in which we increase the percent of multiple-bottlenecked flows.

We create a topology equivalent to the one shown in Figure 7. Clients  $C_0$ ,  $C_1$ , and  $C_2$  send HTTP requests to servers  $S_0$ ,  $S_1$ , and  $S_2$ , respectively. The bandwidth of the bottleneck links ( $R_1 - R_2$  and  $R_3 - R_4$ ) are set to 10 Mbps and all other links have the capacity of 50 Mbps. As indicated above, we distribute the load such that the  $L_0/(L_0 + L_1)$  ratio becomes 0.25, five times above the level in today’s Internet.

Figure 9 (a), (b), and (c) plot the response times distribution of multiple bottlenecked  $S_0 - C_0$  flows, as well as  $S_1 - C_1$  flows.  $S_1 - C_1$  flows share the upstream bottleneck with  $S_0 - C_0$  flows and hence could be negatively impacted when backoff-less TCP is applied.

Figure 9 (a) shows that removing the exponential backoff altogether improves the response times distributions of both  $S_0 - C_0$  and  $S_1 - C_1$  flows for Trace-I. This is because in such an environment, which is entirely dominated by short flows, long pauses due to exponential backoff only degrade these flows’ response times. With Trace-II, the  $S_0 - C_0$  flows improve their response times, while the  $S_1 - C_1$  backoff-less flows only marginally degrade relative to the TCP case. In particular, many long flows along with the short flows in Trace-II also become aggressive, which increases the competition at the upstream link, and affects the overall performance of  $S_1 - C_1$  flows.

The result is similar for Trace-III. In particular, the fraction of flows that complete their transfers in less than 100 seconds increases with backoff-less TCP stack for both  $S_0 - C_0$  and  $S_1 - C_1$  flows. One interesting detail is that the CCDF curve for TCP  $S_0 - C_0$  flows (marked by TCP/0) is shorter than for other stacks. This is because the number of long flows which finish their transfers during the experiment is negligible compared to the backoff-less case.

In summary, the results show that even in environments with a large percent of multiple-bottlenecked flows, such flows improve their overall response times. More importantly, this is achieved *without* causing any catastrophic effects on other flows in the network.

## 5.3 Realistic Network Topologies

Here, we conduct a large-scale simulation experiment to understand the overall effects of removing TCP’s exponen-

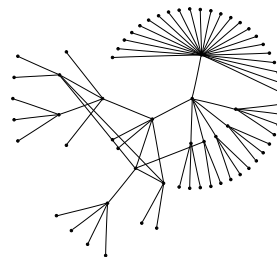


Figure 10: An Orbis-scaled HOT topology

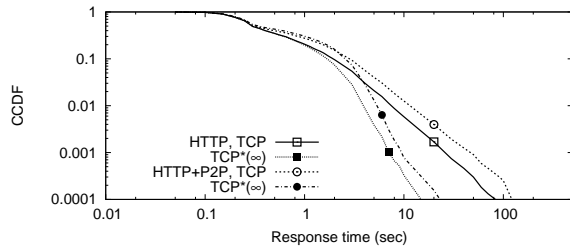


Figure 11: Impact of topology and traffic matrix

tial backoff mechanism in a more complex network environment. In particular, we simulate a scaled, yet realistic, network topology and explore the impact of heterogeneous access bandwidths. We also consider a more complex traffic matrix, which includes both HTTP and p2p-like traffic.

We start from the HOT model [30] to generate a realistic network topology. Next, we use the Orbis approach [32] to scale down the network size while still preserving the original structure. Figure 10 shows a scaled topology which consists of 67 nodes. We randomly select 10% of the edge nodes to be server nodes and the remaining 90% become client nodes. We then augment the core links with 10 Gbps bandwidth, server-side edge links with 100 Mbps, and we randomly distribute the client-side edge links in the 1-10 Mbps range. The delay of all links is set to 10 ms. The routers use RED with the targeted delay parameter set to 5 ms.

To generate traffic between clients and servers, we use the synthetic traffic generator from [1]. Next, to add p2p (BitTorrent-like) traffic in the network, we proceed as follows. Each client downloads an infinite-size file from five randomly chosen peers (from the client set) in the network, thus increasing the load in the network.

Figure 11 plots the results. We aggregate the response times at the clients (for the client-server communication) and plot the results. The insights are the following. First, in absence of p2p traffic, the response times distribution improves significantly. As an example, the maximum response time decreases by five times (200 sec for the ‘HTTP, TCP’ case, and 40 sec for the corresponding TCP\*( $\infty$ ) case). Second, in presence of p2p traffic, the response times further increase. The maximum response time improves by an order of magnitude after the removal of the exponential backoff in this scenario. When p2p traffic is present, short TCP flows experience more timeouts and run into backoff as the client-access links are always congested due to long-lived p2p flows. On the contrary, TCP\*( $\infty$ ) flows avoid such long backoffs.

## 6. INCREMENTAL DEPLOYMENT

Here, we explore the potential for deploying these changes in the Internet. Figures 12(a), 12(b), and 12(c), show the response times for different TCP variants, *i.e.*, TCP\*( $\infty$ ) (0.2, 3) and TCP (0.2, 3), when they *multiplex* together.

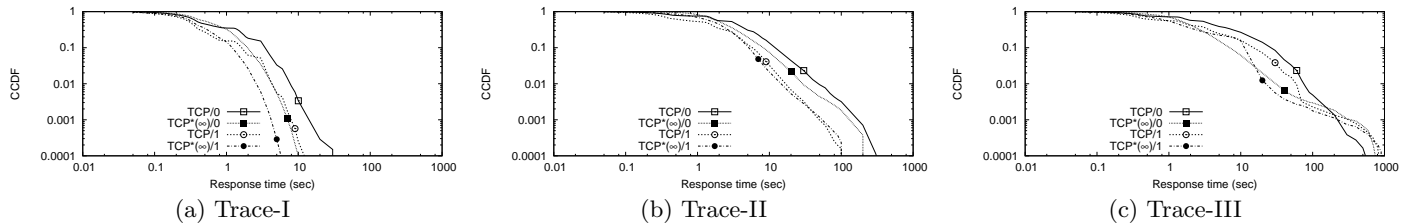


Figure 9: Effect of dead packets on end-to-end performance

We show the results for RED, and confirm that results are consistent with Droptail. We change the percent of each protocol from 10% to 90% as indicated in the figure, and show the CCDF profiles for all the traces. All figures show that as the percent of TCP\*( $\infty$ ) flows increases, the CCDF curves shift towards right, *i.e.*, longer response times. Given that the bottleneck bandwidth is fixed, an increased number of backoff-less flows results in a firmer competition among such flows, shifting the CCDF curve towards right.

Next, an increased number of TCP\*( $\infty$ ) flows in the system necessarily degrades the performance of TCP flows. Unfortunately, in some cases by a non-negligible amount, which may sometimes cause TCP flows to starve. In particular, when TCP\*( $\infty$ ) contributes 90% among all flows in the system, TCP flows suffer the most, particularly in Trace-I and -II scenarios. As an example, the mean response time *doubles* relative to the scenario when there are no TCP\*( $\infty$ ) flows for Trace-II.

Our additional results indicate that a two-step incremental deployment process is much more feasible. We repeat the same experiments as above, only this time we multiplex (a) TCP and TCP\*(3) flows in the first case, and (b) TCP\*(3) and TCP\*( $\infty$ ) in the second case. Our results reveal that choosing TCP\*(3) as an intermediate stack smoothens the incremental-deployability process significantly. The ratio between average response times is reduced to 1.2 between corresponding TCP scenarios when TCP\*(3) is applied for the most critical Trace-II scenario; likewise, the average response time increases by a factor of 1.4 for the TCP\*(3) stack when TCP\*( $\infty$ ) is introduced in the system, again for Trace-II. We conclude that deploying the change, *i.e.*, removing the exponential backoff algorithm in two steps is feasible. Still, if the above increase in response times for legacy TCP versions becomes a concern, this can be addressed by applying a larger number of incremental steps.

## 7. DISCUSSION AND RELATED WORK

**TCP behavior with many flows.** Morris [35] and Qiu *et al.* [38] explored the performance of TCP in scenarios when the number of active flows exceed the bandwidth-delay product. While our findings here are orthogonal to theirs, our work differs in the sense that we fundamentally challenge the need for exponential backoff in the TCP protocol, while they evaluate TCP’s performance in scenarios with a large number of flows. Such evaluations are performed in moderate bandwidth-delay-product environments and with long-lived TCP flows. As we showed in this paper, skewed flow-size distributions and dynamic flow arrivals shed new light on system’s performance.

**DDoS defense by offense.** Our work closely relates to the work of Walfish *et al.* [42], which addresses the problem of application-level distributed denial-of-service attacks against Internet servers. The authors propose a victimized server to encourage all clients to automatically send higher

volumes of traffic, thus out-crowding bad clients and capturing a much larger fraction of the server’s resources. The authors show that as long as the endpoints uphold the principle of request conservation, *i.e.*, the client retries only in response to a message from the server, and the server sends the message when it receives a request, the server resources will be fully utilized across all clients, despite a high request rate. We show that a similar idea is applicable to the *network bandwidth*. As long as clients uphold the implicit packet conservation principle (Section 2.3), the bandwidth resources will be fully utilized and the *end-to-end* performance will not suffer, despite potentially heavy congestion.

**Decongestion control.** Our work closely relates to the recent work of Raghavan and Snoeren [39]. Placing their efforts in the context of the future Internet design, the authors argue that a protocol that relies upon greedy, high-speed transmission of data, endpoint erasure data coding, and fair packet dropping at routes, has the potential to outperform current TCP performance. The commonality between decongestion control and our work lies in promoting a more aggressive endpoint behavior, much more aggressive in the decongestion control scenario, and bounded by the implicit conservation principle in our case. The key difference is that we do not place our efforts in the context of the future Internet design, but rather attempt to improve Internet’s performance “right here, right now.” Unlike decongestion control, our approach can be incrementally deployed at endpoints, and it requires no additional functionality at routers.

## 8. CONCLUSIONS

In this paper, we developed an improved understanding of the TCP’s exponential backoff algorithm, a fundamental piece of congestion control as it has been instantiated in the Internet over the last 20 years. Our key contribution is not in bringing massive performance improvements, but rather in radically increasing our knowledge about congestion control fundamentals. In particular, (i) we made the case for removing the exponential backoff algorithm from TCP and opened the doors for re-evaluating other well-accepted pieces of this congestion control algorithm. (ii) We introduced the implicit packet conservation principle and showed that end-to-end performance can only improve as long as the endpoints uphold the principle. (iii) We demonstrated that this result holds for all explored TCP variants, bottleneck rates, queuing disciplines, and traffic distributions, with gains being particularly emphasized for distributions dominated by short flows. (iv) We showed that the proposed backoff-less TCP approach can improve end-to-end performance even in multiple bottleneck scenarios, and increase network utilization for general network topologies. (v) Finally, we showed that abandoning TCP’s exponential backoff algorithm is an incrementally-deployable two-step task.

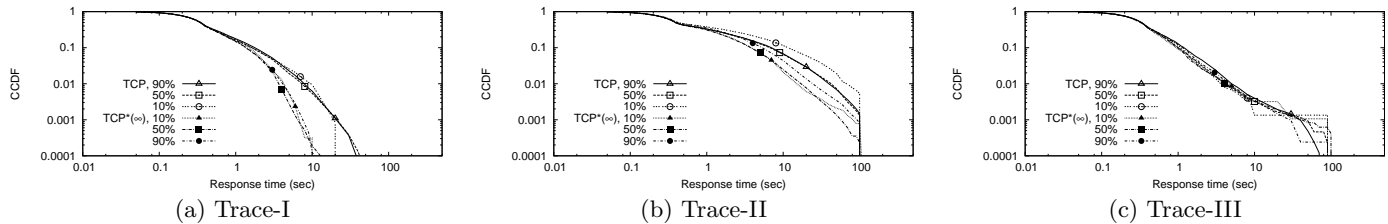


Figure 12: Incremental deployability

## Acknowledgements

We would like to thank Mark Allman, Craig Partridge, and Sally Floyd for their critical comments about this paper.

## 9. REFERENCES

- [1] Packmime: Statistical modeling of connection request variables. <http://stat.bell-labs.com/InternetTraffic/packmime.html>.
- [2] Transmission control protocol. RFC 793.
- [3] A. Akella, S. Seshan, and A. Shaikh. An Empirical Evaluation of Wide-Area Internet Bottlenecks. In *ACM SIGMETRICS '03*.
- [4] D. Aldous. Ultimate instability of exponential back-off protocol for acknowledgement based transmission control of random access communication channels. *IEEE Transactions of Information Theory*, IT-33(2), Mar. 1987.
- [5] M. Allman, S. Floyd, and C. Partridge. Increasing TCP's initial window. RFC 3390.
- [6] M. Allman and V. Paxson. On estimating end-to-end network path properties. In *ACM SIGCOMM '99*.
- [7] T. Anderson, A. Collins, A. Krishnamurthy, and J. Zahorjan. PCP: Efficient endpoint congestion control. In *ACM NSDI '06*.
- [8] G. Appenzeller, I. Keslassy, and N. McKeown. Sizing router buffers. In *ACM SIGCOMM '04*.
- [9] L. Breslau, E. Knightly, S. Shenker, I. Stoica, and H. Zhang. Endpoint admission control: Architectural issues and performance. In *ACM SIGCOMM '00*.
- [10] J. Cao, W. Cleveland, Y. Gao, K. Jeffay, F. Smith, and M. Weigl. Stochastic models for generating synthetic HTTP source traffic. In *IEEE INFOCOM '04*.
- [11] L. Deng and A. Kuzmanovic. Pong: Diagnosing spatio-temporal Internet congestion properties. In *ACM SIGMETRICS '07*.
- [12] M. Dischinger, A. Haeberlen, K. Gummadi, and S. Saroiu. Characterizing residential broadband networks. In *ACM IMC '07*.
- [13] N. Dukkupati and N. McKeown. Why flow-completion time is the right metric for congestion control. *ACM CCR*, 36(1):59–62, 2006.
- [14] S. Floyd. Highspeed TCP for large congestion windows, Dec. 2003. Internet RFC 3649.
- [15] S. Floyd, M. Allman, A. Jain, and P. Sarolahti. Quick-start for TCP and IP, Oct. 2007. Internet-draft.
- [16] S. Floyd and K. Fall. Promoting the use of end-to-end congestion control in the Internet. *IEEE/ACM ToN*, 7(4):458–472, Aug. 1999.
- [17] L. Goldberg, M. Jerrum, S. Kannan, and M. Paterson. A bound on the capacity of backoff and acknowledgement-based protocol. *TR 365, Dept. of Computer Science, University of Warwick, UK*, Jan. 2000.
- [18] L. Guo and I. Matta. The war between mice and elephants. In *IEEE ICNP '01*.
- [19] N. Hu, L. Li, Z. Mao, P. Steenkiste, and J. Wang. Locating Internet bottlenecks: Algorithms, measurements, and implications. In *ACM SIGCOMM '04*.
- [20] V. Jacobson. Congestion avoidance and control. *ACM CCR*, 18(4):314–329, August 1988.
- [21] C. Jin, D. Wei, and S. Low. FAST TCP: Motivation, architecture, algorithms, performance. In *IEEE INFOCOM '04*.
- [22] T. Karagiannis, A. Broido, N. Brownlee, k. claffy, and M. Faloutsos. Is p2p dying or just hiding? In *IEEE GLOBECOM '04*.
- [23] P. Karn and C. Partridge. Improving round-trip time estimates in reliable transport protocols. *ACM ToCS*, 9(4):364–373, Nov. 1991.
- [24] F. Kelly. Stochastic models of computer communication systems. *Journal of Royal Statistical Society*, B 47(3):379–395, 1985.
- [25] T. Kelly. Scalable TCP: Improving performance in highspeed wide area networks. *ACM CCR*, 32(2), Apr. 2003.
- [26] T. Kelly, S. Floyd, and S. Shenker. Patterns of congestion collapse, 2003. <http://www.icir.org/floyd/papers/patterns.pdf>.
- [27] A. Kortebe, L. Muscariello, S. Oueslati, and J. Roberts. Evaluating the number of active flows in a scheduler realizing fair statistical bandwidth sharing. In *ACM SIGMETRICS '05*.
- [28] A. Kuzmanovic. The power of explicit congestion notification. In *ACM SIGCOMM '05*.
- [29] L. Le, J. Aikat, K. Jeffay, and F. Smith. The effects of active queue management on Web performance. In *ACM SIGCOMM '03*.
- [30] L. Li, D. Alderson, W. Willinger, and J. Doyle. A first-principles approach to understanding the Internet's router-level topology. In *ACM SIGCOMM '04*.
- [31] D. Liu, M. Allman, S. Jin, and L. Wang. Congestion control without a startup phase. In *PFLDnet '07*.
- [32] P. Mahadevan, C. Hubble, D. Krioukov, B. Huffaker, and A. Vahdat. Orbis: Rescaling Degree Correlations to Generate Annotated Internet Topologies. In *ACM SIGCOMM '07*.
- [33] A. Medina, M. Allman, and S. Floyd. Measuring the evolution of transport protocols in the internet. *ACM CCR*, 35(2):37–52, 2005.
- [34] R. Metcalfe and D. Boggs. Ethernet: Distributed packet switching for local computer networks. *ACM Communication*, 19(7):395–404, July 1976.
- [35] R. Morris. TCP behavior with many flows. In *IEEE ICNP '97*.
- [36] J. Nagle. Congestion control in IP/TCP internetworks. *ACM CCR*, 14(4):11–17, 1984.
- [37] V. Paxson and M. Allman. Computing TCP's retransmission timer. RFC 2988.
- [38] L. Qiu, Y. Zhang, and S. Keshav. On individual and aggregate TCP performance. In *IEEE ICNP '99*.
- [39] B. Raghavan and A. Snoeron. Decongestion control. In *ACM HotNets-V '06*.
- [40] S. Rewaskar, J. Kaur, and F. D. Smith. A performance study of loss detection/recovery in real-world TCP implementations. In *IEEE ICNP '07*.
- [41] P. Sarolahti and M. Kojo. Forward RTO-Recovery (F-RTO): An Algorithm for Detecting Spurious Retransmission Timeouts with TCP and the Stream Control Transmission Protocol (SCTP), Aug. 2005. Internet RFC 4138.
- [42] M. Walfish, M. Vutukuru, H. Balakrishnan, D. Karger, and S. Shenker. DDoS defense by offense. In *ACM SIGCOMM '06*.