Packet Size Pluggable Transport and Traffic Morphing

March 6, 2012

0.1 Traffic Morphing

0.1.1 Overview

It is well known [4] that Tor traffic can be distinguished from other network prtocols by its distinctive packet size. Due to the static sized Tor cells, most TCP packets of Tor traffic are 586 bytes in size (See Figure 1).

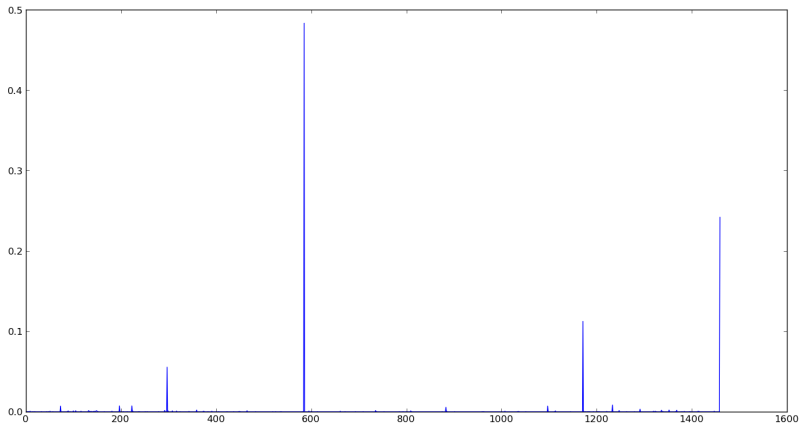

Figure 1: Packet size probability distribution of Tor Client-to-Server traffic

On the other hand, HTTPS, the protocol that Tor tries to simulate [5], has a much more spread out packet size probability distribution (See Figure 2).

This means that an adversary can easily detect Tor traffic by using packets of size *586* as distinguishers [3] [4].

0.1.2 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 [1] [2]. Most implementations of those protocols don't pad by default. The ones that do, add a random amount of padding to the protocol message.

While the random padding solution effectively hides the *586* bytes identifier, it does not contribute at making Tor traffic resemble HTTPS traffic. This means that a sophisticated attacker can distinguish Tor traffic, by searching for

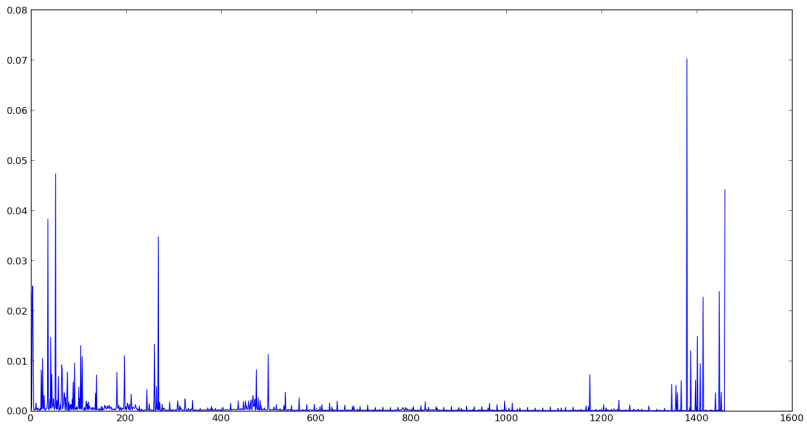

Figure 2: Packet size probability distribution of Tor Client-to-Server traffic

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 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 *direct sampling*.

The problem with direct 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*.

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

0.2 Woes of Traffic Morphing

0.2.1 Issues

Unfortunately, morphing matrices are not a panacea:

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, when the handshake phase of the protocol tends to have a different packet size probability distribution than the rest of the protocol).

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 splitted 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 protoocol when splitted once by its morphing matrix).

Querying the morphing matrix again, for the size of the second half of the splitted 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 direct sampling of the target probability distribution for the length of the splitted packet is what they did in their implementation.

0.2.2 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 [6] that simulates the morphing of a large number of packets. Specifically, our software morphs 500000 packets using both direct sampling and traffic morphing, and plots the overhead. (In traffic morphing, we use direct sampling for the second parts of a splitted 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).

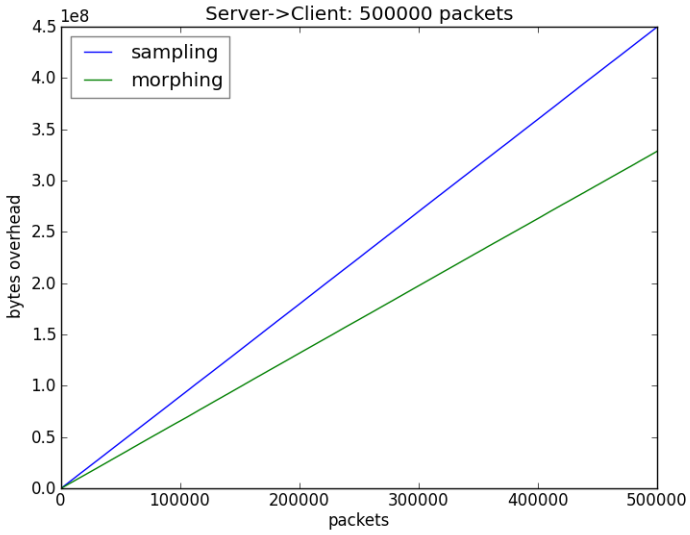

Figure 3: Bandwidth overhead for 0.5 million Server-to-Client packets

Still, 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 direct sampling (See Figure 4).

Our guess is that this happens because in the C-¿S 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 packet, and then when direct 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.
```

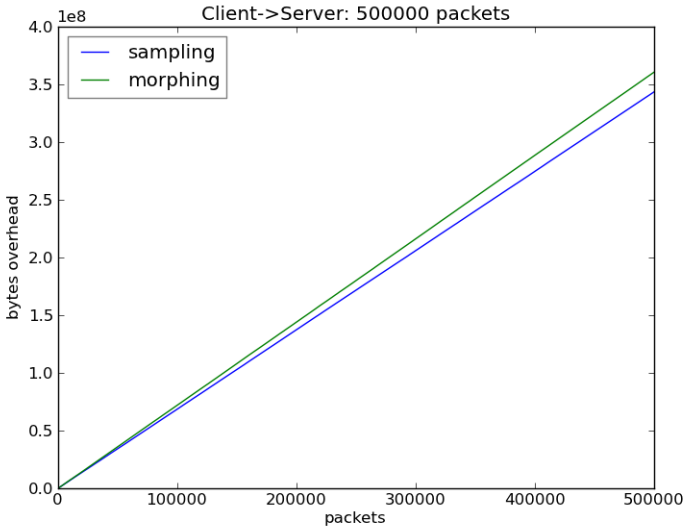

Figure 4: Bandwidth overhead for 0.5 million Client-to-server packets

```
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 splitting algorithm says that morpher should do a direct sampling over the HTTPS probability distribution. Morpher gets 123 bytes as the result of direct 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 direct 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, direct sampling was better due to the randomness of direct sampling.

0.2.3 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 splitted packet. We started implementing this approach ¡link¿ and it seems to help.

0.3 Future

We decided to stall the development of the original morpher transport, because of the above problems which had no trivial solution

Another problem is that moden website fingerprinting research has showed that any kind of padding scheme is not sufficient to protect against semi-sophisticated packet size distinguishers [3] [4] . And even if padding was sufficient to protect against those distinguishers, there are other traffic features which are sufficient for distinguishing network traffic with surprisingly high accuracy [7].

Bibliography

[1] https://www.ietf.org/rfc/rfc4344.txt

[2] https://www.gnu.org/software/gnutls/manual/gnutls.html#On-Record-Padding

[3] M. Liberatore, B. N. Levine, *Inferring the Source of Encrypted HTTP Connections*, CCS2006, October 2006.

[4] Dominik Herrmann, Rolf Wendolsky, and Hannes Federrath. 2009. Website fingerprinting: attacking popular privacy enhancing technologies with the multinomial nave-bayes classifier.

[5] http://archives.seul.org/or/dev/Jan-2011/msg00029.html

[6] *Morpher pluggable transport: Select algorithm for packet size morphing*

https://trac.torproject.org/projects/tor/ticket/5023
http://archives.seul.org/or/dev/Jan-2011/msg00029.html

[7] Andriy Panchenko, Lukas Niessen, Andreas Zinnen, and Thomas Engel. 2011. *Website fingerprinting in onion routing based anonymization networks.*