Abstract—Measuring round trip time (RTT) in a hostile network is an unsolved problem in distributed systems engineering. Accurate RTT measurements are important for distributed systems in which timing packet arrival is useful, such as in a decentralized exchange (DEX) with fairness guarantees. In this paper we present a new RTT measurement algorithm for securely measuring RTT to an untrusted server. We measure the RTT indirectly in order to prevent tampering since a malicious node operator can arbitrarily change any RTT measurement that interacts with his machine. The algorithm approximates the RTT to a node by measuring the RTT to the node that serves the /24 subnet containing the target node’s IP address. This new RTT measurement algorithm provides an accurate, practical, and generic solution for collecting network latency data in a hostile network environment.

Index Terms—decentralized exchange, front-running, network latency measurement, peer-to-peer computing, tamper resistance

I. INTRODUCTION

ODIN is a new algorithm that can be used to measure RTT to an untrusted server. Existing RTT measurement solutions interact directly with the target server and are therefore vulnerable to tampering. There do not exist any prior algorithms for measuring RTT in a hostile environment.

In this paper, we describe how to perform indirect assessments of the target’s RTT to generate an approximate RTT that is accurate enough for use in production systems. The ODIN algorithm leverages structural properties of internet routing and IP address block allocation patterns to produce a close estimate. The critical insight in this paper is that the RTT to a node at an IP address can be measured with consistent accuracy without ever interacting with the node or its gateway. We show that pings, the naive way to measure RTT, are easy to deceive and are therefore not suitable for use in systems that require high-fidelity RTT measurements.

We outline the design of a RTT measurement algorithm, called ODIN, with the following goals:

- Tamper-Resistance: Ensure that the target of the RTT measurement cannot influence the RTT estimate.
- Accuracy: Provide high-quality RTT estimates.
- Generality: Define the algorithm as a generic RTT estimation algorithm that is independent of any specific distributed system implementation.

ODIN achieves these goals through platform independent trustless RTT approximation. By trustless approximation we mean that we generate an estimate of RTT to the target based on RTT to a node that is located near the target in the network. ODIN is designed to be modular so that the system designer can easily replace a ping-based RTT estimate with ODIN. The algorithm is simple and easy to implement, especially when it is built on top of the UNIX Traceroute program.

ODIN was designed to prevent front-running in a decentralized exchange. Front-running is a method used by high-frequency traders to make trades using non-public information. Front-running can occur in a DEX when a node in the network receives information about available trades before other nodes receive the information. The simplest way to prevent front-running is to ensure that all nodes receive the information at the same time by introducing a RTT-dependent delay when sending information to multiple nodes. THOR [1], an algorithm that measures pings to centralized exchanges and delays orders such that the orders arrive at the same time, is an example of RTT measurements being used to prevent front-running.

Preventing front-running in a DEX is a more complex problem than preventing front-running on a centralized exchange. The added complexity comes from the front-runner’s ability to arbitrarily modify the code of the exchange that receives the orders. This means that all direct measurements of the ping latency can be circumvented, e.g. if an ICMP ping is used then a malicious node operator could install a custom server for ICMP packets that artificially delays sending a response in order to increase perceived network latency. This would give the malicious node preferential treatment by an algorithm that relies on ICMP pings to measure RTT for ensuring that messages reach all peers at the same time.

We have implemented ODIN as part of xud, Exchange Union’s decentralized cryptocurrency exchange [2]. We envision that other distributed systems builders will benefit from having the option of using the ODIN algorithm instead of pings for trustless RTT measurements. The ODIN algorithm is sufficiently general to be implementable in any language for any type of networked system.

Overall, this paper describes the design of a secure RTT estimation algorithm and makes the following contributions. First, a malicious actor cannot deceive the RTT estimation algorithm by modifying the node that is being assessed or via DDoS of the gateway router. Second, ODIN is modular, platform-independent, and simple to implement. Finally, the ODIN algorithm is the first RTT estimation algorithm designed for distributed systems operating in hostile environments and therefore enables the development of new kinds of distributed systems.

The rest of this paper is organized as follows. Section 2 provides a broad overview of our approach and describes the tamper-resistance components of ODIN in detail. Section 3 describes our implementation of ODIN on a DEX. Section 4 presents the results of assessing ODIN’s accuracy for a random set of IP addresses. Section 5 surveys related systems. Section 6 describes future work and Section 7 summarizes our contributions.

II. THE ODIN ALGORITHM

ODIN is a generic algorithm for estimating RTT that operates on top of any Traceroute implementation. Traceroute sends sequential probes to discover the path to an IP address. The central
observation of ODIN is that an adversary cannot modify his server’s location in the IP address space without controlling an Autonomous System [3] or an entire /24 subnet, and that neither of these resources are available for use by individuals [4].

This reality of the modern internet makes ODIN suitable for a variety of RTT measurement purposes, and we will use the example of preventing front-running in a DEX to demonstrate how ODIN prevents a dedicated and well-funded adversary from being able to deceive the RTT estimate.

The threat model we mitigate against is a high-frequency trader who co-locates his node with another node in the network and/or arbitrarily modifies his server to appear to have different latencies at different times in order to deceive the front-running prevention mechanism. The malicious trader is profit motivated.

For someone running a node in the same datacenter as another node, the RTT could be as low as 0.5ms [5]. For a node located on the opposite side of the earth, the RTT can exceed 150ms. A high-frequency trader who attempts to front-run should meet overwhelming time and/or cost obstacles to making a profit, but an honest user who happens to run a server far away from the majority of nodes should not be penalised.

All measurements that interact directly with a potentially malicious node can be deceived by artificially delayed response times. Whether it is by slowing an application’s response to ping packets, or by modifying the kernel’s network driver to respond slowly to new connections, there is always a way to deceive a measurement probe as long as the adversary has root or physical access to the server [6].

In order to prevent the adversary from being able to deceive the assessment algorithm, we measure the RTT to a node that with high probability routes packets to the node’s IP address. We find this node by using Traceroute to an IP address in the same /24 subnet as the target. This algorithm produces an approximate RTT that is accurate enough for production use.

A. Sequential probing is the only way to discover the path

Routing in the internet is stateless, i.e. a request’s path is not normally recorded anywhere. Sequential probes along the path is the only way to discover the IP address of a node that serves the target’s /24 subnet without using external aids. Traceroute [7] discovers this path by sending a series of probes addressed to a target IP address. The first probe has the Time To Live (TTL) field set to 1, the next to 2, etc. such that each router on the path to the target sends a response rejecting the probe due to its expired TTL field. The TTL field is used this way in order to prevent the request from being forwarded to the target, and instead elicits a series of responses that describes the route to the target.

The protocol most commonly used by Traceroute is ICMP. These can be sent without super user privileges, and most routers will respond to ICMP requests. The existing implementation of the ODIN algorithm uses Traceroute’s TCP syn mode such that a half-open TCP handshake is used instead of ICMP pings. This modification allows for more accurate data collection since TCP syn probes are less likely to be stopped by firewalls [7].

B. We measure RTT to your neighbor, not to you

An adversary could deceive the route discovery algorithm by sending forged IP packets claiming to be an intermediary between the penultimate node and the target node. In order to secure ODIN from this type of deception we randomize the last octet of the target’s IP address and trace to this address instead. The RTT to an IP address in the same class C network as the target’s IP will usually be very similar to the RTT to the target since class C networks are almost always managed by a single provider. This means that the adversary would need to own an entire class C network, i.e. all 256 addresses in the same /24 subnet, in order to reliably deceive the measurement. Only large corporations and governments are allowed to own class C networks [4]. The tamper-resistance guarantee of ODIN relies on the adversary not being able to arbitrarily control a class C network for the sole purpose of deceiving the RTT measurement.

C. Random assessment intervals

The adversary must be prevented from knowing when the next assessment will take place since the adversary could easily, and at low cost, spawn many short-lived servers at the time of the assessment to do a statistical attack on the assessment algorithm in order to evade the defense described in section II.B. Or, the adversary could DDoS the gateway at the exact time of the assessment to artificially slow the response times. A very short-running DDoS would probably evade detection and/or punitive action. The assessment interval is set using a cryptographically secure pseudorandom number generator to prevent the adversary from predicting the time of the assessment.

D. Incremental increase, immediate decrease

The ODIN algorithm is resilient against botnet DDoS attacks via strategic adjustment of each peer’s RTT estimate. We consider the scenario of an adversary Alice who uses a botnet to DDoS the router that routes to the /24 subnet of her front-running xud server, such that the router’s performance suffers. Bob’s xud server, a peer of Alice’s server, will perceive a RTT to her server that is $\epsilon$ seconds slower than the actual RTT $R$. Alice now commands the botnet to stop the DDoS attack on the router. Bob will believe that the RTT time to Alice’s server is still $R + \epsilon$ for $T$ seconds. $T$ is the number of seconds until Bob’s server performs another assessment of the RTT to Alice. In the context of a DEX, Alice gains $T$ seconds during which Bob’s server sends orders to her server $\epsilon$ seconds faster than to his other peers. This means that Alice can front-run these orders and make a profit. Without preventative measures Alice can repeat this process every $2T$ seconds such that she has the opportunity to front-run $\frac{1}{2}$ all orders she receives.

In order to mitigate this type of attack, we do as follows: when the RTT assessment shows that the RTT to a peer has increased, we increment the peer’s RTT by a fixed rate $\delta$. When the RTT assessment shows that the RTT has decreased, we update
immediately to the new smaller value. This means that Alice must run her DDoS attack for $\frac{T}{2}$ seconds in order to receive preferential treatment from peers, and as soon as she stops the attack she still has only $T$ seconds to take advantage of her deception. If $\delta$ is small enough Alice will not be able to profit from front-running since the cost of performing the DDoS attack will exceed the possible profit from front-running.

This feature of the ODIN algorithm is relevant only for systems that perform repeated assessments to the same node.

III. IMPLEMENTATION

ODIN is a general RTT measurement algorithm that can be used in any distributed system. We have implemented ODIN as a front-running prevention feature \([8]\) of xud, Exchange Union’s open source decentralized trading client. The front-running prevention system is easily enabled/disabled via a boolean in the configuration to support both traders who want optimal prices via front-running protection and those who want optimal trade completion speed.

All nodes in store the following state:

- $rtt$: Estimated round trip time. Initial value should be near 0.5ms to mitigate abuse via datacenter co-location [5].
- $\delta$: The fixed rate of incremental increase of the RTT estimate
- $MAX\_INTERVAL$: Upper bound on the amount of time between RTT measurements.

The algorithm for estimating RTT to the target is as follows:

1. Replace last octet of target IP address with a random value
2. Find the RTT to the last node that routes to the target
3. Randomize the assessment interval so the time of the next latency assessment cannot be predicted

The following pseudocode functions describe the algorithm in more detail:

Algorithm 1 Helper Functions

1: function TRACEROUTE(addr)
   ▷ Uses UNIX Traceroute command to perform trace to IP address. Returns list of nodes in path and latencies of each node in path
2: function CRYPTO\_RANDOM(max_value)
   ▷ Generates a cryptographically secure pseudorandom number between 0 and max_value
3: function RUN\_IN\_FUTURE(f, arg, t)
   ▷ Schedules function f to run with argument arg in t seconds

Algorithm 2 Estimate RTT

1: function ODIN(addr) ▷ Estimates RTT to the IP address
2: octets = addr.split(‘.’)
3: octets[3] = CRYPTO\_RANDOM(255)
4: addr = ‘.’.join(octets)
5: trace_lines = TRACEROUTE(addr)
6: trace_lines = trace_lines.reverse()
7: for ip, time in trace_lines do
   8: if time then
      9: if ip $\neq$ addr then
         10: if time $<$ rtt then rtt := time
         11: else if time $>$ rtt then rtt := rtt + $\delta$
      
   12: wait = CRYPTO\_RANDOM(MAX\_INTERVAL)
13: RUN\_IN\_FUTURE(ODIN, addr, wait)

IV. EVALUATION

In this section, we evaluate the accuracy of the ODIN RTT measurement system. We examine ODIN’s RTT estimates for randomly generated IP addresses and show that ODIN can accurately and precisely estimate RTT. Additionally, we show that the fundamental assumptions of the ODIN algorithm are correct.

A. Setup

We analysed ODIN using 1000 randomly generated reachable IP addresses. We determined whether or not an IP address was reachable by performing a trace to the target and then verifying whether or not the target was reached. Upon success of the the trace, we performed a trace to a random IP address in the same /24 subnet. We chose this new IP address by setting the last octet of the original target IP address to a random number between 0 and 255. We recorded the RTT to both IP addresses and the RTT to each of their penultimate nodes.

We sent three probes to each node on the path to the target IP address. This generated at most 3 RTT values for the penultimate node and the target. In cases where the randomized IP address was not reachable we simply stored the RTT data for the last node discovered by the trace since this was necessarily the node responsible for the unreachable IP address’ /24 subnet [9]. In total we collected 4373 RTT measurements across 1580 IP addresses.

B. RTT to the penultimate node

We assess the difference between the RTT to the target node and the RTT to its penultimate node. We expect a small negative percentage change, i.e. that the RTT to the penultimate node will be slightly smaller than the RTT to the target. The data shows that this intuition is correct. The average percent difference between a target IP address’s penultimate node and the target itself is -9.6%.

In 33.4% of the samples, the RTT to the penultimate node was actually greater than the RTT to the target. This result is likely due to caching, load balancers, improperly configured networks, or even DDoS protection systems.

C. Interchangeable penultimate node for all IP addresses in the same /24 subnet

We assess how close the latency of the random target’s penultimate node is to the latency of the real target’s penultimate
node. The RTT to the random IP address’ penultimate node is on average 2.6% less than the RTT to the target’s penultimate node. We verified that this pattern is caused by the ordering of the assessments, i.e. the second RTT assessment to the same /24 subnet within a short timeframe was 20% more likely to be faster than the first assessment. This phenomenon is likely due to caching by routers along the route to the targeted /24 subnet.

When we control for the ordering of probes, which is reasonable given that each iteration of our data collection script performed two traces while the ODIN algorithm performs only one trace, the error percent drops to around 0.5%. This very low error rate indicates that there is no decrease in accuracy due to the ODIN algorithm’s randomization of the RTT measurement’s target.

### D. Total Error

Finally, we graph the total error in ODIN’s RTT estimates. The average percent error of the estimate was -10.8%. Knowing this average enables automatic adjustment of ODIN’s RTT estimates. Overall, the percent difference of ODIN’s RTT estimate from the target’s true RTT is consistent and small enough for reliable use in production systems.

### E. Summary

In this section, we have evaluated the accuracy of the ODIN RTT estimation algorithm and its foundational assumptions. Our evaluation indicates that ODIN achieves accurate estimates of RTT with consistent error percentages relative to the actual RTT. The foundational assumptions of the ODIN algorithm, i.e. that the penultimate node of a random member of the same /24 subnet provides an accurate indicator of the target’s RTT, are supported by the data. ODIN provides a generic, tamper-resistant, and effective alternative to pings for measuring RTT.

### V. RELATED WORK

Ping is the UNIX program that is typically used to measure RTT. Ping uses an ICMP request to elicit a response from a host or gateway [10]. Ping requires the presence of an ICMP server on the target and is not tamper-resistant.

THOR is an algorithm implemented by RBC Capital Markets [1] that measures pings to centralized exchanges and sends orders with intentional delays to prevent front-running of the orders by competing clients. THOR is the inspiration for ODIN, although the ODIN algorithm differs greatly due to the system architecture differences between centralized and decentralized exchanges.

### VI. FUTURE WORK

The ODIN algorithm is already implemented as part of the xud decentralized exchange. Future development of the algorithm will include automatic tuning of the $\delta$ variable since its optimal value is dependent on the financial profit potential of deceiving the RTT measurement system to front-run orders.

### VII. CONCLUSION

This paper has demonstrated that it is possible to securely estimate RTT to any IP address. We believe the ODIN algorithm is a valuable addition to any distributed systems engineer’s toolkit.

### REFERENCES

[1] RBC Capital Markets. Electronic Execution - THOR. www.rbccm.com/thor/file-549899.pdf.
[2] Exchange Union: An Open-Source Project Connecting Digital Asset Exchanges. www.exchangeunion.com/static/files/ExchangeUnion-WhitePaper_en.pdf, 2019.
[3] Z. Zhang, Y. Zhang, C. Hu, and M. Mao. “Practical Defenses Against BGP Prefix Hijacking.” https://web.eecs.uchicago.edu/~zymao/Papers/connectDefendHijack07.pdf, 2007.
[4] “Can I Obtain a Class C Address Block?” Can I Obtain a Class C Address Block? - RIPE Network Coordination Centre, www.ripe.net/manage-ips-and-asns/resource-management/faq/faq-general-resources/can-i-obtain-a-class-c-address-block.
[5] Dean, Jeff. "Designs, Lessons and Advice from Building Large Distributed Systems." www.cs.cornell.edu/projects/ladis2009/talks/dean-keynote-ladis2009.pdf, 2009
[6] Karam, Ghassan, and Srdjan Capkun. On the Security of End-to-End Network Measurements. ETH Zurich, pdfs.semanticscholar.org/67a7/0b1076100ce6e80c6e222b5b0f91c0f141b664.pdf, 2014.
[7] Traceroute(8) - Linux Man Page, linux.die.net/man/8/traceroute.
[8] ExchangeUnion. "Feat/Thor by Hatmer Pull Request 1187 ExchangeU-nion/Xud." GitHub, github.com/ExchangeUnion/xud/pull/1187.
[9] "Classless Inter-Domain Routing (CIDR): an Address Assignment and Aggregation Strategy." IETF Tools, tools.ietf.org/html/rfc1519.
[10] Ping(8) - Linux Man Page, linux.die.net/man/8/ping.