Network Working Group D. Newman
Internet-Draft Network Test
Expires: April 11, 2006 T. Player
Spirent Communications
October 8, 2005

Hash and Stuffing: Overlooked Factors in Network Device Benchmarking
draft-ietf-bmwg-hash-stuffing-04.txt

Status of this Memo

Copyright Notice

Abstract

Test engineers take pains to declare all factors that affect a given measurement, including offered load, packet length, test duration, and traffic orientation. However, current benchmarking practice overlooks two factors that have a profound impact on test results. First, existing methodologies do not require the reporting of addresses or other test traffic contents, even though these fields can affect test results. Second, "stuff" bits and bytes inserted in

test traffic by some link-layer technologies add significant and variable overhead, which in turn affects test results. This document describes the effects of these factors; recommends guidelines for test traffic contents; and offers formulas for determining the probability of bit- and byte-stuffing in test traffic.

Table of Contents

1. Introduction

Experience in benchmarking networking devices suggests that the contents of test traffic can have a profound impact on test results. For example, some devices may forward randomly addressed traffic without loss, but drop significant numbers of packets when offered packets containing nonrandom addresses.

Methodologies such as [RFC2544] and [RFC2889] do not require any declaration of packet contents. These methodologies do require the declaration of test parameters such as traffic distribution and traffic orientation, and yet packet contents can have at least as great an impact on test results as the other factors. Variations in packet contents also can lead to non-repeatability of test results: Two individuals may follow methodology procedures to the letter, and still obtain very different results.

A related issue is the insertion of stuff bits or bytes by link-layer technologies using PPP with HDLC-like framing. This stuffing is done to ensure sequences in test traffic will not be confused with flag or control characters.

Stuffing adds significant and variable overhead. Currently there is no standard method for determining the probability that stuffing will occur for a given pattern, and thus no way to determine what impact stuffing will have on test results.

This document covers two areas. First, we discuss strategies for dealing with randomness and nonrandomness in test traffic. Second, we present formulas to determine the probability of bit- and byte-stuffing on PPP and POS circuits. In both areas, we provide recommendations for obtaining more repeatability in test results.

2. Requirements

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. General considerations

3.1. Repeatability

Repeatability is a desirable trait in benchmarking, but it can be an elusive goal. It is a common but mistaken belief that test results can always be reproduced provided the device under test and test instrument are configured identically for each test iteration. In fact, even identical configurations may introduce some variations in test traffic, such as changes in timestamps, TCP sequence numbers, or other naturally occurring phenomena.

While this variability does not necessarily invalidate test results, it is important to recognize such variation exists. Exact bit-for-bit reproduction of test traffic in all cases is a hard problem. A simpler approach is to acknowledge that some variation exists, characterize that variation, and describe it when analyzing test results.

3.2. Randomness

This document recommends the use of pseudorandom patterns in test traffic under controlled lab conditions. The rand() functions available in many programming languages produce output that is pseudorandom rather than truly random. Pseudorandom patterns are sufficient for the recommendations given in this document, provided they produce output that is uniformly distributed across the pattern space.

Specifically, for any random bit pattern of length L, the probability of generating that specific pattern SHOULD equal 1 over 2 to the Lth power.

4. Address Pattern Variations

4.1. Problem Statement

The addresses and port numbers used in a test can have a significant impact on metrics such as throughput, jitter, latency, and loss. This is because many network devices feed such addresses into hashing algorithms to determine which path upon which to forward a given packet.

Consider the simple example of an Ethernet switch with eight network processors (NPs) in its switching fabric:

```
                         ingress
                            ||
                            \/
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
      | ___   ___   ___   ___   ___   ___   ___   ___  |
      ||   | |   | |   | |   | |   | |   | |   | |   | |
      ||NP0| |NP1| |NP2| |NP3| |NP4| |NP5| |NP6| |NP7| |
      ||___| |___| |___| |___| |___| |___| |___| |___| |
      |                                                |
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
                            ||
                            \/
                          egress
```

To assign incoming traffic to the various NPs, suppose a hashing algorithm performs an exclusive-or (XOR) operation on the least significant 3 bits of the source and destination MAC addresses in each frame. (This is an actual example the authors have observed in multiple devices from multiple manufacturers.)

In theory, a random distribution of source and destination MAC addresses should result in traffic being uniformly distributed across all eight NPs. (Instances of the term "random" in this document refer to a random uniform distribution across a given address space. Section 3.2 describes random uniform distributions in more detail.) In practice, the actual outcome of the hash (and thus any test results) will be very different depending on the degree of randomness in test traffic.

Suppose the traffic is nonrandom so that every interface of the test instrument uses this pattern in its source MAC addresses:

00:00:PP:00:00:01

where PP is the source interface number of the test instrument.

In this case, the least significant 3 bits of every source and destination MAC address are 001, regardless of interface number. Thus, the outcome of the XOR operation will always be 0, given the same three least significant bits:

001 ^ 001 = 000

Thus, the switch will assign all traffic to NP0, leaving the other seven NPs idle. Given a heavy enough load, NP0 and the switch will become congested, even though seven other NPs are available. At most, this device will be able to utilize approximately 12.5 percent of its total capacity, with the remaining 87.5 percent of capacity unused.

Now consider the same example with randomly distributed addresses. In this case, the test instrument offers traffic using MAC addresses with this pattern:

00:00:PP:00:00:RR

where PP is the source interface number of the test instrument and RR is a pseudorandom number. In this case, there should be an equal probability of the least significant 3 bits of the MAC address having any value from 000 to 111 inclusive. Thus, the outcome of XOR operations should be equally distributed from 0 to 7, and distribution across NPs should also be equal (at least for this particular 3-bit hashing algorithm). Absent other impediments, the device should be able to utilize 100 percent of available bandwidth.

This simple example presumes knowledge on the tester's part of the hashing algorithm used by the device under test. Knowledge of such algorithms is not always possible beforehand, and in any event violates the "black box" spirit of many documents produced by the IETF BMWG.

The balance of this section offers recommendations for test traffic patterns, starting at the link layer and working up to the transport layer. These patterns should overcome the effects of nonrandomness regardless of the hashing algorithms in use.

4.2. Ethernet MAC Addresses

Test traffic SHOULD use pseudorandom patterns in Ethernet addresses. The following source and destination Ethernet address pattern is RECOMMENDED for use when benchmarking Ethernet devices:

(RR & 0xFE):PP:PP:RR:RR:RR

where (RR & 0xFE) is a pseudorandom number bitwise ANDed with 0xFE, PP:PP is the 1-indexed interface number of the test instrument and RR:RR:RR is a pseudorandom number.

The bitwise ANDing of the high-order byte in the MAC address with 0xFE guarantees a non multicast address.

Test traffic SHOULD use PP:PP to identify the source interface number of the test instrument. Such identification can be useful in troubleshooting. Allocating 2 bytes of the MAC address for interface identification allows for tests of up to 65,536 interfaces. A 2-byte space allows for tests much larger than those currently used in device benchmarking; however, tests involving more than 256 interfaces (fully utilizing a 1-byte space) are fairly common.

Further, source interface numbers SHOULD be 1-indexed and SHOULD NOT be 0-indexed. This avoids the low but nonzero probability of an all-0s Ethernet address. Some devices will drop frames with all-0s Ethernet addresses.

It is RECOMMENDED to use pseudorandom patterns in the least significant 3 bytes of the MAC address. Using pseudorandom values for the low-order 3 bytes means choosing one of 16.7 million unique addresses. While this address space is vastly larger than is currently required in lab benchmarking, it does assure more realistic test traffic.

Note also that since only 31 of 48 bits in the MAC address have pseudorandom values, there is no possibility of randomly generating a broadcast or multicast value by accident.

4.2.1. Randomized Sets of MAC Addresses

It is common benchmarking practice for a test instrument to emulate multiple hosts, even on a single interface. This is desirable in assessing DUT/SUT scalability.

However, test instruments may emulate multiple MAC addresses by incrementing and/or decrementing addresses from a fixed starting point. This leads to situations as described above in "Address Pattern Variations" where hashing algorithms produce nonoptimal outcomes.

The outcome can be nonoptimal even if the set of addresses begins with a pseudorandom number. For example, the following source/destination pairs will not be equally distributed by the 3-bit hashing algorithm discussed above:

Source	Destination
00:00:01:FC:B3:45	00:00:19:38:8C:80
00:00:01:FC:B3:46	00:00:19:38:8C:81
00:00:01:FC:B3:47	00:00:19:38:8C:82
00:00:01:FC:B3:48	00:00:19:38:8C:83
00:00:01:FC:B3:49	00:00:19:38:8C:84
00:00:01:FC:B3:4A	00:00:19:38:8C:85
00:00:01:FC:B3:4B	00:00:19:38:8C:86
00:00:01:FC:B3:4C	00:00:19:38:8C:87

Again working with our 3-bit XOR hashing algorithm, we get the following outcomes:

```
101 ^ 000 = 101
110 ^ 001 = 111
111 ^ 010 = 101
000 ^ 011 = 011
001 ^ 100 = 101
010 ^ 101 = 111
011 ^ 110 = 101
100 ^ 111 = 011
```

Note that only three of eight possible outcomes are achieved when incrementing addresses. This is actually the best case. Incrementing from other combinations of pseudorandom address pairs produces only one or two out of eight possible outcomes.

Every MAC address SHOULD be pseudorandom, not just the starting one.

When generating traffic with multiple addresses, it is RECOMMENDED that all addresses use pseudorandom values. There are multiple ways to use sets of pseudorandom numbers. One strategy would be for the test instrument to iterate over an array of pseudorandom values rather than incrementing/decrementing from a starting address. The actual method is an implementation detail; in the end, any method that uses multiple addresses and avoids hash table collisions will be sufficient.

4.3. MPLS Addressing

Similiar to L2 switches, MPLS routers make forwarding decisions based on a 20 bit MPLS label. Unless specific labels are required, it is RECOMMENDED that uniformly random values between 0 and 1,048,575 be used for all labels assigned by test equipment.

4.4. Network-layer Addressing

Routers make forwarding decisions based on destination network address. Since there is no hashing of source and destination addresses, the requirement for pseudorandom patterns at the network layer is far less critical than in the Ethernet MAC address case.

However, there are cases where randomly distributed IPv4 and/or IPv6 addresses are desirable. For example, the equal cost multipath (ECMP) feature performs load-sharing across multiple links. Routers implementing ECMP may perform a hash of source and destination IP addresses in assigning flows.

Since multiple ECMP routes by definition have the same metric, routers use some other "tiebreaker" mechanism to assign traffic to each link. As far as the authors are aware, there is no standard algorithm for ECMP link assignment. Some implementations perform a hash of all bits of the source and destination IP addresses for this purpose.

Just as in the case of MAC addresses, nonrandom IP addresses can have an adverse effect on the outcome of ECMP link assignment decisions.

When benchmarking devices that implement ECMP or any other form of Layer 3 aggregation, it is RECOMMENDED to use a randomly distributed range of IP addresses.

4.5. Transport-layer Addressing

Some devices with transport- or application-layer awareness use TCP or UDP port numbers in making forwarding decisions. Examples of such devices include load balancers and application-layer firewalls.

Test instruments have the capability of generating packets with random TCP and UDP source and destination port numbers. Known destination port numbers are often required for testing application-layer devices. However, unless known port numbers are specifically required for a test, it is RECOMMENDED to use randomly distributed values for both source and destination port numbers.

In addition, it may be desirable to pick pseudorandom values from a

selected pool of numbers. Many services identify themselves through use of reserved destination port numbers between 1 and 1023 inclusive. Unless specific port numbers are required, it is RECOMMENDED to pick randomly distributed destination port numbers between these lower and upper boundaries.

Similarly, clients typically choose source port numbers in the space between 1024 and 65535 inclusive. Unless specific port numbers are required, it is RECOMMENDED to pick randomly distributed source port numbers between these lower and upper boundaries.

5. Control Character Stuffing

5.1. Problem Statement

Link-layer technologies that use HDLC-like framing may insert an extra bit or byte before each instance of a control character in traffic. These insertions prevent confusion with control characters, but they may also introduce significant overhead.

The overhead of these escape sequences is problematic for two reasons. First, the amount of overhead is non-deterministic. The best testers can do is to characterize the probability that an escape sequence will occur for a given pattern. This greatly complicates the requirement of declaring exactly how much traffic is offered to a DUT/SUT.

Second, in the absence of characterization and compensation for this overhead, the tester may unwittingly congest the DUT/SUT. For example, if a tester intends to offer traffic to a DUT at 95 percent of line rate, but the link-layer protocol introduces an additional 1 percent of overhead to escape control characters, then the aggregate offered load will be 96 percent of line rate. If the DUT's actual channel capacity is only 95 percent, congestion will occur and the DUT will drop traffic even though the tester did not intend this outcome.

As described in [RFC1661] and [RFC1662], PPP and HDLC-like framing introduce two kinds of escape sequences: bit and byte stuffing. Bit stuffing refers to the insertion of an escape bit on bit-synchronous links. Byte stuffing refers to the insertion of an escape byte on byte-synchronous links. We discuss each in turn.

5.2. PPP Bit Stuffing

[RFC1662], section 5.2 specifies that any sequence of five contiguous "1" bits within a frame must be escaped by inserting a "0" bit prior to the sequence. This escaping is necessary to avoid confusion with the HDLC control character 0x7D, which contains six "1" bits.

Consider the following PPP frame containing a TCP/IP packet. Not shown is the 1-byte flag sequence (0x7D), at least one of which must occur between frames.

The contents of the various frame fields can be described one of two ways:

1. Field contents never change over the test duration. An example would be the IP version number.

2. Field contents change over the test duration. Some of these changes are known prior to the test duration. An example would be the use of incrementing IP addresses. Some of these changes are unknown. An example would be a dynamically calculated field such as the TCP checksum.

In the diagram below, 30 out of 48 total bytes are subject to change over the test duration. The fields containing the changeable bytes are given in ((double parentheses)).

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|    Address    |    Control    |           Protocol            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|Version|  IHL  |Type of Service|          Total Length         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|         Identification        |Flags|      Fragment Offset    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Time to Live |    Protocol   |       ((Header Checksum))     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                      ((Source Address))                       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                   ((Destination Address))                     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|        ((Source Port))        |     ((Destination Port))      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       ((Sequence Number))                     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                   ((Acknowledgment Number))                   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Data |           |U|A|P|R|S|F|                               |
| Offset| Reserved  |R|C|S|S|Y|I|          ((Window))           |
|       |           |G|K|H|T|N|N|                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|         ((Checksum))          |         Urgent Pointer        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        ((FCS (4 bytes) ))                     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

None of the other fields are known to contain sequences subject to bit-stuffing, at least not in their entirety.

Given the information at hand, and assuming static contents for the rest of the fields, the challenge is to determine the probability that bit-stuffing will occur.

5.2.1. Calculating Bit-Stuffing Probability

In order to calculate bit-stuffing probabilities, we assume that for any string of length L, the probability of the Lth + 1 bit equalling 1 is 0.5 and the probability of the Lth + 1 bit equalling 0 is 0.5. Additionally, the value of the Lth + 1 bit is independant of any previous bits.

We can calculate the probability of bit stuffing for both infinite and finite strings of random bits. We begin with the infinite-string case, which is required to prove the finite-string case. For an infinitely long string of random bits, we will need to insert a stuff bit if and only if state 5 is reached in the following state table.

```
                   |--------------------<----------------------|
                   |                                           |1
    _______      __|__      _____      _____      _____      __|__
   |       | 1  |     | 1  |     | 1  |     | 1  |     | 1  |     |
   | start |--->|  1  |--->|  2  |--->|  3  |--->|  4  |--->|  5  |
   |_______|    |_____|    |_____|    |_____|    |_____|    |_____|
     |   |         |          |          |          |          |
     |   |0        |0         |0         |0         |0         |0
     |-<-|----<----|----<-----|----<-----|----<-----|----<-----|
```

Initially, we begin in the "start" state. A 1 bit moves us into the next highest state, and a 0 bit returns us to the start state. From state 5, a 1 bit takes us back to the 1 state and a 0 bit returns us to "start." From this state table we can build the following transition matrix:

	start	1	2	3	4	5
start	0.5	0.5	0.5	0.5	0.5	0.5
1	0.5	0.0	0.0	0.0	0.0	0.5
2	0.0	0.5	0.0	0.0	0.0	0.0
3	0.0	0.0	0.5	0.0	0.0	0.0
4	0.0	0.0	0.0	0.5	0.0	0.0
5	0.0	0.0	0.0	0.0	0.5	0.0

With this transition matrix we can build the following system of

equations. If P(x) represents the probability of reaching state x, then:

P(start) = 0.5 * P(start) + 0.5 * P(1) + 0.5 * P(2) + 0.5 * P(3) + 0.5 * P(4) + 0.5 * P(5)

P(1) = 0.5 * P(start) + 0.5 * P(5)
P(2) = 0.5 * P(1)
P(3) = 0.5 * P(2)
P(4) = 0.5 * P(3)
P(5) = 0.5 * P(4)

P(start) + P(1) + P(2) + P(3) + P(4) + P(5) = 1

Solving this system of equations yields:

P(start) = 0.5
P(1) = 8/31
P(2) = 4/31
P(3) = 2/31
P(4) = 1/31
P(5) = 1/62

Thus, for an infinitely long string of random bits, the probability of 5 sequential 1 bits is 1/62. Put another way, we expect to add one stuff bit for every 62 bits of random uniform data.

5.2.2. Bit Stuffing for Finite Strings

The above result indicates that for any string of uniformly distributed random bits, we expect a stuffing event to occur every 62 bits. So, given a string of some finite length L, where L >= 5, the expected number of stuffs is simply L * 1/62.

5.2.3. Applied Bit Stuffing

The amount of overhead attributable to bit stuffing may be calculated explicitly as long as the total number of random bits per frame, L_rand-bits, and the probability of stuffing, P(stuff), is known.

% overhead = (P(stuff) * L_rand-bits) / framesize (in bits)

Note that if the entire frame contains random bits, then the percentage overhead is simply the probability of stuffing expressed

as a percentage.

Given that the overhead added by bit-stuffing is at most 1 in 62, or approximately 1.6 percent, it is RECOMMENDED that testers reduce the maximum offered load by 1.6 percent to avoid introducing congestion when testing devices using bit-synchronous interfaces (such as T1/E1, DS-3, and the like).

The percentage given above is an approximation. For greatest precision, the actual offered load SHOULD be calculated using the percentage overhead formula above and then expressed in frames per second, rounded down to the nearest integer.

Note that the DUT/SUT may be able to forward offered loads higher than the calculated theoretical maximum rate without packet loss. Such results are the result of queuing on the part of the DUT/SUT. While a device's throughput may be above this level, delay-related measurements may be affected. Accordingly, it is RECOMMENDED to reduce offered levels by the amount of bit-stuffing overhead when testing devices using bit-synchronous links. This recommendation applies for all measurements, including throughput.

5.3. POS Byte Stuffing

[RFC1662] requires that "Each Flag Sequence, Control Escape octet, and any octet which is flagged in the sending Async-Control-Character-Map (ACCM), is replaced by a two octet sequence consisting of the Control Escape octet followed by the original octet exclusive-or'd with hexadecimal 0x20." The practical effect of this is to insert a stuff byte for instances of up to 34 characters: 0x7E, 0x7D, or any of 32 ACCM values.

A common implementation of PPP in HDLC-like framing is in PPP over Sonet/SDH (POS), as defined in [RFC2615].

As with the bit-stuffing case, the requirement in characterizing POS test traffic is to determine the probability that byte-stuffing will occur for a given sequence. This is much simpler to do than with bit-synchronous links, since there is no possibility of overlap across byte boundaries.

5.3.1. Nullifying ACCM

Testers can greatly reduce the probability of byte-stuffing by configuring link partners to negotiate an ACCM value of 0x00. It is RECOMMENDED that testers configure the test instrument(s) and DUT/SUT to negotiate an ACCM value of 0x00 unless specific ACCM values are required.

One instance where nonzero ACCM values are used is in the layer 2 tunneling protocol (L2TP), as defined in [RFC2661], section 4.4.6. When the default ACCM values are used, the probability of stuffing for any given random byte is 34 in 256, or approximately 13.3 percent.

5.3.2. Other Stuffed Characters

If an ACCM value of 0x00 is negotiated, the only characters subject to stuffing are the flag and control escape characters. Thus, we can say that without ACCM the probability of stuffing for any given random byte is 2 in 256, or approximately 0.8 percent.

5.3.3. Applied Byte Stuffing

The amount of overhead attributable to bit or byte stuffing may be calculated explicitly as long as the total number of random bytes per frame, L_rand-bytes, and the probability of stuffing, P(stuff), is known.

% overhead = (P(stuff) * L_rand-bytes) / framesize (in bytes)

Note that if the entire frame contains random bytes, then the percentage overhead is simply the probability of stuffing expressed as a percentage.

When testing a DUT/SUT that implements PPP in HDLC-like framing and L2TP (or any other technology that uses nonzero ACCM values), it is RECOMMENDED that testers reduce the maximum offered load by 13.3 percent to avoid introducing congestion.

When testing a DUT/SUT that implements PPP in HDLC-like framing and an ACCM value of 0x00, it is RECOMMENDED that testers reduce the maximum offered load by 0.8 percent to avoid introducing congestion.

Note that the percentages given above are approximations. For greatest precision, the actual offered load SHOULD be calculated using the percentage overhead formula above and then expressed in frames per second (rounded down to the nearest integer).

Note also that the DUT/SUT may be able to forward offered loads higher than the calculated theoretical maximum rate without packet loss. Such results are the result of queuing on the part of the DUT/SUT. While a device's throughput may be above this level, delay-related measurements may be affected. Accordingly, it is RECOMMENDED to reduce offered levels by the amount of byte-stuffing overhead when testing devices using byte-synchronous links. This recommendation applies for all measurements, including throughput.

6. Security Considerations

This document recommends the use of pseudorandom patterns in test traffic. The rand() functions of many programming languages produce output that is pseudorandom rather than truly random. As far as the authors are aware, pseudorandom patterns are sufficient for generating test traffic in lab conditions.

[RFC2615], section 6, discusses a denial-of-service attack involving the intentional transmission of characters that require stuffing. This attack could consume up to 100 percent of available bandwidth. However, the test networks described in BMWG documents generally SHOULD NOT be reachable by anyone other than the tester(s).

7. IANA Considerations

This document has no actions for IANA.

8. References

8.1. Normative References

[RFC1661] Simpson, W., "The Point-to-Point Protocol (PPP)", STD 51, RFC 1661, July 1994.

[RFC1662] Simpson, W., "PPP in HDLC-like Framing", STD 51, RFC 1662, July 1994.

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

[RFC2544] Bradner, S. and J. McQuaid, "Benchmarking Methodology for Network Interconnect Devices", RFC 2544, March 1999.

[RFC2615] Malis, A. and W. Simpson, "PPP over SONET/SDH", RFC 2615, June 1999.

[RFC2661] Townsley, W., Valencia, A., Rubens, A., Pall, G., Zorn, G., and B. Palter, "Layer Two Tunneling Protocol "L2TP"", RFC 2661, August 1999.

[RFC2889] Mandeville, R. and J. Perser, "Benchmarking Methodology for LAN Switching Devices", RFC 2889, August 2000.

8.2. Informative References

[Ca63] Campbell, D. and J. Stanley, "Experimental and Quasi-Experimental Designs for Research", 1963.

[Go97] Goralski, W., "SONET: A Guide to Synchronous Optical Networks", 1997.

[Kn97] Knuth, D., "The Art of Computer Programming, Volume 2, Third Edition", 1997.

Appendix A. Acknowledgements

The authors gratefully acknowledge reviews and contributions by Neil Carter, Glenn Chagnot, Rafael Francis, Paul Hoffman, David Joyner, Joe Perches, and Scott Poretsky.

Authors' Addresses

David Newman
Network Test

Email: dnewman@networktest.com

Timmons C. Player
Spirent Communications

Email: timmons.player@spirentcom.com

Intellectual Property Statement

Disclaimer of Validity

Copyright Statement

Acknowledgment

Funding for the RFC Editor function is currently provided by the Internet Society.