Network Working Group
Internet-Draft
Expires: May 24, 2007

D. Newman
Network Test
T. Player
Spirent Communications

November 20, 2006

Hash and Stuffing: Overlooked Factors in Network Device Benchmarking
draft-ietf-bmwg-hash-stuffing-07.txt

Status of this Memo

Copyright Notice

Abstract

Test engineers take pains to declare all factors that affect a given measurement, including intended 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 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 point-to-point protocol (PPP) and packet over Sonet (POS) circuits. In both areas, we provide recommendations for obtaining better repeatability in test results.

Benchmarking activities as described in this memo are limited to technology characterization using controlled stimuli in a laboratory environment, using dedicated address space.

The benchmarking network topology will be an independent test setup and MUST NOT be connected to devices that may forward the test traffic into a production network, or misroute traffic to the test management network.

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 recreated 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 common phenomena.

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

Another issue related to repeatability is the avoidance of randomness in test traffic. For example, benchmarking experience with some IEEE 802.11 devices suggests that nonrandom media access control (MAC) and IP addresses must be used across multiple trials. Although this would seem to contradict some recommendations made in this document, in fact either nonrandom or pseudorandom patterns may be more desirable depending on the test setup. There are also situations where it may be desirable to use combinations of the two, for example by generating pseudorandom traffic patterns for one test trial and then re-using the same pattern across all trials. The keywords in this document are RECOMMENDs and not MUSTs with regard to the use of pseudorandom test traffic patterns.

Note also that this discussion covers only repeatability, which is concerned with variability of test results from trial to trial on the same test bed. A separate concern is reproducibility, which refers to the precision of test results obtained from different test beds. Clearly, reproducibility across multiple test beds requires repeatability on a single test bed.

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.

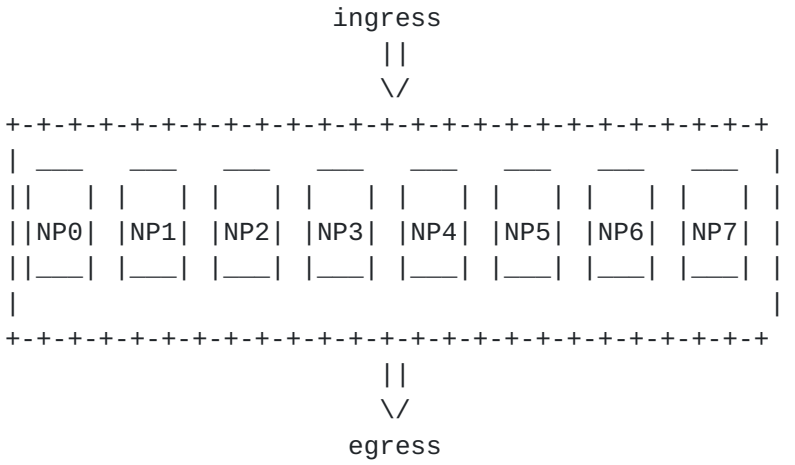
Specifically, for any random bit pattern of length L, the probability of generating that specific pattern SHOULD equal 1 over 2 to the Lth power.

4. Packet Content Variations

4.1. Problem Statement

The contents of test traffic can have a significant impact on metrics such as throughput, jitter, latency, and loss. For example, many network devices feed addresses into a hashing algorithm to determine which path upon which to forward a given packet.

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



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 capacity.

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 Benchmarking Working Group (BMWG).

Therefore, this memo adds a new consideration for benchmarking methodologies, to select traffic patterns that overcome the effects of non-randomness regardless of the hashing algorithms in use. The balance of this section offers recommendations for test traffic patterns to avoid these effects, starting at the link layer and working up to the application layer.

4.2. IEEE 802 MAC Addresses

Test traffic SHOULD use pseudorandom patterns in IEEE 802 MAC addresses. The following source and destination MAC address pattern

is RECOMMENDED:

(RR & 0xFC):PP:PP:RR:RR:RR

where (RR & 0xFC) is a pseudorandom number bitwise ANDed with 0xFC, 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 0xFC sets the high-order two bits of that byte to 0, guaranteeing a non multicast address and a non locally administered 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.

Note that the "PP:PP" designation refers to the source interface of the test instrument, not the device under test/system under test (DUT/SUT). There are situations where the DUT/SUT interface number may change during the test; one example would be a test of wireless LAN roaming. By referring to the (presumably static) source interface number of the test instrument, test engineers can keep track of test traffic regardless of any possible DUT/SUT changes.

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 MAC address. Some devices will drop frames with all-0s MAC 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 30 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 with pseudorandom patterns will be sufficient.

Experience with benchmarking of IEEE 802.11 devices suggests suboptimal test outcomes may result if different pseudorandom MAC and IP addresses are used from trial to trial. In such cases (not just for 802.11 but for any device using IEEE 802 MAC and IP addresses), testers MAY generate a pseudorandom set of MAC and IP addresses once, or MAY generate a nonrandom set of MAC and IP addresses once. In either case, the same MAC and IP addresses MUST be used in all trials.

4.3. MPLS Addressing

Similar to L2 switches, multiprotocol label switching (MPLS) devices 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

When routers make forwarding decisions based solely on destination network address, there may be no potential for hashing collision of source and destination addresses, as in the case of Ethernet switching discussed earlier. However, the potential still exists for hashing collisions to exist at the network layer, and testers SHOULD take this potential into consideration when crafting the network-layer contents of test traffic.

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. Others may perform a hash on one or more bytes in the source and destination IP addresses.

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. In particular, testers SHOULD NOT use addresses that produce the undesired effects of address processing. If, for example, a DUT can be observed to exhibit high packet loss when offered IPv4 network addresses that take the form x.x.1.x/24, and relatively low packet loss when the source and destination network addresses take the form of x.x.R.x/24 (where R is some random value between 0 and 9), test engineers SHOULD use the random pattern.

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 pseudorandom and uniformly 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.

4.6. Application-layer Patterns

Many measurements require the insertion of application-layer header(s) and payload into test traffic. Application-layer packet contents offer additional opportunities for stuffing to occur, and may also present nonrandom outcomes when fed through application layer-aware hashing algorithms. Given the vast number of application-layer protocols in use, we make no recommendation for specific test traffic patterns to be used; however, test engineers SHOULD be aware that application-layer traffic contents MAY produce nonrandom outcomes with some hashing algorithms. The same issues that apply with lower-layer traffic patterns also apply at the

application layer. As discussed in section 5, the potential for stuffing exists with any part of a test packet, including application-layer contents. For example, some traffic generators insert fields into packet payloads to distinguish test traffic. These fields may contain a transmission timestamp; sequence number; test equipment interface identifier and/or "stream" number; and a CRC over the contents of the test payload or test packet. All these fields are potential candidates for stuffing.

5. Control Character Stuffing

5.1. Problem Statement

Link-layer technologies that use high-level data link control (HDLC)-like framing may insert an extra bit or byte before each instance of a control character in traffic. These "stuffing" insertions prevent confusion with control characters, but they may also introduce significant overhead. Stuffing is data-dependent; thus selection of different payload patterns will result in frames transmitted on the media that vary in length, even though the original frames may all be of the same length.

The overhead of these escape sequences is problematic for two reasons. First, explicitly calculating the amount of overhead can be non-trivial or even impossible for certain types of test traffic. In such cases, 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 0x7E, which contains six "1" bits.

Consider the following PPP frame containing a TCP/IP packet. Not shown is the 1-byte flag sequence (0x7E), at least one of which must occur between frames.

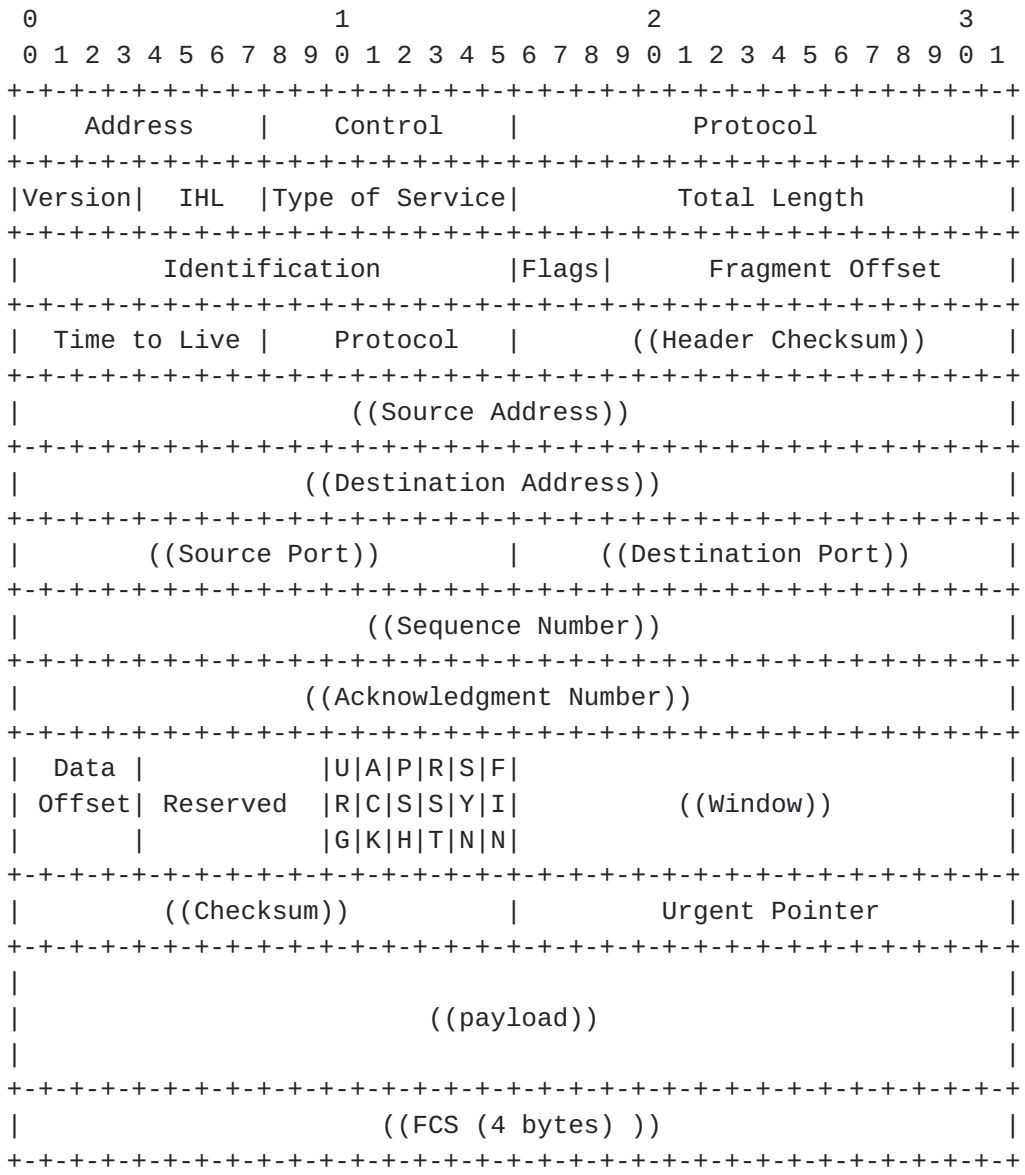
The contents of the various frame fields can be described one of three ways:

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.

3. Field contents may not be known. An example would be proprietary payload fields in test packets.

In the diagram below, 30 out of 48 total bytes in the packet headers are subject to change over the test duration. Additionally, the payload field could be subject to change both content and size. The fields containing the changeable bytes are given in ((double parentheses)).



None of the other fields are known to contain sequences subject to bit-stuffing, at least not in their entirety. Note that there is no payload in this simple example; as noted in section 4.6, the payload contents of test traffic often will present additional opportunities for stuffing to occur, and MUST be taken into account when

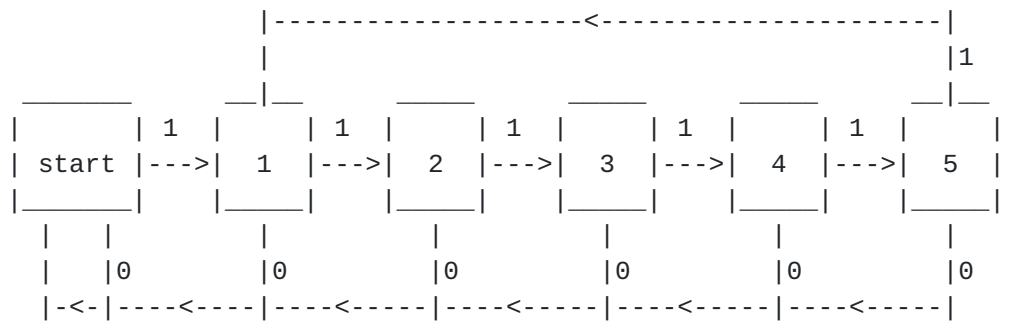
calculating stuff probability.

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, where b_n represents the "n"th bit of the string and 1 <= n <= L, the probability of b_n equalling "1" is 0.5 and the probability of b_n equalling "0" is 0.5. Additionally, the value of b_n is independent of any other 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. For an infinitely long string of uniformly random bits, we will need to insert a stuff bit if and only if state 5 is reached in the following state table.



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 diagram we can build the following transition matrix:

From \ To	start	1	2	3	4	5
start	0.5	0.5	0.0	0.0	0.0	0.0
1	0.5	0.0	0.5	0.0	0.0	0.0
2	0.5	0.0	0.0	0.5	0.0	0.0
3	0.5	0.0	0.0	0.0	0.5	0.0
4	0.5	0.0	0.0	0.0	0.0	0.5
5	0.5	0.5	0.0	0.0	0.0	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 uniformly random bits, the probability of any individual bit causing a transition to state 5, and thus causing a stuff, is 1/62.

5.2.2. Bit Stuffing for Finite Strings

For a uniformly random finite bit string of length L, we can explicitly count the number of bit-stuffs in the set of all possible strings of length L. This count can then be used to calculate the expected number of stuffs for the string.

Let f(L) represent the number of bit-stuffs in the set of all possible strings of length L. Clearly, for 0 <= L <= 4, f(L) = 0 as there are no strings of length 5. For L >= 5, f(L) = 2^(L-5) + (L-5) * 2^(L-6) + f(L-5).

A proof of this formula can be found in Appendix B.

Now, the expected number of stuffing events, E[stuffs], can be found by dividing the total number of stuffs in all possible strings by the total number of strings. Thus for any L, E[stuffs] = f(L) / 2^L.

Similarly, the probability that any particular bit is the cause of a bit-stuff can be calculated by dividing the total number of stuffs in the set of all strings of length L by the total number of bits in the set of all strings of length L. Hence for any L, the probability that L_n, where 5 <= n <= L, caused a stuff is f(L) / (L * 2^L).

5.2.3. Applied Bit Stuffing

The amount of overhead attributable to bit-stuffing may be calculated explicitly as long as the expected number of stuff bits per frame, E[bit-stuffs] is known. For long uniformly random bit-strings, E[bit-stuffs] may be approximated by multiplying the length of the string by 1/62.

% overhead = E[bit-stuffs] / framesize (in bits)

Given that the overhead added by bit-stuffing is approximately 1 in 62, or 1.6 percent, it is RECOMMENDED that testers reduce the maximum intended 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 intended load SHOULD be explicitly calculated from the test traffic.

Note that the DUT/SUT may be able to forward intended 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 byte stuffing may be calculated explicitly as long as the expected number of stuff bytes per frame, E[byte-stuffs], is known. For long uniformly random byte-strings, E[byte-stuffs] may be approximated by multiplying the length of the string by the probability that any single byte is a stuff byte.

% overhead = E[byte-stuffs] / framesize (in bytes)

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 intended 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 intended load by 0.8 percent to avoid introducing congestion.

Note that the percentages given above are approximations. For greatest precision, the actual intended load SHOULD be explicitly calculated from the test traffic

Note also that the DUT/SUT may be able to forward intended 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. This usage requires a uniform distribution, but does not have strict predictability requirements. Although it is not sufficient for security applications, the rand() function of many programming languages may provide a uniform distribution that is usable for testing purposes in lab conditions. Implementations of rand() may vary and provide different properties so test designers SHOULD understand the distribution created by the underlying function and how seeding the initial state affects its behavior.

[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 Tom Alexander, Len Ciavattone, Robert Craig, John Dawson, Neil Carter, Glenn Chagnot, Kevin Dubray, Diego Dugatkin, Rafael Francis, Paul Hoffman, David Joyner, Al Morton, Joe Perches, Jerry Perser, Scott Poretsky, Dan Romanescu, and Kris Rousey.

Appendix B. Proof of Formula for Finite Bit Stuffing

We would like to construct a function, f(L), that allows us to explicitly count the total number of bit-stuffs in the set of all strings of length L. Let S represent a bit string of length L. Additionally, let b_n be the nth bit of string S where 1 <= n <= L.

Clearly, when 0 <= L <= 4, f(L) = 0, as there can be no possible bit-stuff if there are < 5 bits.

Suppose L >= 5, then there are some number of strings that will cause stuffing events. Let us count them.

We begin by counting the number of strings that will cause at least one bit-stuff. Let us suppose that the first 5 bits, b_1,...,b_5, cause a stuffing event. Then, there are (L-5) bits that could have any value, i.e. the bits in position b_6 to b_L. So, there must be 2^(L-5) strings where the first 5 bits cause a stuff.

Now suppose that some other sequence of bits cause a stuff, b_n to b_(n+4) for some 1 < n <= L-4. In order to guarantee that b_n starts a stuff sequence, b_(n-1) must be 0, otherwise a stuff could occur at b_(n+3). Thus, there are a total of 6 bits which must have fixed values in the string, S, and a total of L-6 bits which do not have fixed values. Hence, for each value of n, there are 2^(L-6) possible strings with at least one bit-stuff for a total of (L-5) * 2^(L-6)

So, given a string of length L, where L >= 5, we know that there are 2^(L-5) + (L-5) * 2^(L-6) strings which will be transmitted with at least one stuffed bit. However, if L >= 10, then there could be more than one bit-stuff within the string S. Let Z represent a sequence of 5 sequential ones bits. Consider the bit string ..., b_n, b_(n+1), b_(n+2), Z, b_(n+8), b_(n+9), ... where 1 <= n <= L-9. For the above sequence of bits to generate two stuffing events, there must be at least one run of five sequential one's bits in ..., b_n, b_(n+1), b_(n+2), b_(n+8), b_(n+9), ... Note that the position of Z in the above sequence is irrelevant when looking for bit-stuffs. Additionally, we've already determined that the number of strings with at least one stuff in a bit string of length L is 2^(L-5) + (L-5) * 2^(L-6). Thus, the total number of stuffing events in the set of all bit strings of length L can be represented as f(L) = 2^(L-5) + (L-5) * 2^(L-6) + f(L-5) for all L >= 5.

Appendix C. Explicit Calculation of Bit Stuffing Overhead for IPv4

Consider a scenario where a tester is transmitting IPv4 test packets across a bit synchronous link. The test traffic has the following parameters (values are in decimal):

Field	Value
IP Version	4
IP Header Length	5
Type of service (TOS)	0
Datagram Length	1028
ID	0
Flags/Fragments	0
Time to live (TTL)	64
Protocol	17
Source IP	192.168.13.1-192.168.13.254
Destination IP	192.168.1.10
Source UDP Port	pseudorandom port
Destination UDP Port	pseudorandom port
UDP Length	1008
Payload	1000 pseudorandom bytes

We want to calculate the expected number of stuffs per packet, or E[packet stuffs].

First, we observe that we have 254 different IP headers to consider, and secondly, that the changing 4th octet in the IP source addresses will produce occasional bit-stuffing events, so we must enumerate these occurrences. Additionally, we must take into account that the 3rd octet of the source IP and the first octet of the destination IP will affect stuffing occurrences.

An exhaustive search shows that cycling through all 254 headers produces 51 bit stuffs for the destination IP address. This gives us an expectation of 51/254 stuffs per packet due to the changing source IP address.

For the IP CRC, we observe that the value will decrement as the source IP is incremented. A little calculation shows that the CRC values for these headers will fall in the range of 0xE790 to 0xE88F. Additionally, both the protocol and source IP address must be considered, as they provide a source of extra leading and trailing ones bits.

An exhaustive search shows that cycling through all 254 headers will produce 102 bit stuffs for the CRC. This gives us an expectation of 102/254 stuffs per packet due to the CRC.

Since our destination IP address is even and the UDP length is less than 32768, the random source and destination ports provide 32 bits of sequential random data without forcing us to consider the boundary bits. Additionally, we will assume that since our payload is pseudorandom, our UDP CRC will be too. The even UDP length field again allows us to only consider the bits explicitly contained within the CRC and data fields. So, using the formula for the expected number of stuffs in a finite string from section 5.2.2, we determine that E[UDP stuffs] = f(32)/2^32 + f(8000+16)/2^(8000+16). Now, f(32)/2^32 is calculable without too much difficulty and is approximately 0.465. However, f(8016)/2^8016 is a little large to calculate easily, so we will approximate this value by using the probability value obtained in section 5.2.1. Thus, E[UDP] ~ 0.465 + 8016/62 ~ 129.755.

Hence, E[packet stuffs] = 51/254 + 102/254 + 129.755 = 130.357. However, since we cannot have a fractional stuff, we round down to 130. Thus, we expect 130 stuffs per packet.

Finally, we can calculate bit-stuffing overhead by dividing the expected number of stuff bits by the total number of bits in the IP datagram. So, this example traffic would generate 1.58% overhead. If our payload had consisted exclusively of zero bits, our overhead would have been 0.012%. An all ones payload would produce 19.47% overhead.

Appendix D. Explicit Calculation of Bit Stuffing Overhead for IPv6

Consider a scenario where a tester is transmitting IPv6 test packets across a bit synchronous link. The test traffic has the following parameters (values are in decimal except for IPv6 addresses, which are in hexadecimal):

Field	Value
IP Version	6
Traffic Class	0
Flow Label	pseudorandom label
Payload Length	1008
Next Header	17
Hop Limit	64
Source IP	DEAD:0:0:1::1-DEAD:0:0:1::FF
Destination IP	DEAD:0:0:2::10
Source UDP Port	pseudorandom port
Destination UDP Port	pseudorandom port
UDP Length	1008
Payload	1000 pseudorandom bytes

We want to calculate the expected number of stuffs per packet, or E[packet stuffs].

First, we observe that we have 255 different IP headers to consider, and secondly, that the changing 4th quad in the IP source addresses will produce occasional bit-stuffing events, so we must enumerate these occurrences. Additionally, we must take into account that the first quad of the destination IP address will affect stuffing occurrences, since binary representation of the address provides leading 1's bits.

An exhaustive search shows that cycling through all 255 headers produces 36 bit stuffs for the destination IP address. This gives us

an expectation of 36/255 stuffs per packet due to the changing source IP address.

We also have to consider our pseudorandomly generated flow label. However, since our Traffic Class field is 0 and our Payload Length field is less than 32768 (and thus the leading bit of the Payload Length field is 0), we may consider the flow label as 20 bits of random data. Thus the expectation of a stuff in the flow label is f(20)/2^20 ~ .272.

Similar to the flow label case above, the fourth quad of our destination IP address is even and the UDP length field is less than 32768, so the random source and destination ports provide 32 bits of sequential random data without forcing us to consider the boundary bits. Additionally, we will assume that since our payload is pseudorandom, our UDP CRC will be too. The even UDP length field again allows us to only consider the bits explicitly contained within the CRC and data fields. So, using the formula for the expected number of stuffs in a finite string from section 5.2.2, we determine that E[UDP stuffs] = f(32)/2^32 + f(8000+16)/2^(8000+16). Now, f(32)/2^32 is calculable without too much difficulty and is approximately 0.465. However, f(8016)/2^8016 is a little large to calculate easily, so we will approximate this value by using the probability value obtained in section 5.2.1. Thus, E[UDP stuffs] ~ 0.465 + 8016/62 ~ 129.755.

Now we may explicitly calculate that E[packet stuffs] = 36/255 + 0.272 + 129.755 = 130.168. However, since we cannot have a fractional stuff, we round down to 130. Thus, we expect 130 stuffs per packet.

Finally, we can calculate bit-stuffing overhead by dividing the expected number of stuff bits by the total number of bits in the IP datagram. So, this example traffic would generate 1.55% overhead. If our payload had consisted exclusively of zero bits, our overhead would have been 0.010%. An all ones payload would produce 19.09% overhead.

Appendix E. Terminology

Hashing

Also known as a hash function. In the context of this document, an algorithm for transforming data for use in path selection by a networking device. For example, an Ethernet switch with multiple processing elements might use the source and destination MAC addresses of an incoming frame as input for a hash function. The hash function produces numeric output that tells the switch which processing element to use in forwarding the frame.

Randomness

In the context of this document, the quality of having an equal probability of any possible outcome for a given pattern space. For example, if an experiment has N randomly distributed outcomes, then any individual outcome has a 1 in N probability of occurrence.

Repeatability

The precision of test results obtained on a single test bed, but from trial to trial. See also "reproducibility."

Reproducibility

The precision of test results between different setups, possibly at different locations. See also "repeatability."

Stuffing

The insertion of a bit or byte within a frame to avoid confusion with control characters. For example, RFC 1662 requires the insertion of a 0 bit prior to any sequence of five contiguous 1 bits within a frame to avoid confusion with the HDLC control character 0x7E.

Authors' Addresses

David Newman
Network Test

Email: dnewman@networktest.com

Timmons C. Player
Spirent Communications

Email: timmons.player@spirent.com

Full Copyright Statement

Intellectual Property

Acknowledgment

Funding for the RFC Editor function is provided by the IETF Administrative Support Activity (IASA).