Packet Size Pluggable Transport and Traffic Morphing

George Kadianakis
asn@torproject.org

Tor Tech Report 2012-03-004
March 13, 2012

1 Traffic Morphing

1.1 Overview

It is well known [1] that Tor traffic can be distinguished from other network protocols by its distinctive packet size. Due to the sized Tor cells, most TCP packets of Tor traffic are 586 bytes in size (See Figure 1).

On the other hand, HTTPS, the protocol that Tor tries to simulate,[1] has a much more spread out packet size probability distribution (See Figure 2)

This means that an adversary can detect Tor traffic by using packets of size *586* as distinguishers [2] [1].

1.2 Solutions

An obvious solution to this problem is to introduce a padding scheme to Tor.

Some network protocols already use padding to defend against traffic fingerprinting attacks. SSH and TLS, for example, both support padding in their messages[2,3]. Most implementations of those protocols don't pad by default. The ones that do, add a random amount of padding to the protocol message.

The Tor protocol also supports padding cells (named *PADDING* and *VPADDING*) which attempt to generate covert traffic.

While the random padding solution effectively hides the *586* bytes identifier, it does not fully make Tor traffic resemble HTTPS traffic. This means that a sophisticated attacker can distinguish Tor traffic, by searching for SSL handshakes that try to look like HTTPS but actually follow a packet size probability distribution different from the one of HTTPS.

A solution to that, is to collect a large amount of HTTPS packets and form their packet size probability distribution. We can then randomly sample that probability distribution for

[1] https://lists.torproject.org/pipermail/tor-dev/2011-January/001077.html
[2] https://www.ietf.org/rfc/rfc4344.txt
[3] https://www.gnu.org/software/gnutls/manual/gnutls.html#On-Record-Padding

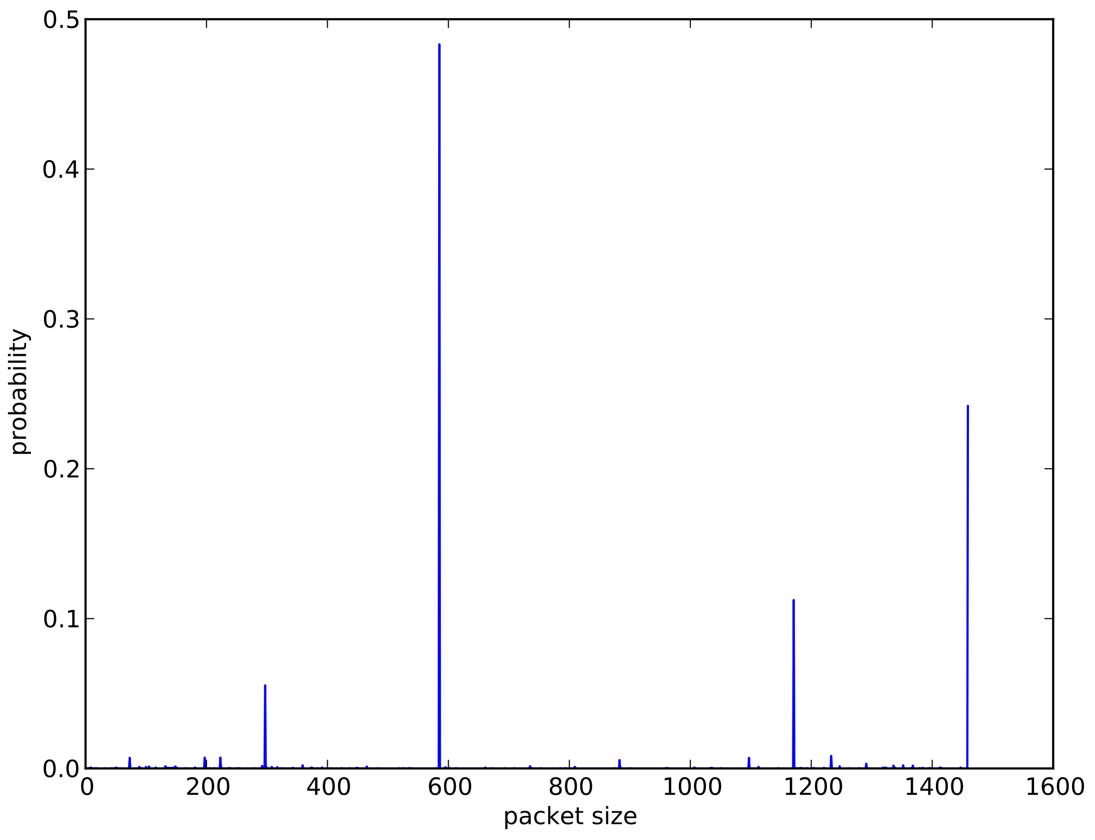

Figure 1: Packet size probability distribution of Tor Client-to-Server traffic

every packet we need to send. Specifically, for every Tor packet, we would sample the HTTPS probability distribution, find a target packet size and split or pad our packet accordingly. We call this method *random sampling*.

The problem with random sampling, is that it radically increases our bandwidth overhead.

To reduce this overhead, we looked into a paper of Charles Wright, Scott Coulls and Fabian Monrose called *Traffic Morphing: An efficient defense against statistical traffic analysis* [4].

1.3 Traffic Morphing

Traffic Morphing minimizes bandwidth overhead when transforming packets from one packet size probability distribution to another. That is, given two packet size probability distributions, one for the source protocol and another for the target protocol, it produces a *morphing matrix* that acts as an oracle and describes how to efficiently morph packets between those two probability distributions.

Traffic Morphing works by modeling the problem of transforming packets as a problem of linear programming. To construct the morphing matrix, we consider *bandwidth overhead* as

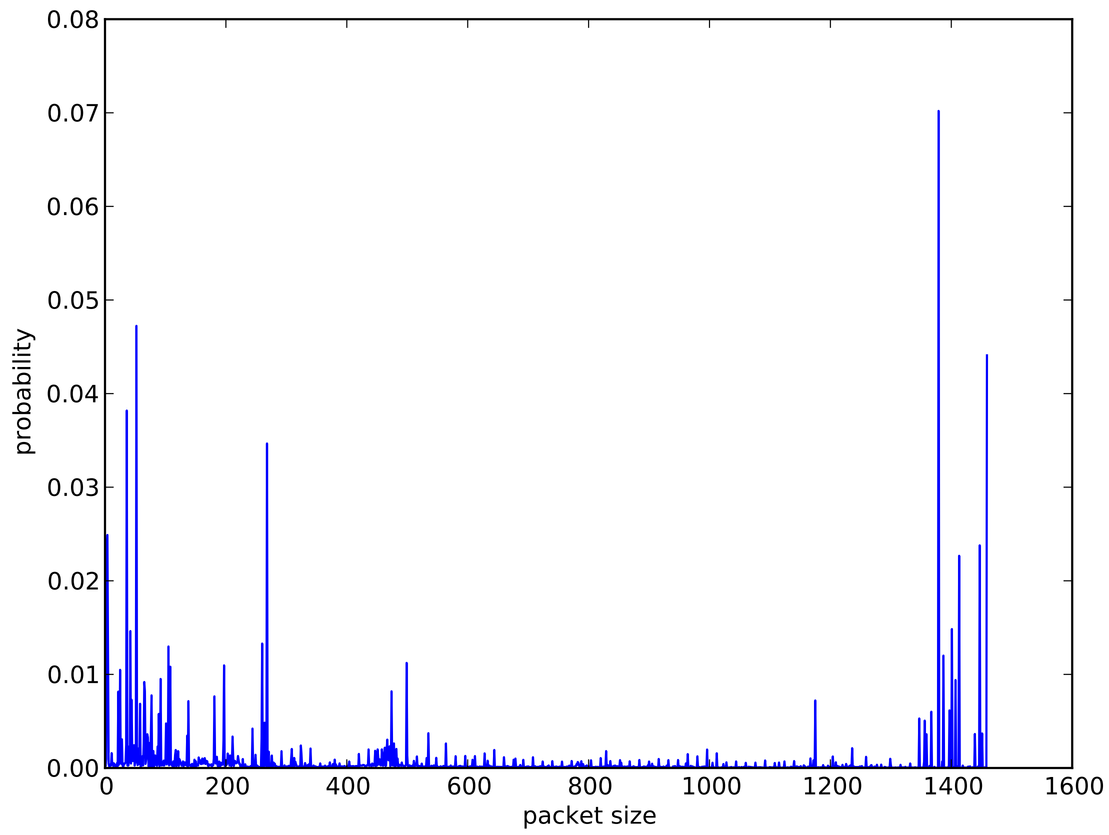

Figure 2: Packet size probability distribution of HTTPS Client-to-Server traffic

the objective function of our linear program, and use appropriate problem constraints that transform the source probability distribution to the target probability distribution.

2 Woes of Traffic Morphing

2.1 Issues

Unfortunately, morphing matrices are not a panacea, for the following reasons:

2.1.1 Statelessnes

Morphing matrices are not stateful. In other words, morphing matrices don't handle protocols whose packet size probability distribution changes through the protocol runtime.

For example, HTTPS follows different packet size probability distributions between the handshake and the data phase, and morphing matrices are unable to understand the difference.

2.1.2 Splitting

The original traffic morphing paper did not specify an algorithm for splitting packets: When a morphing matrix suggests a packet size **smaller** than the original packet size, we have to split the original packet into two parts. The problem is that the original paper does not clearly define what should be done with the second half of the split packet.

Sending the second half of the packet to the network is not correct, because its packet size does not belong to the packet size probability distribution of the target protocol (specifically, it belongs to the packet size probability distribution of the target protocol when split once by its morphing matrix).

Querying the morphing matrix again, for the size of the second half of the split packet, is also not correct, since that packet size does not belong to the probability distribution of the source protocol and morphing matrices mishandle unknown packet sizes.

Even though the original paper didn't specify the splitting algorithm that should be used, it hinted that doing a random sampling of the target probability distribution for the length of the split packet is what they did in their implementation.

2.2 Evaluation of issues

Unfortunately, the splitting problem of the previous section is not only theoretical. To evaluate the bandwidth overhead of morphing matrices, we made software[4,5] that simulates the morphing of a large number of packets. Specifically, our software morphs 500000 packets using both random sampling and traffic morphing, and plots the overhead. (In traffic morphing, we use random sampling for the second parts of a split packet.)

We can see that in the case of Server-to-Client Tor-to-HTTPS packet morphing, morphing matrices actually reduce the bandwidth overhead by a substantial amount (See Figure 3).

However, in the case of Client-to-Server Tor-to-HTTPS packet size morphing, the bandwidth overhead of morphing matrices is actually larger than the bandwidth overhead of random sampling (See Figure 4).

Our guess is that this happens because in the Client-to-Server case, HTTPS has large probabilities of outputting small packet sizes (See Figure 2).

This means that our morphing matrix tries to split our packets into small packets, and then when random sampling is used for the second half of our packet it tries to pad it to 1460 (which is also a very popular packet size in HTTPS):

For example:

```
Packet size 586. We must morph it to 127. Splitting to 127+459 and sending 127.
Packet size 459. We must morph it to 123. Splitting to 123+336 and sending 123.
Packet size 336. We must morph it to 1459. Padding with 1123 and sending.
```

In this case, morpher gets a packet of 586 bytes to morph. It queries the morphing matrix which suggests a packet size of 127 bytes. Morpher splits the original packet to 127+459 bytes, sends 127 bytes to the network, and is left with 459 bytes. Since the packet was split, our faulty

[4] https://trac.torproject.org/projects/tor/ticket/5023
[5] https://lists.torproject.org/pipermail/tor-dev/2011-January/001077.html

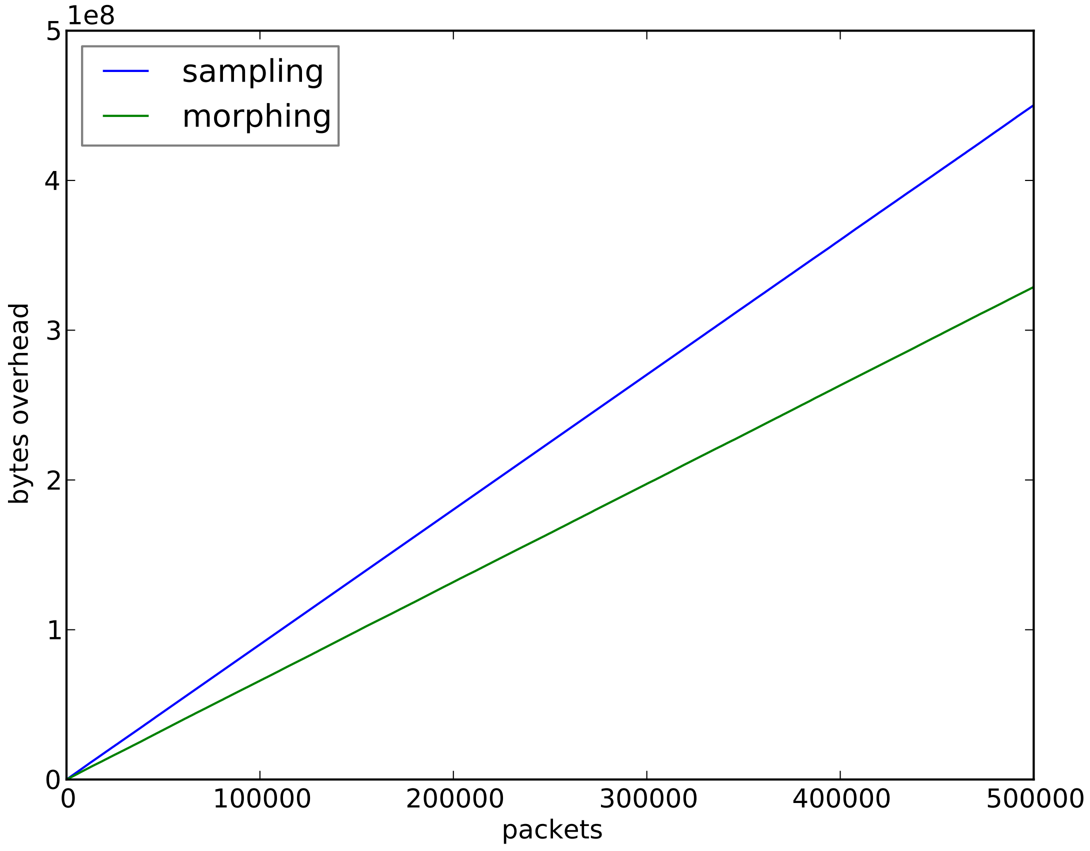

Figure 3: Bandwidth overhead for 0.5 million Server-to-Client packets

splitting algorithm says that morpher should do a direct sampling over the HTTPS probability distribution. Morpher gets 123 bytes as the result of random sampling, and splits the packet to 123+336 bytes. It sends 123 bytes to the network, and is left with 459 bytes. Morpher again does random sampling over the HTTPS probability distribution, and gets 1459 bytes as the result. This means that it must pad the 336 bytes packet to 1459 bytes. This results in an overhead of 1123 bytes.

For the same case, random sampling was better due to the randomness of random sampling.

2.3 Fixes

A potential fix for the statelessness of morphing matrices, is to generate multiple morphing matrices for each phase of the protocol. We have not attempted this approach.

A potential fix for the splitting algorithm issue, is to generate a morphing matrix for every split level (since the number of splits that can happen to a packet are finite), and use the appropriate morphing matrix everytime we have a split packet. We have not implemented or evaluated this fix yet.

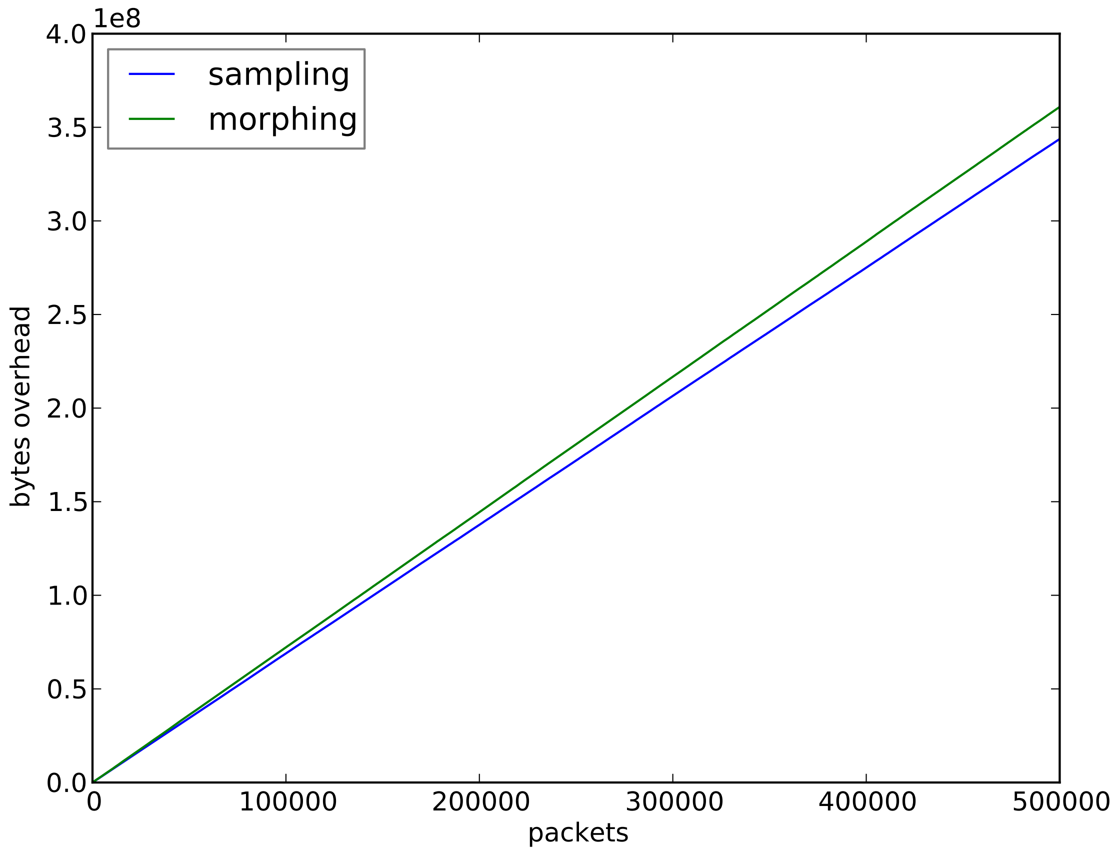

Figure 4: Bandwidth overhead for 0.5 million Client-to-server packets

3 Future

For the short-term future, we decided to postpone the development of pluggable transports based on Traffic Morphing. We think that more research is needed, as well as a stronger focus towards realistic adversaries.

Specifically, we believe that if we investigate the strength and limitations of modern Deep Packet Inspection kits, we can provide adequate defences without the implementation complexity and drawbacks of the Traffic Morphing technology.

We plan on investigating packet size pluggable transports with less overhead, even if they don't deliver a perfect probability distribution match to the target protocol, since we believe that such pluggable transports can effectively mitigate many practical packet-size-based detection attacks.

At the same time, we also think it's important to remain up-to-date with modern academic research. For example, modern website fingerprinting research has showed that any kind of padding scheme is insufficient to protect against sophisticated packet size distinguishers [2] [1]. Furthermore, the academic community has found other network traffic features that can be used as distinguishers with surprisingly high accuracy [3].

Meanwhile, we are also researching alternative pluggable transports with better resistance against payload fingerprinting, which we think real-life attackers are likely to do, and also pluggable transports which can get through HTTP proxy servers.

Acknowledgments

Thanks to Steven J. Murdoch and the Tor Project for the fruitful conversations on packet size pluggable transports.

The packet length probability distributions were formed after analysis of traffic traces kindly provided by CAIDA. The traffic traces were captured by monitoring an Equinix datacenter in Chicago, IL.[6],[7]

References

[1] Dominik Herrmann, Rolf Wendolsky, and Hannes Federrath. Website fingerprinting: attacking popular privacy enhancing technologies with the multinomial naïve-bayes classifier. In *Proceedings of the 2009 ACM workshop on Cloud computing security (CCSW '09)*, pages 31–42, New York, NY, USA, 2009. ACM. http://freehaven.net/anonbib/#ccsw09-fingerprinting.

[2] Marc Liberatore and Brian Neil Levine. Inferring the Source of Encrypted HTTP Connections. In *Proceedings of the 13th ACM conference on Computer and Communications Security (CCS 2006)*, pages 255–263, October 2006. http://freehaven.net/anonbib/#Liberatore:2006.

[3] Andriy Panchenko, Lukas Niessen, Andreas Zinnen, and Thomas Engel. Website fingerprinting in onion routing based anonymization networks. In *Proceedings of the Workshop on Privacy in the Electronic Society (WPES 2011)*. ACM, October 2011. http://freehaven.net/anonbib/#wpes11-panchenko.

[4] Charles Wright, Scott Coull, and Fabian Monrose. Traffic morphing: An efficient defense against statistical traffic analysis. In *Proceedings of the Network and Distributed Security Symposium - NDSS '09*. IEEE, February 2009. http://freehaven.net/anonbib/#morphing09.

[6] https://gitorious.org/morpher/morpher/blobs/master/ACKNOWLEDGMENTS
[7] https://gitorious.org/morpher/morpher/trees/master/data