A very comparable solution is described in the paper, “Proactive Server Roaming for Mitigating Denial-of-Service Attacks.” It is very comparable for the following reason: they roam their server instead of “floating” overlay nodes as in our proposed solution. Additionally, they rely upon having a large pool of servers (N servers) available to choose from such that an attacker, even if he knows of the entire pool of servers, may not know which one is active at any one time, thus he may have to dilute his attack by spreading it across the N servers. This is also comparable to our approach, in that we have N floating overlay nodes, but only M are active, such that even if an attacker knows all N overlay nodes, he may have to dilute his attack by spreading it across the N servers.

The biggest problem with comparing our results to theirs is the fact that they use TCP (they use state-migration schemes, which is probably be outside the scope of our present work). However, it may be trivial to generate results from their work using UDP instead, in which case a more apples-to-apples comparison can be done.

Per Flow Packet Sampling for High-Speed Network Monitoring

So, a big part of our proposed solution is avoiding having to do per flow packet monitoring. In this paper, they propose a sampling method instead to estimate it. The biggest problem with this one is, it’s very different from our proposed solution in which we use “roaming” overlay nodes. Roaming overlay nodes has many other benefits also – we could use these extra benefits to easily show an improvement in our solution over theirs on certain patterns of attack given certain resource limits.

Given how starkly different this approach is (in fact, I would argue that our proposed solutions are mutually beneficial in that they can be combined to, for instance, prune the “tree” way before logN), I don’t think this is the best candidate to compare our results to.

“Roaming Honeypots for Mitigating Service-level Denial-of-Service Attacks” is a very interesting solution.

First, they use ns-2 to do experiments, as we will probably do.

Second, it’s comparable to our solution in that only M severs out of a pool of N servers are active at any one time. Some subset of these N-M servers are made into honeypots in which, if traffic is directed to them, they assume that it is attack traffic (since authorized users know which servers are active and therefore would avoid hitting the inactive servers). This has the disadvantage, as you described elsewhere, in that legit users turned bad or hijacked users can stir up trouble without being detected (within their proposed framework).

It should be pointed out that I see a lot of synergy between our two proposed solutions, especially in the sense that some of our inactive overlay nodes could in theory act as honeypots. The only problem with doing this is, for instance, a harmless user might not have received his latest user agent proxy (overlay ingress node), so he may direct traffic to the honeypot anyway. We could keep track of which user agent proxy a user thinks he or she is assigned to (which user agent proxy update did this user last acknowledge?), and give those users a pass if they try to access it when it is a honeypot, but this (a) further complicates our design, and (b) is outside the scope of our current work. But it does point the way towards combining proposed solutions. I think this idea, by itself, of combining multiple proposals to see how well they work together (e.g., making up for one another’s weaknesses), is pretty interesting.

Our proposed solution and theirs attempt to detect attackers and deals with them accordingly – moreover, they do packet filtering on upstream nodes once they detect them, just like we do (upstream nodes in our case are the proxies). Our detection mechanisms differ in that they use a honeypot and we use a splitting algorithm.

Pingali’s “Protecting Public Servers from DDoS Attacks Using Drifting Overlays” is also very comparable. They use a floating overlay, e.g., the overlay network’s topology itself changes. The safehouses which are used to access the service (safehouse traffic is routed to the service over the overlay network) can be discovered at runtime, although they don’t go into details about a particular implementation of this would work – their position seems to be that they just want to create the framework to give customers more flexibility in how they deal with DDoS.

Also, at the time of the paper, they were in the process of constructing a prototype system. For all of these reasons, I don’t think this is a good choice to compare our results to.

OverDoSe, I feel, is overly complicated.

So, I know you said to mention only one, but I wanted to mention several and give my thoughts on why we might like to compare them.