TCP Maintenance and Minor Extensions
Internet-Draft
Intended status: Experimental
Expires: January 4, 2015

T. Moncaster, Ed.
University of Cambridge
B. Briscoe
A. Jacquet
BT

July 03, 2014

# A TCP Test to Allow Senders to Identify Receiver Non-Compliance
draft-moncaster-tcpm-rcv-cheat-03

## Abstract

The TCP protocol relies on receivers sending accurate and timely feedback to the sender. Currently the sender has no means to verify that a receiver is correctly sending this feedback according to the protocol. A receiver that is non-compliant has the potential to disrupt a sender’s resource allocation, increasing its transmission rate on that connection which in turn could adversely affect the network itself. This document presents a two stage test process that can be used to identify whether a receiver is non-compliant. The tests enshrine the principle that one shouldn’t attribute to malice that which may be accidental. The first stage test causes minimum impact to the receiver but raises a suspicion of non-compliance. The second stage test can then be used to verify that the receiver is non-compliant. This specification does not modify the core TCP protocol - the tests can either be implemented as a test suite or as a stand-alone test through a simple modification to the sender implementation.

## Status of This Memo

## Copyright Notice

## Table of Contents

## 1. Introduction

This document details an experimental test designed to allow a TCP sender to identify when a receiver is misbehaving or is non-compliant. It uses the standard wire protocol and protocol semantics of basic TCP [RFC0793] without modification. The hope is that if the experiment proves successful then we will be able to obsolete the experimental TCP nonce [RFC3540], hence freeing up valuable codepoints in both the IPv4 header and the TCP header.

When any network resource (e.g. a link) becomes congested, the congestion control protocol [RFC5681] within TCP/IP expects all receivers to correctly feed back congestion information and it expects each sender to respond by backing off its rate in response to this information. This relies on the voluntary compliance of all senders and all receivers.

Over recent years the Internet has become increasingly adversarial. Self-interested or malicious parties may produce non-compliant protocol implementations if it is to their advantage, or to the disadvantage of their chosen victims. Enforcing congestion control when trust can not be taken for granted is extremely hard within the current Internet architecture. This specification deals with one specific case: where a TCP sender is TCP compliant and wants to ensure its receivers are compliant as well.

Simple attacks have been published showing that TCP receivers can manipulate feedback to fool TCP senders into massively exceeding the compliant rate [Savage]. Such receivers might want to make senders unwittingly launch a denial of service attack on other flows sharing part of the path between them [Sherwood]. But a more likely motivation is simple self-interest---a receiver can improve its own download speed with the sender acting as an unwitting accomplice. [Savage] quotes results that show this attack can reduce the time taken to download an HTTP file over a real network by half, even with a relatively cautious optimistic acknowledgemnt strategy.

There is currently no evidence that any TCP implementations are exploiting any of the attacks mentioned above. However this may be simply because there is no widely available test to identify such attacks. This document describes a test process that can identify such non-compliance by receivers should it start to become an issue. The aim of the authors is to provide a test that is safe to implement and that can be recommended by the IETF. The test can be deployed as a separate test suite, or in existing senders, but this document does not mandate that it should be implemented by senders.

The measures in this specification are intended for senders that can be trusted to behave. This scheme can not prevent misbehaving senders from causing congestion collapse of the Internet. However the very existence of a test scheme such as this should act as a disincentive against non-compliant receivers.

Senders do not have to be motivated solely by "the common good" to deploy these changes. It is directly in their own interest for senders serving multiple receivers (e.g. large file servers and certain file-sharing peers) to detect non-compliant receivers. A large server relies in part on network congestion feedback to efficiently apportion its own resources between receivers. If such a large server devotes an excessive fraction of its own resources to non-compliant receivers, it may well hit its own resource limits and have to starve other half-connections even if their network path has spare capacity.

The proposed tests do not require the receiver to have deployed any new or optional protocol features, as any misbehaving receiver could simply circumvent the test by claiming it did not support the optional feature. Instead, the sender emulates network re-ordering and then network loss to test that the receiver reacts as it should according to the basic TCP protocol. It is important that the level of emulated re-ordering that such a test introduces should not adversely impact compliant receivers.

This document specifies a two-stage test in which the sender deliberately re-orders some data segments so as to check if the destination correctly acknowledges out-of-order segments. The first stage test introduces a small reordering which will have a related very minor performance hit. It is not a conclusive test of compliance. However, failing it strongly suggests the receiver is non-compliant. This raises sufficient suspicion to warrant the more intrusive but conclusive second stage if this non-compliance is going to be sanctioned. The second stage proves beyond doubt whether the receiver is non-compliant but it also requires significant re-ordering, which harms performance. Therefore it should not be used unless a receiver is already strongly suspected of non-compliance (through failing the first stage).

The technique is designed to work with all known variants of TCP, with or without ECN [RFC3168], with or without SACK [RFC2018], and so on. The technique is probably transferable to derivatives of TCP, such as SCTP [RFC2960], but separate specifications will be required for such related transports. The requirements for a robust solution in Section 4 serve as guidelines for these separate specifications.

## 2. Requirements notation

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

## 3. The Problems

TCP is widely used as the end-to-end transport in the Internet. TCP utilises a number of mechanisms to avoid congestion [RFC5681] in order to avoid the congestion collapses that plagued the Internet in the mid 1980s. These mechanisms all rely on knowing that data has been received (through acknowledgments of that data) and knowing when congestion has happened (either through knowing that a segment was lost in flight or through being notified of an Explicit Congestion Notification (ECN) [RFC3168]). TCP also uses a flow control mechanism to control the rate at which data is sent [RFC0813]. Both the flow control and congestion avoidance mechanisms utilise a transmission window that limits the number of unacknowledged segments that are allowed to be sent at any given time. In order to work out the size of the transmission window, TCP monitors the average round trip time (RTT) for each flow and the number of unacknowledged segments still in flight.

A strategising receiver can take advantage of the congestion and flow control mechanisms to increase its data throughput. The three known ways in which it can do this are: optimistic acknowledgements, concealing segment losses and dividing acknowledgements into smaller parts. The first two are examined in more detail below and details of the third can be found in [Savage].

### 3.1. Concealing Lost Segments

TCP is designed to view a lost segment as an indication of congestion on the channel. This is because TCP makes the reasonable assumption that packets are most likely to be lost through deliberately being dropped by a congested node rather than through transmission losses or errors.

In order to avoid congestion collapse [RFC3714], whichever TCP connection detects the congestion (through detecting that a packet has been dropped or marked) is expected to respond to it either by reducing its congestion window to 1 segment after a timeout or by halving it on receipt of three duplicate acks (the precise rules are set out in [RFC5681]).

For applications where missing data is not an issue, it is in the interest of a receiver to maximise the data rate it gets from the sender. If it conceals lost segments by falsely generating acknowledgements for them it will not suffer a reduction in data rate. There are a number of ways to make an application loss-insensitive. Some applications such as streaming media are inherently insensitive anyway, as a loss will just be seen as a transient error. TCP is widely used to transmit media files, either audio or video, which are relatively insensitive to data loss (depending on the encoding used). Also senders may be serving data containing redundant parity to allow the application to recreate lost data. A misbehaving receiver can also exploit application layer protocols such as the partial GET in HTTP 1.1 [RFC2616] to recover missing data over a secondary connection.

```
     |---.__    Drop            |     |---.__    Drop            |
     |---.__`---#200            |     |---.__`---#200            |
     |      `---.__             |     |      `---.__             |
     |             `---.__      |     |             `---.__      |
     |                  _,`300->|     |                  _,`300->|
     |           __,---'        |     |           __,---'        |
     |     _,---'               |     |     _,---'               |
     |<-100                     |     |<-300                     |
     |---.__                    |     |---.__                    |
     |---.__`---.__             |     |---.__`---.__             |
     |      `---.__`---.__      |     |---.__`---.__`---.__      |
     |             `---.__`400->|   ,-|---.__`---.__`---.__`400->|
     |                  _,`500->|   | |      `---.__`---._,`500->|
     |           __,---'        |   |R|           __~---.__`600->|
     |     _,---'               | - |T|     _,---'       _,`700->|
     |<-100                     | | |T|<-500      __,---'        |
     |---.__                    | | | |     _,---'               |
   ,-|---.__`---.__             | | `-|<-700                     |
   | |      `---.__`---.__      | |<-.
   | |             `---._,`600->| |   \
   |N|           __,---'_,`700->| -    +----------------------+
   |E|     _,---'__,---'        |      | receives segment 700 |
   |W|<-100_,---'               |      | much sooner          |
   | |<-100_                    |      +----------------------+
   |R|---.__`---.__             |
   |T|      `---.__`---.__      |
   |T|             `---._,`200->|
   | |           __,---'  `300->| <-- No ack as duplicate data
   | |     _,---'               |
   `-|<-700                     |
```

Figure 1: Concealing lost segments

## 3.2. Optimistic Acknowledgements

Optimistic acknowledgements were identified as a possible attack in [Savage]. If a receiver is downloading a file from a server, it is probably in its interest to acquire as high a bandwidth as possible for this. One way of increasing the bandwidth is to encourage the sender to believe the round trip time is shorter than it actually is. This means the sender will open up its transmission window faster and thus will send data faster. Of course any lost segments will also be concealed during this attack.

The receiver can achieve this by sending acknowledgements for data it hasn't actually received yet. As long as the acknowledgement is for a packet that has already been transmitted, the sender will assume the RTT has become shorter. This will cause it to increase its transmission window more rapidly and thus send more data. Optimistic acknowledgements are particularly damaging since they can also be used to significantly amplify the effect of a denial of service (DoS) attack on a network. This form of attack is explained in more detail in [Sherwood].

```
  |---.__                    |     |---.__                    |
  |      `---.__             |     |      `---.__             |
  |             `---.__      |     |             `---.__      |
  |                  _,`100->|     |                  _,`100->|
  |           __,---'        |     |           __,---'        |
  |     _,---'               |     |     _,---'               |
  |<-100                     |     |<-100                     |
  |---.__                    |     |---.__                    |
,-|---.__`---.__             |   ,-|---.__`---.__             |
| |      `---.__`---.__      |   |R|      `---.__`---.__      |
|R|             `---.__`200->|   |T|             `---._,`200->|
|T|                  _,`300->|   |T|           __,---'  `300->|
|T|           __,---'        |   | |     _,---'               |
| |     _,---'               |   `-|<-300                     |
`-|<-300                     |     |---.__                    |
  |---.__                    |     |---.__`---.__             |
  |---.__`---.__             |     |---.__`---.__`---.__      |
  |---.__`---.__`---.__      |     |---.__`---.__`---._,`400->|
  |---.__`---.__`---.__`400->|     |      `---.__`---._,`500->|
  |      `---.__`---._,`500->|     |     _,---'__~---.__`600->|
  |           __~---.__`600->|     |<-500_,---'         `700->|
  |     _,---'       _,`700->|     |<-700                     |
  |<-500      __,---'        |
  |     _,---'               |
  |<-700                     |
```

The flow on the left acknowledges data only once it is received. The flow on the right acknowledges data before it is received and consequently the apparent RTT is reduced.

Figure 2: Optimistic acknowledgements

In 2005 US-CERT (the United States Computer Emergency Readiness Team) issued a vulnerability notice [VU102014] specifically addressed to 80 major network equipment manufacturers and vendors who could be affected if someone maliciously exploited optimistic acknowledgements to cause a denial of service. This highlights the potential severity of such an attack were one to be launched. It should be noted however that the primary motivation for using optimistic acknowledgement is likely to be the performance gain it gives rather than the possible negative impact on the network. Application writers may well produce "Download Accelerators" that use optimistic acknowledgements to achieve the performance increase rather than the current parallel connection approach most use. Users of such software would be effectively innocent parties to the potential harm that such a non-compliant TCP could cause.

## 4. Requirements for a robust solution

Since the above problems come about through the inherent behaviour of the TCP protocol, there is no gain in introducing a new protocol as misbehaving receivers can claim to only support the old protocol. The best approach is to provide a mechanism within the existing protocol to test whether a receiver is compliant. The following requirements should be met by any such test in TCP and are likely to be applicable for similar tests in other transport protocols:

1. The compliance test must not adversely affect the existing congestion control and avoidance algorithms since one of the primary aims of any compliance test is to reinforce the integrity of congestion control.

2. Any test should utilise existing features of the TCP protocol. If it can be implemented without altering the existing protocol then implementation and deployment are easier.

3. The receiver should not play an active role in the process. It is much more secure to have a check for compliance that only requires the receiver to behave as it should anyway.

4. It should not require the use of any negotiable TCP options. Since the use of such options is by definition optional, any misbehaving receiver could just choose not to use the appropriate option.

5. If this is a periodic test, the receiver must not be aware that it is being tested for compliance. If a misbehaving receiver can tell that it is being tested (by identifying the pattern of testing) it can choose to respond compliantly only whilst it is being tested. If the test is always performed this clearly doesn’t apply.

6. If the sender actively sanctions any non-compliance it identifies, it should be certain of the receiver’s non-compliance before taking action against it. Any false positives might lead to inefficient use of network resources and could damage end-user confidence in the network.

7. The testing should not significantly reduce the performance of an innocent receiver.

## 5. Existing Proposals

### 5.1. Randomly Skipped Segments

[Sherwood] suggests a simple approach to test a receiver's compliance. The test involves randomly dropping segments at the sender before they are transmitted. All TCP "flavours" require that a receiver should generate duplicate acknowledgements for all subsequent segments until a missing segment is received. This system requires that SACK be enabled so the sender can reliably tell that the duplicate acknowledgements are generated by the segment that is meant to be missing and are not concealing other congestion. Once the first duplicate acknowledgement arrives, the missing segment can then be "re-transmitted". Because this loss has been deliberately introduced, the sender doesn't treat it as a sign of congestion. If a receiver sends an acknowledgement for a segment that was sent after the gap, it proves it is misbehaving or that its TCP is completely non-compliant. It can then be sanctioned. As soon as the first duplicate acknowledgement is received the missing segment is "re-transmitted". This will introduce a 1 RTT delay for some segments which could adversely affect some low-latency applications.

This scheme does work perfectly well in principle and does allow the sender to clearly identify misbehaviour. However it fails to meet requirement 4 in Section 4 above since it requires SACK to be used. If SACK were not used then it would fail to meet requirement 1 as it would be impossible to differentiate between the loss introduced on purpose and any additional loss introduced by the network.

It might be possible to incentivise the use of SACK by receivers by stating that senders are entitled to discriminate against receivers that don't support it. Given that SACK is now widely implemented across the Internet this might be a feasible, but controversial, deployment strategy. However the solution in Section 6 builds on Sherwood's scheme but avoids the need for SACK.

### 5.2. The ECN nonce

The authors of the ECN scheme [RFC3168] identified the failure to echo ECN marks as a potential attack on ECN. The ECN nonce was proposed as a possible solution to this in the experimental [RFC3540]. It uses a 1 bit nonce in every IP header. The nonce works by randomly setting the ECN field to ECT(0) or ECT(1). The sender then maintains the least significant bit of the sum of this value and stores the expected sum for each segment boundary. At the receiver end, the same cumulative 1-bit sum is calculated and is echoed back in the NS (nonce sum) flag added to the TCP header. If a packet has been congestion marked then it loses the information of which ECT codepoint it was carrying. A receiver wishing to conceal the ECN mark will have to guess whether to increment NS or not. Once congestion has been echoed back and the source has started a congestion response the nonce sum in the TCP header is not checked. Once congestion recovery is over the source resets its NS to that of the destination and starts checking again.

On the face of it this solution also fully covers the two problems identified in Section 3. If a receiver conceals a lost segment it has to guess what mark was there and, over several guesses, is very likely to be found out. If a receiver tries to use optimistic acknowledgements it has to guess what nonce was set on all the packets it acknowledges but hasn’t received yet. However there are some key weaknesses to this system. Firstly, it assumes that ECN will be widely deployed (not currently true). Secondly, it relies on the receiver honestly declaring support for both ECN and the ECN nonce - a strategising receiver can simply declare it is neither ECN nor ECN nonce capable and thus avoid the nonce. Thirdly, the mechanism is suspended during any congestion response. Comparing it against the requirements in Section 4 above, it is clear that the ECN nonce fails to meet requirements 3 and 4 and arguably fails to meet requirement 2 as [RFC3540] is experimental. The authors do state that any sender that implements the ECN nonce is entitled to discriminate against any receiver that doesn’t support it. Given there are currently no implementations of the ECN nonce, discriminating against the overwhelming majority of receivers that don’t support it is not a feasible deployment strategy.

### 5.3. A transport layer nonce

One possible solution to the above issues is a multi-bit transport layer nonce. Two versions of this are proposed in [Savage]. The first is the so called "Singular Nonce" where each segment is assigned a unique random number. This value is then echoed back to the receiver with the ack for that segment. The second version is the "Cumulative Nonce" where the nonce is set as before, but the cumulative sum of all nonces is echoed back. Whilst such a system is robust and allows a sender to correctly identify a misbehaving receiver, it has the key drawback that it requires either the creation of a new TCP option to carry the nonce and nonce reply or it requires the TCP header to be extended to include both these fields.

This proposal clearly breaches several of the requirements listed in Section 4. It breaches requirement 2 in that it needs a completely new TCP option or a change to the TCP header. It breaches requirement 3 because it needs the receiver to actively echo the nonce (as does the ECN nonce scheme) and if it uses a TCP option it breaches requirement 4. On the face of it there is no obvious route by which this sort of system can be widely implemented.

# 6. The Test for Receiver Non-compliance

## 6.1. Solution Overview

The ideal solution to the above problems should fully meet the requirements set out in Section 4. The most important of these is that the solution should leverage existing TCP behaviours rather than mandating new behaviours and options. The proposed solution utilises TCP’s receiver behaviour on detecting missing data. To test a receiver the sender delays a segment during transmission by D segments. There is a trade off because increasing D increases the probability of detecting non-compliance but also increases the probability of masking a congestion event during the test. The completely safe strategy for the sender would be to reduce its rate pessimistically as if there were congestion during the test however this will impact the performance of its receivers, thus breaching requirement 7. To overcome this dilemma, the test consists of two stages. In the first stage, the sender uses small displacements without the pessimistic congestion response to determine which receivers appear to be non-compliant. The sender can then prove the non-compliance of these receivers by subjecting them to a deterministic test. This test uses a longer displacement but given the receiver is already under suspicion, it can risk harming performance by pessimistically reducing its rate as if the segment it held back was really lost by the network. The tests can either be implemented as part of a test suite or as a stand-alone modification to the TCP sender implementation. References to the TCP sender in the rest of this document should be taken to include either type of implementation.

## 6.2. Probabilistic Testing

The first requirement for a sender is to decide when to test a receiver. This document doesn’t specify when the test should be performed but the following guidance may be helpful. The simplest option is for a sender to perform the test at frequent random intervals for all its half-connections. There are also some heuristic triggers that might indicate the need for a test. Firstly, if a sender is itself too busy, it would be sensible for it to test all its receivers. Secondly, if the sender has many half-connections that are within a RTT of a congestion response, it would be sensible to test all the half-connections that aren’t in a congestion response. Thirdly, the sender could aim to test all its half-connections at least once. Finally it is to be expected that there is a certain degree of existing segment reordering and thus a sender should be suspicious of any receiver that isn’t generating as many duplicate acknowledgements as other receivers. [Piratla] explores how prevalent reordering might be in the Internet though it is unclear whether the figures given are more widely applicable.

Like the skipped segment solution in Section 5.1, the proposed solution depends on the strict requirement that all TCP receivers have to send a duplicate acknowledgement as soon as they receive an out-of-order segment. This acknowledges that some data has been received, however the acknowledgement is for the last in order segment that was received (hence duplicating an acknowledgment already made). SACK extends this behaviour to allow the sender to infer exactly which segments are missing. This leads to a simple statement: if a receiver is behaving compliantly it must respond to an out-of-order packet by generating a duplicate acknowledgement.

Following from the above statement, a sender can test the compliance of a given receiver by simply delaying transmission of a segment by several places. A compliant receiver will respond to this by generating a number of duplicate acknowledgements. The sender would strongly suspect a receiver of non-compliance if it received no duplicate acknowledgements as a result of the test. A misbehaving receiver can only conceal its actions by waiting until the delayed segment arrives and then generating an appropriate stream of duplicate acknowledgements to appear to be honest. This removes any benefits it may be gaining from cheating because it will significantly increase the RTT observed by the sender.

### 6.2.1. Performing the Probabilistic Test

The actual mechanism for conducting the test is extremely simple. Having decided to conduct a test the sender selects a segment, N. It then chooses a displacement, D (in segments) for this segment where strictly 2 < D < K - 2 where K is the current window size. In practice only low values of D should be chosen to conceal the test among the background reordering and limit the chance of masking congestion. D SHOULD be 6 or less for an initial test. D MUST be greater than 2 to allow for the standard fast retransmit threshold of 3 duplicate acknowledgements. If K is less than 5, the sender should arguably not perform any compliance testing. This is because when the window is so small then non-compliance is not such a significnat issue. The exception to this might be when this test is being used for testing new implementations. To conduct the probabilistic test, instead of transmitting segment N, it transmits N+1, N+2, etc. as shown in the figure below. Once it has transmitted N+D it can transmit segment N. The sender needs to record the sequence number, N as well as the displacement, D.

According to data in [Piratla], as many as 15% of segments in the Internet arrive out of order though this claim may not be accurate. Whatever the actual degree of re-ordering, receivers always expect occasional losses of packets which they cannot distinguish from re-ordering without waiting for the re-ordered packet to arrive. Consequently a misbehaving receiver is unsure how to react to any out-of-order packets it receives.  It should be noted that the natural reordering may reduce the displacement deliberately introduced by the test so the sender should conduct the test more than once.

```
    |--.._                        |
    |--.._‘--.._                  |
    |--.._‘--.._‘--.._            |   +----------------------------+
    |--.._‘--.._‘--.._‘--.._      |   | This figure shows how a    |
    |--.._‘--.._‘--.._‘--.._‘N-1->|   | compliant receiver reacts  |
    |--.._‘--.._‘--.._‘--.._‘N+1->|   | to a probabilistic test    |
    |--.._‘--.._‘--.._‘-=.._‘N+2->|   | with D=4. It sends 4 dup.  |
    |     ‘--.._‘-=.._‘-=.._‘N+3->|   | acknowledgements back to   |
    |      _,--’_-=.._‘-=.._‘N+4->|   | the sender before sending  |
    |<-N-1’_,--’__,--’:-=.._‘-N-->|   | an acknowledgement for N+4 |
    |<-N-1’_,--’__,--’__,--’‘N+5->|   +----------------------------+
    |<-N-1’_,--’__,--’__,--’__,--’|
    |<-N-1’_,--’__,--’__,--’      |
    |<-N-1’_,--’__,--’            |
    |<-N+4’_,--’                  |
    |<-N+5’
```

Figure 3: A receiver reacting honestly to a probabilistic test

During testing, loss of segment L in the range from N+1 to N+D inclusive will be temporarily masked by the duplicate acknowledgements from the intentional gap that was introduced.  In this case the sender’s congestion response will be delayed by at most the offset D.  If there is an actual loss during the test then, once the receiver receives segment N, it will generate an acknowledgement for L-1.  This will lie between N and N+D.  Thus it is reasonable to treat receipt of any acknowledgement between N and N+D inclusive as an indication of congestion and react accordingly.  This will also discourage the receiver from sending optimistic acknowledgements in case these prove to lie in the middle of a testing sequence, in which case it will trigger a congestion response by the sender.  It also means a dishonest receiver has to wait for a full K segments after any genuine lost segment to be sure it isn’t a test as it will otherwise trigger a congestion response.  Delaying by that long will quickly increase the RTT estimate and will soon reduce the transmission rate by as much as if the receiver had reacted honestly to the congestion.

As an additional safety measure, if the sender is performing slow start when it decides to test the receiver, it should change to congestion avoidance. The reason for this is in case there is any congestion that is concealed during the test. If there is congestion, and the sender’s window is still increasing exponentially, this might significantly exacerbate the situation. This does mean that any receiver being tested during this period will suffer reduced throughput, but such testing should only be triggered by the sender being overloaded.

### 6.2.2. Assessing the Probabilistic Test

This approach to testing receiver compliance appears to meet all the requirements set out in Section 4. The most attractive feature is that it enforces equivalence with compliant behaviour. That is to say, a receiver can either honestly report the missing packets or it can suffer a reduced throughput by delaying segments and increasing the RTT. The only significant drawback is that during a test it introduces some delay to the reporting of actual congestion. Given that TCP only reacts once to congestion in each RTT the delay doesn’t significantly adversely affect the overall response to severe congestion.

Some receivers may choose to misbehave despite this. These can be quickly identified by looking at their acknowledgements. A receiver that never sends duplicate acknowledgements in response to being tested is likely to be misbehaving. Equally, a receiver that delays transmission of the duplicate acknowledgements until it is sure it is being tested will leave an obvious pattern of acknowledgements that the sender can identify. Because a receiver is unlikely to be able to differentiate this test from actual re-ordering events, the receiver will be forced to behave in the same fashion for any re-ordered packet even in the absence of a test, making it continually appear to have longer RTT.

### 6.2.3. RTT Measurement Considerations

Clearly, if the sender has re-ordered segment N, it cannot use it to take an accurate RTT measurement. However it is desirable to ensure that, during a test, the sender still measures the RTT of the flow. One of the key aspects of this test is that the only way for an actually dishonest receiver to cheat the test is to delay sending acknowledgements until it is certain a test is happening. If accurate RTTs can be measured during a test, this delay will cause a dishonest receiver to suffer an increase in RTT and thus a reduction in data throughput.

Measurement of the RTT usually depends on receiving an acknowledgement for a segment and measuring the delay between when the segment was sent and when the acknowledgement arrives.  The TCP timestamp option is often used to provide accurate RTT measurement but again, this is not going to function correctly during the test phase.  During a test therefore, the RTT has to be estimated using the arrival of duplicate acknowledgements.  Figure 4 shows how one can measure the RTT in this way, and also demonstrates how this will increase if a dishonest sender chooses to cheat.  However it is not sufficient simply to measure a single RTT during the test.

```
       |‘--._                      |
    ,--|‘--._‘--._                 |  +----------------------------+
   | C |‘--._‘--._‘--._            |  | Segment N is delayed by 3  |
   | h |‘--._‘--._‘--._‘--._       |  | segments. This triggers 3  |
   | e |‘--._‘--._‘--._‘--._‘-N-1->|  | duplicate acknowledgements |
   | c |     ‘--._‘--._‘--._‘-N+1->|  +----------------------------+
   | k |          ‘--._‘--._‘=N+2->|
   |   |               ‘-=._‘=N+3->|  +----------------------------+
   | R |           _,--’_,- ‘=-N=->|  | The RTT can be measured by |
   | T |      _,--’_,--’_,--’_,-’ ,|  | timing the gap between N+1 |
   | T |<-N-1’_,--’_,--’_,--’_,--’ |  | being sent and the 1st     |
    ‘--|<-N-1’_,--’_,--’_,--’      |  | duplicate acknowledgement  |
       |<-N-1’_,--’_,--’           |  | being received.            |
       |<-N-1’_,--’                |  +----------------------------+
       |<-N+3’                     |
       |                           |


       |‘--._                      |
    ,--|‘--._‘--._                 |  +----------------------------+
   | R |‘--._‘--._‘--._            |  | Segment N is delayed by 3  |
   | T |‘--._‘--._‘--._‘--._       |  | segments. The sender has   |
   | T |‘--._‘--._‘--._‘--._‘-N-1->|  | decided to cheat so it has |
   |   |     ‘--._‘--._‘--._‘-N+1->|  | to wait until it gets sent |
   | g |          ‘--._‘--._‘=N+2->|  | segment N.                 |
   | r |                ‘-=.‘-N+3->|  +----------------------------+
   | e |           _,--’    ‘--N-->|
   | a |      _,--’               ,|  +----------------------------+
   | t |<-N-1’               _,--’,|  | Once N arrives it has to   |
   | e |   |            _,--’_,--’,|  | send a couple of duplicate |
   | r |  GAP      _,--’_,--’_,--’,|  | acknowledgements so it     |
   |   |   |  _,--’_,--’_,--’      |  | appears to be honest. This |
    ‘--|<-N-1’_,--’_,--’           |  | will increase the RTT that |
       |<-N-1’_,--’                |  | the sender is measuring.   |
       |<-N+3’                     |  +----------------------------+
       |                           |
```

Figure 4: Measuring the RTT during a test

### 6.2.4. Negative Impacts of the Test

It is important to be aware that keeping track of out-of-order data segments uses some memory resources at the receiver. Clearly this test introduces additional re-ordering to the network and consequently will lead to receivers using additional resources. In order to mitigate against this, any sender that implements the test should only conduct the test at relatively long intervals (of the order of several RTTs).

### 6.2.5. Protocol Details for the Probabilistic Test

- Any TCP sender MAY use the probabilistic test periodically and randomly to check the compliance of its receivers. In particular, it would be advantageous for any sender that is heavily loaded to identify if it is being taken advantage of by non-compliant receivers.

- The decision to test MUST be randomised and MAY be based on: the current load on the sender; whether the receiver is undergoing a congestion response; whether the receiver appears to have different flow characteristics to the others; when the receiver was last tested. The interval between tests SHOULD be relatively long (order of several RTTs).

- To perform the test, the sender selects a segment N. The transmission of this segment will be delayed by D places. D MUST lie between 2 and K-2 exclusively where K is the current size of the transmit window. D SHOULD lie between 3 and 6 inclusively except in those circumstances when a receiver has failed to respond as expected to an earlier test but the sender chooses not to proceed to the deterministic test. D MUST be generated pseudo-randomly and unpredictably. The actual delay SHOULD be such that the receiver can’t distinguish the test segment from the background traffic. If there are less than D segments worth of data in the send buffer then the test SHOULD be omitted.

- If K < 5, the sender SHOULD NOT conduct a compliance test.

- The sequence number N of the delayed segment MUST be recorded by the sender as must the amount of delay D.

- The senders enters the test phase when it transmits segment N+1 instead of N.

- The sender MUST NOT use segment N to measure the RTT of the flow. This is because it won’t get a true acknowledgement for this segment.

- The sender SHOULD use segment N+1 to measure the RTT using the first duplicate acknowledgement it receives to calculate the RTT. This is to ensure that a dishonest receiver will suffer from an increased RTT estimate. The sender SHOULD continue checking the RTT throughout the test period.

- If the sender receives any duplicate acknowledgements during the test phase it MUST check to see if they were generated by the delayed segment (i.e. the acknowledged sequence number must be that of the preceding segment). If they are generated to report the missing segment N the sender SHOULD NOT react as if they are an indication of congestion.

- If the sender receives an acknowledgement for a segment with a sequence number between N and N+D inclusively it MUST treat this as an indication of congestion and react appropriately.

- A sender stops being in the test phase when either it receives the acknowledgement for segment N+D or when it has received at least D duplicate acknowledgments, whichever happens sooner.

- If a sender in the test phase receives D or more duplicate acknowledgements, then it MUST retransmit segment N and react as if there is congestion as specified in [RFC5681]. This is to allow for the possibility that segment N may be lost.

- If the sender is in the slow start phase it MUST move to congestion avoidance as soon as it begins a test. It MAY choose to return to slow start once the test is completed.

- If a sender is in the test phase and receives no duplicate acknowledgements from the receiver it MUST treat this as suspicious and SHOULD perform the more rigorous deterministic test set out in Section 6.3.3.

- If a sender is in the test phase and the next segment to be transmitted has either the FIN or RST bits set, then it must immediately stop the test, and transmit segment N before transmitting the FIN or RST segment.

- A sender MAY choose to monitor the pattern of acknowledgements generated by a receiver. A dishonest receiver is likely to send a distinctive pattern of duplicate acknowledgments during the test phase. As they are unable to detect whether it is a test or not they are also forced to behave the same in the presence of any segment reordering caused by the network.

## 6.3. Deterministic Testing

If after one or more probabilistic tests the sender deems that a receiver is acting suspiciously, the sender can perform a deterministic test similar to the skipped segment scheme in Section 5.1 above.

### 6.3.1. Performing the Deterministic Test

In order to perform the deterministic test the sender again needs to choose a segment, M to use for testing. This time the sender holds back the segment until the receiver indicates that it is missing. Once the receiver sends a duplicate acknowledgement for segment M-1 then the sender transmits segment M. In the meantime data transmission should proceed as usual. If SACK is not in use, this test clearly increases the delay in reporting of genuine segment losses by up to a RTT. This is because it is only once segment M reaches the receiver that it will be able to acknowledge the later loss. Therefore, unless SACK is in use, the sender MUST pessimistically perform a congestion response following the arrival of 3 duplicate acknowledgements for segment M-1 as mandated in [RFC5681].

### 6.3.2. Assessing the Deterministic Test

A dishonest receiver that is concealing segment losses will establish that this isn’t a probabilistic test once the missing segment fails to arrive within the space of 1 congestion window. In order to conceal the loss the receiver will simply carry on acknowledging all subsequent data. The sender can therefore state that if it receives an acknowledgement for a segment with a sequence number greater than M before it has actually sent segment M then the receiver must either be cheating or is very non-compliant.

It is important to be aware that a third party who is able to correctly guess the initial sequence number of a connection might be able to masquerade as a receiver and send acknowledgements on their behalf to make them appear non-compliant or even dishonest. Such an attack can be identified because an honest receiver will also be generating a stream of duplicate acknowledgements until such time as it receives the missing segment.

### 6.3.3. Protocol Details for the Deterministic Test

- If a sender has reason to suspect that a receiver is reacting in a non-compliant manner to the probabilistic test it SHOULD perform the more thorough deterministic test.

- To perform the deterministic test the sender MUST select a segment M at random. The sender MUST store this segment in the buffer of unacknowledged data without sending it and MUST record the sequence number.

- If SACK is not being used, the receiver MUST pessimistically perform a congestion response following the arrival of the first 3 duplicate acknowledgments for segment M-1 as mandated in [RFC5681].

- If the receiver sends an acknowledgement for a segment that was sent after segment M should have been sent, but before segment M is actually sent, then the receiver has proved its non-compliance. The only possible exception to this is if the receiver is also sending a correct stream of duplicate acknowledgements as this implies that a third party is interfering with the connection.

- As soon as the first duplicate acknowledgement for segment M-1 arrives, segment M MUST be transmitted. The effective delay, D, of segment M MUST be calculated and stored.

- If a sender is in the test phase and the next segment to be transmitted has either the FIN or RST bits set, then it must immediately stop the test, and transmit segment N before transmitting the FIN or RST segment.

- Any subsequent acknowledgement for a segment between M and M+D MUST be treated as an indication of congestion and responded to appropriately as specified in [RFC5681].

### 6.4. Responding to Non-Compliance

Having identified that a receiver is actually being dishonest, the appropriate response is to terminate the connection with that receiver. If a sender is under severe attack it might also choose to ignore all subsequent requests to connect by that receiver. However this is a risky strategy as it might give an increased incentive to launch an attack against someone by making them appear to be behaving dishonestly. It is also risky in the current network where many users might share quite a small bank of IP addresses assigned dynamically to them by their ISP’s DHCP server. A safer alternative to blacklisting a given IP address might be to simply test future connections more rigorously.

### 6.5. Possible Interactions With Other TCP Features

In order to be safe to deploy, this test must not cause any unforeseen interactions with other existing TCP features. This section looks at some of the possible interactions that might happen and seeks to show that they are not harmful.

### 6.5.1. TCP Secure

[RFC5961] is a WG Internet Draft that provides a solution to some security issues around the injection of spoofed TCP packets into a TCP connection. The mitigations to these attacks revolve round limiting the acceptable sequence numbers for RST and SYN segments. In order to ensure there is no unforeseen interaction between TCP Secure and this test the test protocol has been specified such that the test will be aborted if a RST segment is sent.

### 6.5.2. Nagle Algorithm

The Nagle algorithm [RFC0896] allows a TCP sender to buffer data waiting to be sent until such time as it receives an acknowledgement for the previous segment. This means that there is only ever one segment in flight and as such this test should not be performed when the Nagle algorithm is being used.

### 6.5.3. Delayed Acknowledgements

[RFC5681] allows for the generation of delayed acknowledgements for data segments. However the tests in this document rely on triggering the generation of duplicate acknowledgements. These must be generated for every out of order packet that is received and should be generated immediately the packet is received. Consequently these mechanisms have no effect on the tests set out in this document.

### 6.5.4. Best Effort Transport Service

The Best Effort Transport Service (BETS) is one operating mode of the Space Communications Protocol Standards (SCPS) [SCPS]. SCPS is a set of communications protocols optimised for extremely high bandwidth-delay product links such as those that exist in space. SCPS-TP (SCPS - Tranpsort Protocol) is based on TCP and is an official TCP option (number 20). The BETS option within SCPS-TP is designed to provide a semi-reliable transport between endpoints. As such it doesn’t necessarily ACK data in the same manner as TCP and thus, if this option has been negotiated on a link the tests described above should not be used.

## 6.6. Possible Issues with the Tests

Earlier in this document we asserted that these tests don’t change the TCP protocol. We make this assertion for two reasons. Firstly the protocol can be implemented as a shim that sits between the TCP and IP layers. Secondly the network and receiver are unable to differentiate between a sender that implements these tests and a sender where the IP layer re-orders packets before transmission.

However the tests might have some impact on the debugging of a TCP implementation. It will also have an impact on debugging traces as it creates additional reordering. The authors feel that these effects are sufficiently minor to be safely ignored. If an author of a new TCP implementation wishes to be certain that they won't be affected by the tests during debugging they simply need to ensure that the sender they are connecting to is not undertaking the tests.

A potentially more problematic consequence is the slight increase in packet reordering that this test might introduce. However the degree of reordering introduced in the probabilistic test is strictly limited. This should have minimal impact on the network as a whole although this assertion would benefit from testing by the wider Internet Community.

The final potential problem is that this test relies on the flows being long-running. However this may not be a real issue since for a short running flow none of the attacks described in Section 3 would give the receiver any advantage in a short flow.

## 7. Comparison of the Different Solutions

The following table shows how all the approaches described in this document compare against the requirements set out in Section 4.

| Requirement | Rand skip segs | ECN nonce | Transp. nonce | Stage 1 test | Stage 2 test |
|---|---|---|---|---|---|
| Congestion Control unaffected | Yes | Yes | Yes | Yes | Yes |
| Utilise existing features | Yes | No** | No | Yes | Yes |
| Receiver passive role | Yes | No | No | Yes | Yes |
| No negotiable TCP options | Yes * | No | No | Yes | Yes |
| Receiver unaware | Yes | N/A | N/A | Yes | Yes |
| Certain of non-compliance | Yes | Yes | Yes | strong suspicion | Yes |
| Innocent rcvr. not adversely affected | No | Yes | Yes | Yes | No |

*  Safer when SACK is used
** Currently Experimental RFC with no known available implementation

Comparing different solutions against the requirements

The table highlights that the three existing schemes looked at in detail in Section 5 all fail on at least two of these requirements. Whilst this doesn’t necessarily make them bad solutions it does mean that they are harder to deploy than the new tests presented in this document. These new tests do have potential issues (see Section 6.6). However, as the table shows, they are minor compared to the problems the nonce-based schemes face, particularly the need for cooperation from the receiver and the use of additional codepoints in the IPv4 and TCP headers.

## 8. Alternative Uses of the Test

Thus far, the two stage test process described in this document has been examined in terms of being a test for compliance by a receiver to the TCP protocol, specifically in terms of the protocol's reaction to segment reordering. The probabilistic test however could also be used for other test purposes. For instance the test can be used to confirm that a receiver has correctly implemented TCP SACK. Because the sender knows exactly which segments have been reordered, it can confirm that the gaps in the data as reported by SACK are indeed correct. The test could also be incorporated as part of a test suite to test the overall compliance of new TCP implementations.

## 9. Evaluating the Experiment

As stated in the introduction, this is an experimental protocol. The main aim of the experiment is to prove that the two tests described in Section 6 provide a robust and safe test for receiver non-compliance. The second aim is to show that the experimental ECN Nonce is no longer needed as these tests provide a more robust defence against receiver non-compliance.

### 9.1. Criteria for Success

The criteria for a successful experiment are very simple.

- o Do the tests accurately identify misbehaving receivers?
- o Are the tests as described in Section 6.2 and Section 6.3 safe? By this we mean is the impact of the test such that it causes no harm to other flows and only minimal harm to honest receivers?

### 9.2. Duration of the Experiment

We believe that the experiment should be proved one way or another within a one year period (subject to volunteers agreeing to help with the evaluation). At the end of the experiment if it is shown to be successful we will go back to the IESG to ask for this test to be moved to standards track. At that point, it would be possible to obsolete the experimental ECN Nonce [RFC3540] and recover the codepoints assigned to it.

### 9.3. Arguments for Obsoleting the ECN Nonce

We believe the tests presented in this document provide significantly greater protection against misbehaving TCP receivers than that provided by the ECN Nonce[RFC3540].

1. The ECN Nonce is acting to block the wider use of the two ECT codepoints defined in ECN [RFC3168]. Currently these have to be treated as having identical meanings except in specific controlled circumstances as mandated in [RFC4774] (PCN [RFC6660] is an example of such a use). The authors are aware of a number of research projects to reduce queuing latency or to speed up slow-start that depend on the availability of the ECT(1) codepoint. If the codepoint were freed up, these projects would gain traction and those with promise could be brought to the IETF. Furthermore the nonce is also holding back a flag in the TCP header (the Nonce Sum or NS flag).

2. The ECN Nonce is an experimental standard intended to allow a sender to test whether ECN CE markings (or losses) are being suppressed by the receiver (or anywhere else in the feedback loop, such as another network or a middlebox). In the 11 years since it was presented there has been no evidence of any deployment. To the best of our knowledge only two implementations have ever existed. One was that of the original authors and the other was written to test an alternative use of the nonce [Spurious]. Furthermore the nonce would now be nearly impossible to deploy retrospectively, because to catch a misbehaving receiver it relies on the receiver volunteering feedback information to incriminate itself. A receiver that has been modified to misbehave can simply claim that it does not support nonce feedback, which will seem unremarkable given so many other hosts do not support it either.

3. As explained in Section 7, the ECN Nonce is only a limited solution to the security implications of failing to provide accurate congestion feedback. However some authors may not realise its limitations and may choose to argue that its existence offers them sufficient protection from misbehaving receivers.

## 10. IANA Considerations

This memo includes no request to IANA.

## 11. Security Considerations

The two tests described in this document provide a solution to two of the significant security problems that were outlined in [Savage]. Both these attacks could potentially cause major congestion of senders own resources (by making them transmit at too high a rate) and could lead to network congestion collapse through subverting the correct reporting of congestion or by amplifying any DoS attack [Sherwood]. The proposed solution cannot alone prevent misbehaving senders from causing congestion collapse of the Internet. However, the more widely it is deployed by trustworthy senders, the more these particular attacks would be mitigated through ensuring accurate reporting of segment losses. The more senders that deploy these measures, the less likely it is that a misbehaving receiver will be able to find a sender to fool into causing congestion collapse.

It should be noted that if a third party is able to correctly guess the initial sequence number of a connection, they might be able to masquerade as a receiver and send acknowledgements on their behalf to make them appear dishonest during a deterministic test.

Due to the wording of [RFC5681] a receiver wishing to establish whether a probabilistic test is happening can keep their acknowledgement clock running (thus maintaining transmission rate) by generating pairs of duplicate acknowledgements for segments it received prior to the gap in the data stream caused by the test. This would allow a receiver to subsequently send any additional duplicate acknowledgements that would be necessary to make it appear honest. Such behaviour by a receiver would be readily apparent by examining the pattern of the acknowledgements. Should receivers prove able to exploit this to their advantage, there might be a need to change some of the musts and shoulds laid out in Section 6.2.5.

[Savage] also identified a further attack involving splitting acknowledgements into smaller parts. TCP is designed such that increases in the congestion window are driven by the arrival of a valid acknowledgement. It doesn’t matter if this acknowledgement covers all of a transmitted segment or not. This means a receiver that divides all its acknowledgements into two will cause the congestion window to open at twice the rate it would do otherwise. The tests described above can’t protect against that attack. However there is a straightforward solution to this - every time the sender transmits a new segment it increments a counter; every acknowledgment it receives decrements that counter; if the counter reaches zero, the sender won’t increase its congestion window in response to a new acknowledgement arriving. To comply with this document, senders MUST implement a solution to this problem.

## 12. Conclusions

The issue of mutual trust between TCP senders and receivers is a significant one in the current Internet. This document has introduced a mechanism by which senders can verify that their receivers are compliant with the current TCP protocol. The whole process is robust, lightweight, elegant and efficient. The probabilistic test might delay a congestion notification by a fraction of a RTT, however this is compensated for by the protocol reacting more rapidly to any such indication. The deterministic test carries a greater risk of delaying congestion notification and consequently the protocol mandates that a congestion response should happen whilst performing the test. The two tests combine to provide a mechanism to allow the sender to judge the compliance of a receiver in a manner that both encourages compliant behaviour and proves non-compliance in a robust manner. The most attractive feature of this scheme is that it requires no active participation by the receiver as it utilises the standard behaviour of TCP in the presence of missing data. The only changes required are at the sender.

As mentioned in the introduction, the tests described in this document aren’t intended to become a necessary feature for compliant TCP stacks. Rather, the intention is to provide a safe testing mechanism that a sender could choose to implement were it to decide there is a need. If optimistic acknowledgements do start to become widely exploited the authors of this draft feel it would be valuable to have an IETF-approved test that can be used to identify non-compliant receivers. In the mean-time these tests can be used for a number of alternative purposes such as testing that a new receiver stack is indeed compliant with the protocol and testing if a receiver has correctly implemented SACK.

In the longer term it would be hoped that the TCP protocol could be modified to make it robust against such non-compliant behaviour, possibly through the incorporation of a cumulative transport layer nonce as described in Section 5.3.

## 13. Acknowledgements

The authors would like to acknowledge the assistance and comments they received from contributors to the TCPM mailing list. In particular we would like to thank Mark Allman, Caitlin Bestler, Lars Eggert, Gorry Fairhurst, John Heffner, Alfred Hoenes, David Mallone, Gavin McCullagh, Anantha Ramaiah, Rob Sherwood, Joe Touch and Michael Welzl.

Bob Briscoe was part-funded by the European Community under its Seventh Framework Programme through the Reducing Internet Transport Latency (RITE) project (ICT-317700). The views expressed here are solely those of the authors.

## 14. Comments Solicited

Comments and questions are encouraged and very welcome. They can be addressed to the IETF TCP Maintenance and Minor Extensions working group mailing list <tcpm@ietf.org>, and/or to the authors.

# 15. References

## 15.1. Normative References

[RFC0793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, September 1981.

[RFC0813] Clark, D., "Window and Acknowledgement Strategy in TCP", RFC 813, July 1982.

[RFC2018] Mathis, M., Mahdavi, J., Floyd, S., and A. Romanow, "TCP Selective Acknowledgment Options", RFC 2018, October 1996.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC5681] Allman, M., Paxson, V., and E. Blanton, "TCP Congestion Control", RFC 5681, September 2009.

[RFC5961] Ramaiah, A., Stewart, R., and M. Dalal, "Improving TCP’s Robustness to Blind In-Window Attacks", RFC 5961, August 2010.

## 15.2. Informative References

[Piratla] Piratla, N., Jayasumana, A., and T. Banka, "On reorder density and its application to characterization of packet reordering", IEEE Conference on Local Computer Networks 2005, 2005.

[RFC0896] Nagle, J., "Congestion control in IP/TCP internetworks", RFC 896, January 1984.

[RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.

[RFC2960] Stewart, R., Xie, Q., Morneault, K., Sharp, C., Schwarzbauer, H., Taylor, T., Rytina, I., Kalla, M., Zhang, L., and V. Paxson, "Stream Control Transmission Protocol", RFC 2960, October 2000.

[RFC3168] Ramakrishnan, K., Floyd, S., and D. Black, "The Addition of Explicit Congestion Notification (ECN) to IP", RFC 3168, September 2001.

[RFC3540] Spring, N., Wetherall, D., and D. Ely, "Robust Explicit Congestion Notification (ECN) Signaling with Nonces", RFC 3540, June 2003.

[RFC3714] Floyd, S. and J. Kempf, "IAB Concerns Regarding Congestion Control for Voice Traffic in the Internet", RFC 3714, March 2004.

[RFC4774] Floyd, S., "Specifying Alternate Semantics for the Explicit Congestion Notification (ECN) Field", BCP 124, RFC 4774, November 2006.

[RFC6660] Briscoe, B., Moncaster, T., and M. Menth, "Encoding Three Pre-Congestion Notification (PCN) States in the IP Header Using a Single Diffserv Codepoint (DSCP)", RFC 6660, July 2012.

[SCPS] Consultative Committee for Space Data Systems, "Space Control Protocol Specification - Transport Protocol", CCSDS Recommended Standard CCSDS 714.0-B-2, 2006.

[Savage] Savage, S., Cardwell, N., Wetherall, D., and T. Anderson, "TCP congestion control with a misbehaving receiver", ACM SIGCOMM Computer Communications Review Vol.29/5, 1999.

[Sherwood] Sherwood, R., Bhattacharjee, B., and R. Braud, "Misbehaving TCP receivers can cause Internet-wide congestion collapse", Proceedings of the 12th ACM conference on Computer and communications security 2005, 2005.

[Spurious] Welzl, M., "Using the ecn nonce to detect spurious loss events in TCP", IEEE Global Telecommunications Conference 2008, 2008.

[VU102014] US Cert, "Optimistic TCP acknowledgements can cause denial of service", Vulnerablility Note 102014, 2005.

## Appendix A. Changes from previous drafts (to be removed by the RFC Editor)

From -02 to -03:

Draft revived after 6 year hiatus. Status changed to experimental. The primary am of the experiment is to show that these tests correctly and safely identify misconfigured or misbehaving TCP receivers. The secondary aim is to demonstrate that the ECN Nonce is not needed and hence show that that experiment has failed. Minor changes made to tighten the text.

From -01 to -02:

A number of changes made following an extensive review from Alfred Hoenes. These were largely to better comply with the stated aims of the previous version but also included some tidying up of the protocol details and a new section on a possible unwanted interaction.

From -00 to -01:

Draft rewritten to emphasise testing for non-compliance. Some changes to protocol to remove possible unwanted interactions with other TCP variants. Sections added on comparison of solutions and alternative uses of test.

## Authors' Addresses

Toby Moncaster (editor)
University of Cambridge
Computer Laboratory
J.J. Thomson Avenue
Cambridge  CB3 0FD
UK

Phone: +44 1223 763654
Email: toby.moncaster@cl.cam.ac.uk

Bob Briscoe
BT
B54/77, Adastral Park
Martlesham Heath
Ipswich  IP5 3RE
UK

Phone: +44 1473 645196
Email: bob.briscoe@bt.com

Arnaud Jacquet
BT
B54/70, Adastral Park
Martlesham Heath
Ipswich  IP5 3RE
UK

Phone: +44 1473 647284
Email: arnaud.jacquet@bt.com