1 Review A

```
 I reviewed the paper and I had only two comments, but unfortunately
they are both substantive, and at least the first one needs to be addressed
before the paper is published.

1.  The paper doesn't deal with the effects of Karn's algorithm.
Recall that Karn's algorithm changes the RTO algorithm if a segment whose RTT
is being measured is lost.  As Karn's algorithm isn't discussed, the RTO
discussion in section II is incomplete and the understanding of attack
periods is also incomplete.  NOTE: I don't think Karn's algorithm actually
affects the attack -- as Karn's algorithm simply requires using a multiple of
the original RTO as the new RTO.  But the point needs to be discussed and the
relationship described as Karn's algorithm is an integral part of the RTT
estimator unless PAWS is turned on.  [Karn's algorithm is published in ACM
TOCS, Nov 1991 and/or Proc. ACM SIGCOMM '87]
```

———

(*) We have cited the above paper (now reference [21]), and changed the text in Section 2 to better explain Karn's algorithm:

" At this moment, the sender enters the exponential backoff phase: it reduces the congestion window to one, doubles the RTO value to 2 seconds according to Karn's algorithm [21], retransmits the unacknowledged packet with sequence number n, *and resets the retransmission timer with this new RTO value.*

*If the packet is lost again (*not *shown in Figure 1), exponential backoff continues as the sender waits for the 2 sec-long retransmission timer to expire. At* $t = 3\,sec$*, the sender again applies Karn's algorithm, doubles the RTO value to 4 seconds and repeats the process."*

In addition, we have changed the text and discussed the relationship between Karn's algorithm and the attack in Section 3.

"If the attacker creates a second outage between time 1 and $1 + 2\,RTT$*, it will force TCP to wait another 2 sec. By creating similar outages at times 3, 7, 15,* $\cdots$*, an attacker could exploit Karn's algorithm and deny service to the TCP flow while transmitting at extremely low average rate.*

While potentially effective for a single flow, a DoS attack on TCP aggregates in which flows continually arrive and depart requires periodic (vs. exponentially spaced) outages at the minRTO time-scale. Moreover, if all flows have an identical minRTO parameter as recommended in RFC 2988 [32], the TCP flows can be forced into continual timeouts if an attacker creates periodic outages."

———

```
2.   The second issue is that while the paper looks at varying the original
RTO value, an open question is what would happen if you varied the multiplier
for RTO backoff such that it was no longer a single integer but some
fractional multiplier, uniformly distributed -- would this not break the RTO
```

```
effect?
```

———

(*) This is an interesting approach. However, as shown in Figure 1, as long as the period of the attack is longer than 1 sec + 2 RTTs, the attacker is able to fully mitigate the effects of Karn's algorithm; thus, the proposed use of fractional multiplier would be inefficient in such scenarios.

———

2 Review B

```
This paper provides a rather broad and thorough exploration
of the effects of pulsed attack traffic (or non-congestion
controlled cross traffic) on TCP flows. It brings a combination
of analysis, simulation and Internet experimentation to bear on
the problem. It finds that relatively low aggregate rate attack
traffic can have a disproportionately large adverse effect due
to TCP retransmission mechanisms, and especially the minRTO.

I have no substantial issues with the correctness or
completeness of the paper, and so I believe the paper is ready
for publication more-or-less as is. However, I do have several
comments on the point-of-view taken in the paper and suggest
several steps that the authors can take to improve their paper.

Most importantly, the paper is weakened by resting too much of
its case on [2]. A recommendation that minRTO be set to 1 second
doesn't seal the issue of what makes a good choice in practice.
There may be little practical difference between a minRTO of 1
second or half a second or less, and a smaller choice would then
be sensible in light of your attacks! This issue becomes
problematic to the reader in VI when it becomes apparent that
many TCPs do not implement minRTO as 1 second but do something
else (unspecified) which for all I know is more successful at
deterring shrew attacks. A better approach for your paper would
be to 1) analyze the dependence on minRTO, 2) note the tension
that prevents minRTO being either very large or very small and
3) give us an appreciation of what happens to the results if
minRTO is shrunk by a factor of 2, 4, or 10. How small would
minRTO need to be to negate the attacker's advantage? You
basically do 1) and 2) in VII.B (which is why I rate your paper
as an accept already) but you punt on 3). Note that I perceive
the value of your paper as lying in the characterization of this
space, rather than the description of an attack that cannot be
countered. So I do not really care if a minRTO 10x smaller would
```

```
mostly solve the problem -- that is what I want to know.
```

———

(*) Undoubtedly, decreasing minRTO to a small value (*e.g.*, 50 ms) would solve the DoS part of the problem. This is clearly indicated by Equation (4) for the single-flow case (*e.g.*, T=a=b=50 ms), and by Equation (8) for flow aggregates. We change the text in Section 7 to address this issue explicitly.

"There are two apparent strategies for increasing throughput on $T = b$ time-scales. First, it appears attractive to decrease a which would significantly increase TCP throughput. Alternatively, decreasing both a and b would force the attacker to send very close bursts, making it no longer a stealthy attack. However, recall that conservative timeout mechanisms are fundamentally required to achieve high performance during periods of heavy congestion [2]."

As indicated by the last sentence in the above paragraph, decreasing minRTO would necessarily reduce the protocol performance in the absence of attack. The main goal of our Section 7 was to explore these tensions between performance and security.

From the practical point of view, the fact that many TCPs do not implement minRTO as 1 second[1] does not mean that the attacks are artificial. Instead, our contribution lies in showing that while increasing minRTO improves protocol performance, it also opens a severe security hole.

———

```
A general issue is that the writing has gotten rather stiff with
boilerplate in various places, e.g., at the start of sections
with an introduction that gets repeated later. This is not
conducive to a good read. An edit pass in which you try and tell
it more like a story can fix this and leave you with a more
memorable paper!
```

———

(*) We have made an edit pass and addressed the above issue wherever possible.

———

```
I found it curious that SACK was not considerably more effective
than other TCP variants, and in fact that SACK is vulnerable to
the shrew attack in the first place. Why? (This is an example of
an interesting issue that you gloss over while providing all the
boilerplate about experiments and outcomes instead.) As long as
some packets are getting through and some ACKs are getting back,
it would seem that a TCP SACK flow should be able to make some
progress rather than stall with a timeout. But evidently this is
not so. Why? Is it an artifact of how SACK has been added to TCP
as advisory only, or a reflection of an implicit assumption on
reasonable drop rates? Perhaps this is an opportunity for a more
robust transport protocol?
```

[1]but rather 200 ms. We have fixed the text in Section 6 to indicate this explicitly.

(*) Shrew attacks create short, but extremely high packet loss-ratio events. During such outages, if all packets from the window of data are lost, TCP (including Sack) has no alternative but to wait for a retransmission timer to expire. Moreover, the key problem is that after a timeout, it takes each flow some time to regain their fair share rate of the network bandwidth. In such scenarios, loss of a small fraction of packets from the window of data will enforce another timeout. We address this issue in the text.

"Figure 13 also indicates that all TCP variants, including Sack, are the most vulnerable to DoS in the 1 - 1.2 sec time-scale region. During this period, TCP flows are in slow-start and have small window sizes such that a smaller number of packet losses are needed to force them to enter the retransmission timeout."

```
The above issue is one thread that leads me to believe that your
paper is stronger on the impact of shrew attacks than at
strategies for countering them. It seems plausible that better
transport protocol or router solutions will be found to mitigate
shrew attacks, and at that time the title of your paper may seem
a misnomer (since it implies that you provide effective counter
strategies).
```

(*) Developing better transport protocols and better router solutions to counter the attack is certainly possible, yet beyond the scope of this paper. We explored an important class of existing counter-DoS mechanisms as well as some "first-cut" attack-specific solutions (*e.g.*, minRTO randomization); in that context, we thought that it was reasonable to reflect this in the title.

```
Finally, as a nit, the DOS flow filtering result could simply be
given in English rather than as an unproven formalism. I don't
think the formalisms you provide are buying you much.
```

(*) The "DoS flow filtering result" was written formally in order to be consistent with the previous "DoS TCP throughput result."

---

3 Review C

```
My comments mainly concern two sections in the paper (the
rest of it is already on top of things in my view):
```

Section VII-A is the weakest part of the paper (unfortunate given the title change of the paper). I was happy to see that you simulated CHOKe in this version but there are some basic issues that the paper does not completely get at:

- seems to me that scheduling schemes such as FQ should completely eliminate the shrew. but you investigate only preferental dropping which is fine as long as you clearly state that:

1. scheduling will work (if you believe that claim).
2. are only investigating preferential dropping because you are curious as to whether simpler strategies would work (and not because they have similar power).

- more broadly, the effectiveness of the shrew attack depends on the timescales at which fairness is enforced. scheduling schemes do so at smaller timescales and are more effective (at some cost) as a result . preferential dropping schemes do so at longer timescales and are less effective as a result. this is why CHOKe is more effective than RED-PD. and maybe why SFB might be more effective than CHOKe.
- as a nit, referring to [24] repeatedly is a bit annoying. you should be doing what you think is right and not becuase [24] said something. especially because their statements were probably made in the context of a particular form of bad flows - those with high average rate of sending.

(*) The reviewer correctly observes that a packet-level FQ scheduling has the capacity to eliminate the effects of the shrew attack. However, there are two problems with the FQ approach. First, achieving *per-flow* packet-level FQ has serious scalability problems which prevents its deployment. Indeed, exactly due to these problems, researchers developed *scalable* schemes such as CHOKe or RED-PD, on which we focus in the paper. Second, if the attacker spoofs packets or performs a distributed shrew attack, then even packet-level FQ would *not* help in such scenarios. Because we focus on the generic shrew attack in the paper, we do not explore spoofed or distributed shrew attacks. We have change the text in Section 7 to address this issue explicitly.

"Mechanisms for per-flow treatment at the router can be classified as scheduling or preferential dropping. Scheduling algorithms place flows in different logical partitions and determine the service rate received by each partition. For example, per-flow Fair Queuing (FQ) scheduling (e.g., [5]) treat each flow as a single partition. While such an approach would completely protect the system against the shrew attack (unless the attacker performs a distributed shrew attack or spoofs packets), per-flow FQ has serious scalability limitations which prevent its deployment. Due to implementation simplicity and other advantages of preferential dropping over scheduling, we focus on dropping algorithms for detection of DoS flows and/or achieving fairness among adaptive and non-adaptive flows."

However, the price for scalability of preferential dropping schemes such as CHOKe and RED-PD is their inability to achieve fairness on very short (packet-level) time-scales, and to protect the system against the shrew attacks. We have fixed the text regarding reference [24], now [26].

———

```
Compared to the rest of the paper, which is very well-written,
Section IV is a bit confusing. Here are my sources of
confusion:
- I don't buy the optimality claim. Maybe the optimal pattern
is something completely different. Moreover, given the later
results that the shrew need not attack at C means that what you
propose might not be optimal. Maybe you are better off calling
it, for instance, optimized or smarter shrew.
- Section IV.B should be minimum *packet* (DoS streams)
instead of minimum *rate* since you are minimizing the number of
packets and not the rate?
- the term "double-rate" was confusing. there is no doubling
involved. maybe two-rate?
- call B the buffer size instead of queue size?
```

———

(*) We have fixed all of the above.

———

```
Out of curiosity, why didn't you simulate 1.5 in fig 19 instead
of 1.2? especally because some OS's might be doing 1.5 already.
results with a larger b would also be interesting.
```

———

(*) It's been a while since we plotted this figure. Most probably, we picked it because it looked better than others.

———