Network Testing with Simulated Traffic. Does Realism Matter?

An Ixia BreakingPoint Case study
# Table of Contents

- Introduction .................................................................................................. 4  
- Background .................................................................................................. 4  
- No Big Deal? ................................................................................................. 4  
- The Reality .................................................................................................... 4  
- The Theory .................................................................................................... 5  
- Underlying Algorithm .................................................................................. 5  
- Example 1 .................................................................................................... 6  
- Example 2 .................................................................................................... 7  
- Data Matters ................................................................................................ 8  
- The Practice ................................................................................................. 8  
- Affected Devices ......................................................................................... 8  
- Proxy .......................................................................................................... 8  
- Firewall ...................................................................................................... 9  
- The Only Conclusion .................................................................................. 11
Introduction

Background

Realism has been a long-standing focus of Ixia BreakingPoint products. From the beginning, the concept of superflows has allowed users to model the relationship between related types of traffic. Markov text generation generates content for application traffic that looks a whole lot like natural language. It’s an application of Markov chains that builds sequences of words based on the probabilities of a particular word following the previous word. Most people have seen some examples of this technique in their spam folders, because spammers use it to make their auto-generated emails harder for spam filters to spot. It’s actually kind of fun to play with because it produces a babble of words that you could believe a real person might say, albeit kind of a crazy person.

Markov text is just one feature in the property of BreakingPoint tests that we put a lot of work into for every release, which is realism.

No Big Deal?

It’s not that unusual for someone to be dismissive of how much realism features like Markov text actually matters. “It’s neat,” they say, “but my device is just a (whatever) so a packet is just a packet to it. The content doesn’t really matter.”

When a BreakingPoint device is evaluated in a lab with that point of view, there is another conversation that often follows. “We’ve been testing using vendor x for a few years now, and consistently transmit 700 Mbps through our device. Your test device is only transmitting 420 Mbps. What’s wrong with your device?”

Our follow-up to that question is generally to ask for a pcap of the traffic that gives good performance. If we disable realism features until we match their old traffic, the lost performance miraculously comes back. “What’s wrong” turns out to be that our default behavior is making the device under test work harder.

The Reality

It is surprising how often this series of events is followed by a request to make our product work just like whatever the lab is accustomed to. It’s too hard politically to give up the performance numbers that they’ve been claiming. The easy path is to keep things unchanged.

There are two facts that make that a poor choice. First, the BreakingPoint traffic much more closely resembles traffic seen in the real world. Second, and most importantly, at some point every device leaves the lab. If the tests in the lab are a cakewalk, it simply means that there can be a latent problem that is found in the field after deployment, possibly even by a customer. The real world gets the final word.
The Theory

Underlying Algorithm

When a device is doing deep packet inspection, it needs to use some kind of matching algorithm. A very common example is the Boyer-Moore string search algorithm. This algorithm was developed by Bob Boyer and J Strother Moore in 1977, and is a generally-recognized benchmark for fast searching.

The Boyer-Moore algorithm works by processing the string that is being searched for and then building two tables defining the behavior after a mismatch. Those tables allow the search to use the information gained when checking for a match in one location to rule out as many potential match positions as possible. Each check that results in a mismatch is followed by a table lookup to see whether that check included a substring of the match expression, and how far the expression must shift over to align with that substring. Whichever table provides the largest shift is used.

To get the maximum benefit from these tables, the Boyer-Moore algorithm starts making its comparison from the end of the string it’s matching. That way, if the character that’s examined doesn’t appear in any substring of the match string, the whole string cannot fit where it is and can be moved over by its entire length.

To illustrate, consider what is involved with searching for the string “mana”.

First, we build a table defining how much of what was seen is a possible match. The value for each row is the number of characters the pattern would need to shift to potentially

<table>
<thead>
<tr>
<th>Observed</th>
<th>Matches</th>
<th>Substring</th>
<th>Shift</th>
</tr>
</thead>
<tbody>
<tr>
<td>?</td>
<td>0</td>
<td>The first character examined was something other than ‘a’, so the pattern must shift one to potentially match.</td>
<td>1</td>
</tr>
<tr>
<td>?a</td>
<td>1</td>
<td>What was examined was something other than ‘n’ followed by ‘a’. That is a potential match for mana so the pattern must shift by 2 to align.</td>
<td>2</td>
</tr>
<tr>
<td>?na</td>
<td>2</td>
<td>Something other than ‘a’ followed by ‘na’ does not appear in the string, so must shift by the length of the whole pattern</td>
<td>4</td>
</tr>
<tr>
<td>?ana</td>
<td>3</td>
<td>Something other than ‘m’ followed by ‘ana’ does not appear in the string, so must shift by the length of the whole pattern</td>
<td>4</td>
</tr>
</tbody>
</table>
Next, calculate the second table. For this table, starting at the second-to-last character, each character seen that is not already in the table is added with a value of its distance from the rightmost character. This is sometimes referred to as the ‘bad character shift’. When using this table, the result is how far the bad character is from the rightmost character minus how far the current character we are examining is from the rightmost table. In other words, the result is how far we would need to shift the comparison to line up the bad character with where it exists in the search pattern.

<table>
<thead>
<tr>
<th>Character</th>
<th>Shift</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>1</td>
</tr>
<tr>
<td>a</td>
<td>2</td>
</tr>
<tr>
<td>m</td>
<td>3</td>
</tr>
<tr>
<td>any other character</td>
<td>4</td>
</tr>
</tbody>
</table>

### Example 1

With the preparation in place, consider how Boyer-Moore will perform when searching within the string ‘***************************’.

1. The match string will be tried in the leftmost position. The first character checked does not match.

```
* * * * * * * * * * * * * * * m a n a
```

2. The maximum shift comes from the ‘bad character’ table, and is 4. Since ‘*’ doesn’t appear anywhere in the match string, the match string can’t possibly fit anywhere that includes that position, so is shifted over by its entire length.

```
* * * * * * * * * * * * * * * m a n a
```

3. Once again there is no substring match, so once again the search string can be shifted by its entire length.

```
* * * * * * * * * * * * * * * m a n a
```

4. This process will repeat 3 more times before running past the end of the source data.

This example shows the best possible performance from the Boyer-Moore algorithm. Uniform data that doesn’t match any substring of the match string can be skipped through very quickly, with only a light load on the processor.
Example 2

Next consider how Boyer-Moore will perform when searching within the string ‘a man a plan a canal panama’.

1. The match string will be tried in the leftmost position. The characters ‘ma’ will be examined before a mismatch is found.

```
a   m a n   a   p l a n   a   c a n a l   p a n a m a
m a n a
```

2. In this case, both tables give a shift of 2, so the match string is shifted by that amount and tested again.

```
a   m a n   a   p l a n   a   c a n a l   p a n a m a
m a n a
```

3. In this position, the first character examined is a mismatch. The bad character table gives a shift of 4.

```
a   m a n   a   p l a n   a   c a n a l   p a n a m a
m a n a
```

4. This results in another shift of 4.

```
a   m a n   a   p l a n   a   c a n a l   p a n a m a
m a n a
```

5. Here the bad character table gives a shift of 4.

```
a   m a n   a   p l a n   a   c a n a l   p a n a m a
m a n a
```

6. This time both tables give a shift of only 1.

```
a   m a n   a   p l a n   a   c a n a l   p a n a m a
m a n a
```

7. It’s notable that in this pass, the algorithm is reexamining characters that were used in the last comparison. This time, the both tables give a shift of 4.

```
a   m a n   a   p l a n   a   c a n a l   p a n a m a
m a n a
```

8. The bad character table then allows us to shift by 4.

```
a   m a n   a   p l a n   a   c a n a l   p a n a m a
m a n a
```

9. And finally, we have run out of room to shift.
This example is a somewhat more difficult case for Boyer-Moore. Since the data contains matches for the substring, there is less opportunity to shift the match and skip parts of the data. In fact, in step 7 of this example, a character seen in step 2 must be reexamined in a new position.

Data Matters

In the examples above, very uniform data can be seen to be a best-case scenario for the Boyer-Moore Algorithm, while a random string of characters that happens to overlap the set of characters in the match pattern is approaching a worst-case. A real-world example would fall somewhere in between those. Whether a device uses Boyer-Moore specifically or some other algorithm, the fact is that the specific content that it must examine has a dramatic effect on the work load of the device.

Thus, it’s critically important when evaluating a device to use data that closely-resembles the content it is likely to see in the real world. That’s the only way to get a view of the actual performance of the device.

The Practice

Affected Devices

The remainder of this document will look at the performance of three specific devices when faced with real-world traffic. The selection of devices includes a proxy and a firewall. These devices include DPI as an optional item in their job description, so it should be no surprise to see the realism of the traffic influence their performance.

Proxy

For the first example, we will consider a real-world proxy device. It’s tempting to think that for a proxy, all data is the same, and it is only the quantity that matters, but experience suggests otherwise.

This specific proxy had been tested for some time with another vendor’s test tool. Using that tool, the proxy was consistently achieving a bandwidth of 700 Mbps. At that bandwidth, the measured CPU load on the proxy was 80%.

The device was tested with a BreakingPoint device using a test that was constructed to match the former test as closely as possible, with particular attention to bandwidth and amount of data. Using this test, the device was only capable of 420 Mbps, and was measured to be at 96% CPU load.
Firewalls are primarily concerned with TCP sessions. Surely the application data is irrelevant, right? Once again, experience does not bear this out.

Upon investigation, it was determined that one simple change allowed the device to perform as it had been before - the dynamic, realistic HTTP content generated by the BreakingPoint device need only be replaced with static content taken from a packet capture from the old test device. With only that change, and all other test parameters kept the same, the device showed its previous performance.

The implications of this are very important. A real proxy seemed to be performing fine in the lab, thanks to the fake content it was being fed. That same device when faced with realistic traffic lost nearly 50% of its performance. If put into the real world with that performance characteristic, it is likely to disappoint its buyer.

Firewall

It may not be surprising that a proxy device’s performance is highly dependent on the application data content. What about a firewall? Firewalls are primarily concerned with TCP sessions. Surely the application data is irrelevant, right? Once again, experience does not bear this out.

A customer’s firewall device had been tested using a competing product for some time, and had been achieving an HTTP transaction rate of 100,000 transactions per second consistently. Faced with realistic data, the firewall’s performance was reduced to only 87,000 transactions per second.
This difference is not quite as dramatic as what was seen with the proxy, but it is still easy to imagine a buyer of this firewall being disappointed in its performance when being placed in a real network.
The Only Conclusion

If a proxy and even a firewall are directly affected by the realism of the application data, it’s reasonable to believe that there is some chance of being affected for any device. Most devices, especially high-end devices, include some advanced features that may make them susceptible to influence by the data content. Given this, it’s vital to make the data used in testing any device as realistic as possible. Accepting less simply adds risk and disappoints your customers.

...it’s vital to make the data used in testing any device as realistic as possible. Accepting less simply adds risk and disappoints your customers.