PLOS ONE

RESEARCH ARTICLE

Addressless: A new internet server model to prevent network scanning

Shanshan Hao[1,2], Renjie Liu[1,2], Zhe Weng[1,2], Deliang Chang[1,2], Congxiao Bao[1], Xing Li[1,2]*

1 Institute for Network Sciences and Cyberspace, Tsinghua University, Beijing, China, **2** Department of Electronic Engineering, Tsinghua University, Beijing, China

* xing@cernet.edu.cn

Abstract

Eliminating unnecessary exposure is a principle of server security. The huge IPv6 address space enhances security by making scanning infeasible, however, with recent advances of IPv6 scanning technologies, network scanning is again threatening server security. In this paper, we propose a new model named addressless server, which separates the server into an entrance module and a main service module, and assigns an IPv6 prefix instead of an IPv6 address to the main service module. The entrance module generates a legitimate IPv6 address under this prefix by encrypting the client address, so that the client can access the main server on a destination address that is different in each connection. In this way, the model provides isolation to the main server, prevents network scanning, and minimizes exposure. Moreover it provides a novel framework that supports flexible load balancing, high-availability, and other desirable features. The model is simple and does not require any modification to the client or the network. We implement a prototype and experiments show that our model can prevent the main server from being scanned at a slight performance cost.

Citation: Hao S, Liu R, Weng Z, Chang D, Bao C, Li X (2021) Addressless: A new internet server model to prevent network scanning. PLoS ONE 16(2): e0246293. https://doi.org/10.1371/journal.pone.0246293

Editor: Hua Wang, Victoria University, AUSTRALIA

Received: June 4, 2020

Accepted: January 18, 2021

Published: February 2, 2021

Data Availability Statement: All relevant data are within the manuscript and its Supporting information files.

Funding: The author(s) received no specific funding for this work.

Competing interests: The authors have declared that no competing interests exist.

Introduction

Exhaustion of IPv4 addresses has long been recognized and is now a reality. IPv6 [1] was proposed in 1995 to solve this problem. The main improvement is the 128-bit address over the 32-bit IPv4 address, together with other goals like end-to-end feature and better security. As of March 2020, about 25% information resources (websites, emails, etc.), 60% DNS servers, and 30% Internet clients support IPv6 [2]. With the rise of the Internet of Things, 5G, and cloud computing, it is predicted that more than 75 billion devices will be connected to the Internet by 2025 [3], while there are only 4 billion IPv4 addresses in total. An inevitable and faster adoption of IPv6 can be expected. IT giants such as Facebook [4] and Microsoft [5] have been moving to an IPv6-only internal network. An IAB (Internet Architecture Board) statement expected that the IETF would stop requiring IPv4 compatibility in new or extended protocols, and future work would optimize for and depend on IPv6 [6]. Pure IPv6 is the future of the Internet.

IPv6 is developed with security in mind, realizing that security as well as privacy has always been one of the biggest threats to the Internet. It is natural to start with utilizing the massive address space of IPv6 to enhance security and privacy, considering that this is its biggest difference from IPv4.

By making network scanning difficult, this massive address space of IPv6 has naturally provided preliminary protection. The IP address is the identifier and locator of the Internet. Thus network scanning is usually the first phase of an attack to obtain the IP addresses of potential victims. This is easy in IPv4. The scanning time of the entire IPv4 address space is only about 45 minutes [7]. However, it takes more than 100 quintillion years to scan the entire IPv6 address space with the same efficiency. The massive address space of IPv6 helps a lot to hide the address of a device.

However, on one hand, NAT in the IPv4 era, albeit not designed for security, brings the byproduct of hiding devices and topology of the network from the outside. NAT is stateful and violates the end-to-end principle, thus deprecated by IPv6 designers. Many network administrators, however, are so used to the invisibility provided by NAT that they feel that NAT-less IPv6 rather poses a threat of exposure, despite the difficulty of scanning IPv6's huge address space.

On the other hand, recent advances of IPv6 scanning technology have threatened the preliminary invisibility provided by the huge address space. Various approaches have been proposed to scan the IPv6 Internet more efficiently mainly in two ways: collecting active IPv6 address records [8–15], and using statistical and machine learning methods to generate hitlists [16–21]. Most of these approaches are server-specific. Scanning IPv6 clients remains difficult, which brings many security benefits for the clients.

We cannot help but think, *can we further utilize the IPv6 address space to make IPv6 servers unscannable as well, avoiding as much unnecessary exposure as possible, thereby allowing servers to enjoy the security advantages brought by invisibility, but not violating end-to-end reachability?*

There have been attempts to enhance the anti-scanning feature of the IP addresses, mainly by: 1) generating addresses that is semantically opaque and more random [22], 2) using temporal [23, 24] or hopping [25–30] addresses with short lifetime, and further 3) using a unique address for each connection [31–35] or even packet [36, 37].

The former two approaches are basically incremental improvements. Each device still has one IP at a time which can be scanned. The per-connection address is an interesting idea but previous models are very complex, unscalable, and hard to deploy. Specifically, they largely remain within the framework of dynamically shared address pools. Address mapping and address collision become problematic, the network needs to be modified to enable routing, and the system is complicated. Moreover, few works are about servers, and generally require synchronous cooperation of the client-side.

On the other hand, the current network security model relies on encryption. For instance, SEND [38], TLS [39], and DNSSEC [40] are used at the data link layer, the transport layer, and the application layer, respectively. At the network layer, IPsec [41] encrypts the payload of IP packets but the (outer) IP addresses are left exposed. Since TCP/IP was introduced, the creators of the Internet have considered introducing encryption into the IP address itself. This is an extravagant dream in the era of IPv4 since addresses are scarce and reused with great care. However, this is changed in IPv6, and the 128-bit address provides sufficient space to carry the encrypted information.

Previous work [35–37] propose to encrypt host identity into pseudo-random temporal addresses, so that the host identity is hidden from the outside network. But it has to be done on local routers, otherwise routing will fail. The network needs to be modified and it is sort of

encapsulation. CGA [42] applies hashing in auto-generation of addresses, but it aims to solve link-local address spoofing and again the addresses are exposed. The literature has not seen a simple and scalable mechanism to introduce encryption into the IP address itself.

In this paper, a model named addressless public server is proposed. We creatively use the prefix delegation mechanism [43] in a way that is different from its original intention. So that we introduce encryption into the per-connection address in a very simple way and no modification of other participants is needed. And we make it usable for public servers to make them difficult to be scanned. Not only is security strengthened in this way, but our model also offers a novel architecture to enable flexible high-availability and other features.

The addressless sever has two modules, one module provides independent entrance of the server, the other provides the main services. When receiving access requests, the entrance module generates a destination address using encryption, and redirects the request to the generated destination address. This destination address is different for each connection request. The main service module is allocated a prefix instead of an IPv6 address and listens on all the addresses under the prefix. When the main service module receives a data flow, it conducts verification on the destination address, and only responds to flows that pass the verification. Scanning traffic and attackers that visit the main service module directly cannot pass the verification, therefore will be immediately dropped. In this way, we make the main server imperceptible.

This model separates the network entrance and provides isolation for the main server. It allocates a prefix instead of an address to the main server, so that it no longer has one address at a time, thus imperceptible by the outside network. At the same time, our model naturally supports flexible high-availability solutions such as lightweight load balancing, active-active cluster, and CDN. More security and functional mechanisms can be developed based on it. All in all, our model provides a novel perspective on various problems faced by public servers.

Background and related work

Our model uses the massive IPv6 address space to prevent the server from being scanned. So in this section, we first introduce network scanning and IPv6 address space security. Then we introduce prefix delegation, the address configuration mechanism that our model built upon. Finally, we make a detailed analysis of previous work in using IPv6 address space to enhance security, and compare our model to the most related ones.

Network scanning

Network scanning is a technology to collect the active addresses by sending packets to a huge set of addresses. Early scanning techniques like Nmap [44] often take days to scan the entire IPv4 address space. Zmap [7] proposed in 2013 reduces the scanning time of the entire IPv4 address space to about 45 minutes. This makes the cost of scanning under IPv4 negligible.

Due to the massive address space, network scanning in IPv6 has always been considered impossible. Although Zmapv6 [45] is proposed to scan IPv6 addresses, it is only a scanning tool that uses the same technique as Zmap without improving scanning efficiency. However, a series of techniques has been proposed in recent years to reduce the difficulty of IPv6 scanning.

The work on IPv6 scanning can be divided into two types. One is to generate hitlists by collecting active addresses on the Internet. Various sources can be used to collect active addresses. Fiebig et al. [8] propose to generate the hitlist using DNS data, and Borgolte et al. [9] propose to generate the hitlist using DNSSEC-signed reverse zones, while rDNS data is used by Fiebig et al. [10]. These researches mainly focus on generating the hitlists of Internet servers, while

Beverly et al. [11] and Rohrer et al. [12] introduce approaches to collect router addresses, and Rye et al. [13] probe and collect addresses of last-hop routers using traceroutes. Gasser et al. [14] summarize these methods and give a large hitlist set, and publish a compiled, open-source, and frequently updated hitlist whose quality is enhanced using active measurements [15].

The other is to generate hitlists by predicting active addresses using statistical or machine learning algorithms. Foremski et al. [16] first propose that active IPv6 addresses can be predicted by statistical algorithms. They introduce Entropy/IP, an algorithm to generate hitlists using the Bayesian algorithm. Ullrich et al. [17] introduce a scanning algorithm based on pattern recognition, and Zuo et al. [18] analyze the active addresses and make predictions through association rule learning. Murdock et al. [19] propose 6Gen, an algorithm that uses some active addresses as seeds to generate hitlists. Liu et al. [20] introduce 6Tree, which uses hierarchical clustering to predict addresses and generate hitlists. Deep learning methods have also been introduced. Cui et al. [21] stack gated convolutional network to encode address structure and generate hitlists.

IPv6 address space security

Since the birth of IPv6, researchers have been looking for ways to increase the security and privacy of IPv6 addresses.

In RFC 4291 [46], an IPv6 address is divided into two parts: a 64-bit prefix and a 64-bit identifier interface (IID). The most widely used IPv6 address configuration methods are SLAAC [47] and DHCPv6 [43]. In SLAAC, the IID is generated from the MAC address of the device using the EUI-64 algorithm [48]. This makes the IID part of the IPv6 address remaining constant in the lifetime of a device. In DHCPv6, the rules of address configuration are determined by the network administrator. In the early days, these rules are simple and regular, which make the addresses used in the network showing obvious patterns. Scanning is quite easy for both SLAAC and DHCPv6 addresses, thus posing a threat to security.

To solve this problem, SLAAC Privacy Extension (SLAAC PE) is proposed in RFC 4941 [23]. In RFC 4941, the client is recommended to use a temporary address to communicate with servers. The IID of the temporary address is one-time, generated by applying the MD5 algorithm [49] to the previous IID. In this way, the security and privacy of the client are enhanced in SLAAC. DHCPv6 also proposes an approach of allocating temporary addresses [50], but it is not widely used because it brings a series of problems [51].

More in-depth work is introduced on this basis. Semantically opaque IID is recommended to be used in SLAAC and DHCPv6 by RFC 7217 [22], RFC 7493 [52], and RFC 8065 [53], etc. RFC 7707 [24] recommends to reduce the lifetime and increase the randomness of IPv6 addresses used by network devices based on a summary of scanning algorithms.

At the same time, researchers also put forward some work on the measurement of active IPv6 address space. Plonka et al. [54] analyze the temporal and spatial characteristics of active IPv6 addresses. Li et al. [55] describe the distribution characteristics of Internet IPv6 prefixes. These works demonstrate how the IPv6 addresses are actually used thus objectively shows the security status of the IPv6 address space.

There are also a few works focused on the detection and defense of IPv6 scanning. Fukuda et al. [56] introduce an approach to detect IPv6 scanning and evaluate relevant severity. Plonka et al. [57] introduce kIP, a new approach to increase the anonymity of IPv6 addresses.

IPv6 prefix delegation

In our model, the main service module is assigned an IPv6 prefix. Assigning a prefix to a device is allowed in IPv6, typically using the DHCP-PD [43] mechanism. DHCP-PD is originally

proposed to allow a DHCP server to assign a prefix to a DHCP client, so that this DHCP client can further allocate the addresses under the prefix to other devices. In our model DHCP-PD is used for a different purpose. The main service module is assigned a prefix using DHCP-PD, but after that, it uses the addresses under the prefix itself, instead of allocating them to others.

Besides DHCP-PD, prefix delegation is also used in other scenarios. RFC 8273 [58] allows each device in the same subnet to be configured with a unique IPv6 prefix, so that they are logically under different subnets and cannot send packets to each other except through the first-hop router. Using this, isolation is provided between the devices in a shared-access network. However, each device only uses one fixed address under the prefix. To the best of our knowledge, the literature has not made deeper use of the prefix allocated to a device.

Using IPv6 address space to enhance security and privacy

IP address hopping. The technique to dynamically and frequently change the IP address of a device has long been used to prevent attackers from finding the target since the era of IPv4. In the IPv4 era, it can only be achieved with the help of the network service, such as DHCP servers [59] and SDN [29, 30]. In the IPv6 era, the massive address space and SLAAC enable auto-configuration of dynamic addresses [26]. IP address hopping of a server is harder, since its address needs to be known by clients to allow inbound traffic. In previous work, the sender has to update the addresses synchronously with the receiver based on a shared secret [25, 27, 28].

Per-connection address and beyond. IPv6 addressing model specifically supports assigning multiple IP addresses to a single interface [46]. Thus researchers have gone beyond address hopping to assign each connection (or a set of closely related connections) a unique address [31–35] to enhance privacy. Our model also uses per-connection addresses, and a detailed comparison is given later in this subsection.

The per-connection address is not the end of the road either. Researchers extend this idea temporally to propose per-packet address [36, 37], and spatially to propose prefix alteration [60–62]. The per-packet address means to use a unique address for each packet. However, the stronger privacy is achieved at the cost of higher complexity of demultiplexing packets to flows and modification of local networks to enable routing [36, 37]. Prefix alteration means not only the IID part but also the prefix part of a device's address can be varying. It can be achieved by prefix hopping, prefix bouquets, prefix sharing, and variable-length prefix [60, 61], or by exploiting mobile IPv6 [62].

Limitation of previous work. The idea of using address space to hide node identity stems from IP address hopping in the IPv4 era, which can only be achieved through a pool of dynamic addresses shared by a group of devices due to scarcity of addresses. Although previous works have applied the idea to IPv6 and extended it to per-connection or even per-packet address, in essence, they have not gone beyond dynamically shared addresses pool. That is, devices are still sharing a set of dynamic addresses, although this set becomes enormous; it contains all the addresses under the IPv6 prefix.

The scheme of dynamically shared addresses pool brings two problems. First, mapping an address to a device or a connection and the related routing are difficult. Early work is stateful, the mapping needs to be recorded [31, 33, 34]. More recent work achieve stateless mapping by encrypting the host identity into the one-time pseudo-random address. However, decryption needs to be done at the local router because the host identity needs to be extracted for local routing [35–37]. Modification of the network service makes it unlikely to be deployed.

Second, IPv6 enables self-generated pseudo-random addresses without relying on stateful DHCP servers. But making sharing of addresses stateless brings the possibility of address

collisions. Previous works skip this problem by stating that the possibility is negligible or use duplicate address detection to detect and avoid collisions [33, 34]. Some use external unique identifiers to generate addresses, such as Electronic Product Code [63] in IoT scenarios.

Further, few works are about servers. On one hand, it is complex for a server to maintain a huge set of addresses assigned to each of its connection under this dynamically shared addresses pool scheme. For instance, Sakurai et al. [34] use sliding windows to maintain active addresses. On the other hand, the contradiction between inbound traffic and dynamic addresses has not been solved. Sender and receiver need to cooperate and synchronously update a pseudo-random sequence of addresses calculated based on a secret [25, 27, 28, 34]. This is infeasible for public servers.

Comparison of our model. We innovatively utilize the prefix delegation mechanism [43]. Our model is built on the idea of the per-connection address. However, the device is assigned all the addresses under the prefix, thus the complexity of routing and address collision is completely eradicated, and modification of the network service is no longer necessary. And specifically for a server with a huge amount of connections, the maintenance of the active addresses is no longer a problem.

Prefix delegation also enables us to introduce encryption into the address at the endpoint. Previously it has to be done on local routers [35–37] otherwise the dynamically shared addresses cannot be routed properly. Together with a novel stateless salting algorithm, we achieve stateless mapping of the per-connection address to the client.

And to make the public server accessible at the per-connection addresses, we innovatively separate the Internet entrance module from the main service module, and use the entrance module to generate the encrypted address and redirect the client. In this way, no modification of the client-side is needed. The exposed entrance module bears no other logic and is simple, and the main service is isolated and hidden in the huge address space.

Design of addressless server

In this section, the design of addressless server is presented.

Design principles

The addressless server model is designed to prevent the server from being perceived and scanned by the attackers. To reach this goal, the model takes advantages of the following features:

1. Separate the Internet entrance module from the main service module, and provide isolation to the main service module.
2. Allocate a prefix instead of an address to the main service module.
3. Eliminate the one-to-one correspondence between the server and the IP address.
4. Introduce encryption into IPv6 Address to make use of the redundant IPv6 address space.

In this way, the model can provide a triple guarantee for the server: strip the Internet entrance from the main server, hide the legitimate address in the massive IPv6 address space, and use encryption to ensure that only the legitimate address generated by the entrance module can be visited. By decoupling the server and the IP address, attackers can no longer use the IP address as the identification of the server. This is the meaning of "addressless". Through this, the server is protected from being perceived and scanned by the outside devices, so that security is enhanced.

System design

We divide the server into two modules, the entrance module and the main service module. The entrance module has a fixed Internet address; the main service module is configured with a prefix and uses all the addresses under the prefix to communicate with clients. The topology of the addressless server is shown in Fig 1.

The main service module is directly connected to the first-hop router. The main service module and the first-hop router are both configured with a non-public IPv6 address. This address is used for prefix delegation and routing. It is usually a link-local address. If there are more management requirements, other private addresses such as ULA [64] can also be configured. The first-hop router routes all the packets whose destination addresses are under this prefix to the main service module. The entrance module is configured with a fixed IPv6 address. This address should be configured as an AAAA record in the DNS system.

When an Internet client initiates a connection with the server, it first sends a DNS query to the DNS server. The DNS server returns the entrance address to the client. Then the client sends the request to this address. After receiving the request, the entrance module uses the prefix of the main service module and the source address of the packet to calculate an IPv6 address through an encryption algorithm, then returns this address to the client. Finally, the client initiates connections with the main service module using this address as the destination address.

The calculation process can be formulated as the following equations. We denote the client address as SA, the encryption process as function $f()$ (the specific process of $f()$ is discussed in next subsection), and the prefix of the main service module as $prefix$, then the destination address is:

$$DA_{1:N} = prefix \tag{1}$$

$$DA_{N+1:128} = f(SA) \tag{2}$$

To achieve compatibility, clients that are agnostic of our model should be able to communicate with the public server without modification. So we use the redirection mechanism to

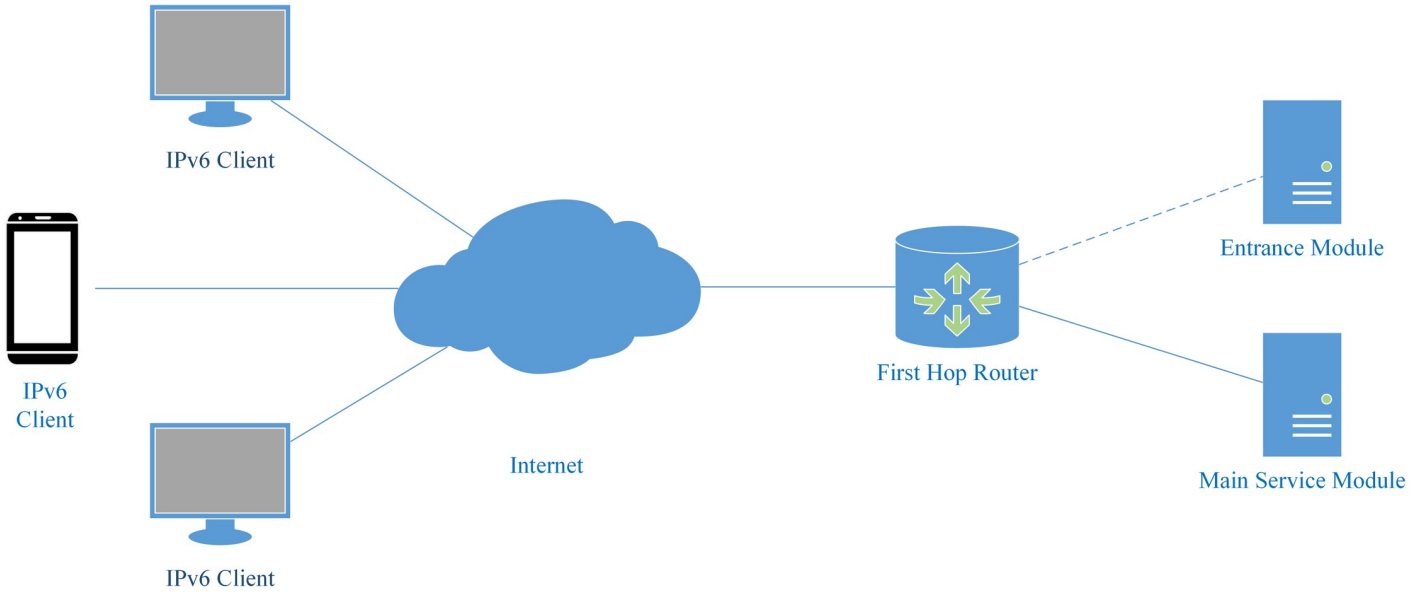

Fig 1. Topology of the addressless server. The server is separated into the entrance module and the main service module. The main service module is directly connected to the first-hop router. The dotted line indicates that the entrance module can be deployed anywhere on the Internet.

https://doi.org/10.1371/journal.pone.0246293.g001

achieve the process of entrance module returning the generated address and client connecting that address. In this case, the entrance module temporarily redirects the client's request to the generated address. After receiving this message, the client sends a new request to the main service module. When the main service module receives a flow, it verifies the destination address using the source address through the same encryption algorithm. The verification process can be described by Eq (3)

$$Res = g(SA, DA_{N+1:128}) \tag{3}$$

In Eq (3), $g()$ is the verification function. *Res* is a bool value. True means the flow passes the verification while False means the flow fails the verification. If the verification fails, the server discards the packets.

The mechanism is described in Fig 2.

Noted that the client here is just a common IPv6 client. It is configured with an IPv6 address, not a prefix. The address used in the whole connection lifetime should remain unchanged. Otherwise, it may cause the source address used in the encryption and the source address used in the decryption different, leading to the failure of the verification.

The verification process is performed in each flow. The main service module only needs to perform the verification once for each flow because the source address and destination address remain unchanged during the flow. Once the connection is established, the server can directly accept the following packets until the end of the flow. After a connection is terminated, the client should visit the address of the entrance module if it is going to initiate another connection, and the entrance module generates another address and redirects the request to this new address.

To prevent attackers from intercepting data flows and launching replay attacks, the generated address should be different each time. To reach this goal, we add a time-varying factor in the encryption process $f()$, which is described as salt. In this case, when the attacker intercepts the packets and forges an attack message using the source and destination address from those packets, it will not be effective because the legitimate destination address is different because the salt has changed.

In our model, the entrance module can be deployed anywhere on the Internet, logically and geographically. It can be configured on the same device as the main service module, and connected to the Internet through the same first-hop router. The entrance module can also be configured on different devices in the same subnet, or even a location far from the main server

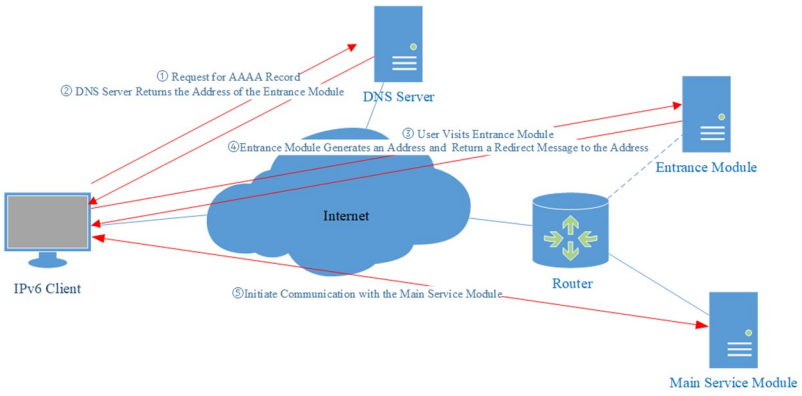

Fig 2. Mechanism of the addressless server. The communication process of how a client successfully initiates a flow to the server.

https://doi.org/10.1371/journal.pone.0246293.g002

on the Internet. The address allocated to the entrance module can be one of the addresses under the prefix delegated to the main service module, or it can be a totally independent global unicast address. Furthermore, considering that the entrance module only provides simple and reproducible services, it is easy to stack or distributed deploy the entrance modules. All in all, the configuration of the entrance module is very flexible. Various strategies can be used on the deployment of the entrance modules to achieve better results.

In the addressless server mechanism, the entrance module is responsible for calculating the destination address and returning a redirect message to the client. If the server needs a stricter strategy, the entrance module can also provide user authentication service, and only replies to the client who passes the authentication. This ensures that all the clients that can perceive the main service module are authenticated, which further ensures server security without affecting the simplicity.

From the above discussion, we can see that our model takes advantage of the redundant space of the IPv6 address by introducing encryption into IPv6 suffixes. The IPv6 space is very large, and only a small part is actually used in traditional scenarios. As a result, we can 'waste' the address space to prevent the server from being scanned. We assign a prefix to the server to free up the suffix space and let it carry the authentication information. In our model, the use of the prefix and the suffix of the destination address is different. The prefix is used for routing while the suffix is used for carrying authentication information to provide additional security benefits.

Encryption algorithm

The encryption in this paper is essentially a signature-verification process. When a client initiates communication, the entrance module first signs the source address, embeds the result into the destination address, and returns it to the client. Then the client initiates connections to that address, and the main service module conducts verification using the source address and the destination address. Since we do not need to restore the message, $f()$ here does not need to be reversible.

The encryption process $f()$ is described as follows:

$$H\_SA = Hash(SA) \tag{4}$$

$$P\_SA = \Phi(H\_SA, salt) \tag{5}$$

$$DA_{65-128} = e(P\_SA, key) \tag{6}$$

$$DA = strcat(prefix, DA_{65-128}) \tag{7}$$

And the verification process $g()$ is described as follows:

$$H\_SA = Hash(SA) \tag{8}$$

$$P\_SA = e^{-1}(DA_{65-128}, key) \tag{9}$$

$$Result = \Psi(P\_SA, H\_SA, salt) \tag{10}$$

In Eqs (4)–(10), SA is the client address (source address of the connection request); DA is the generated address under the prefix of the main service module (destination address of the connection request).

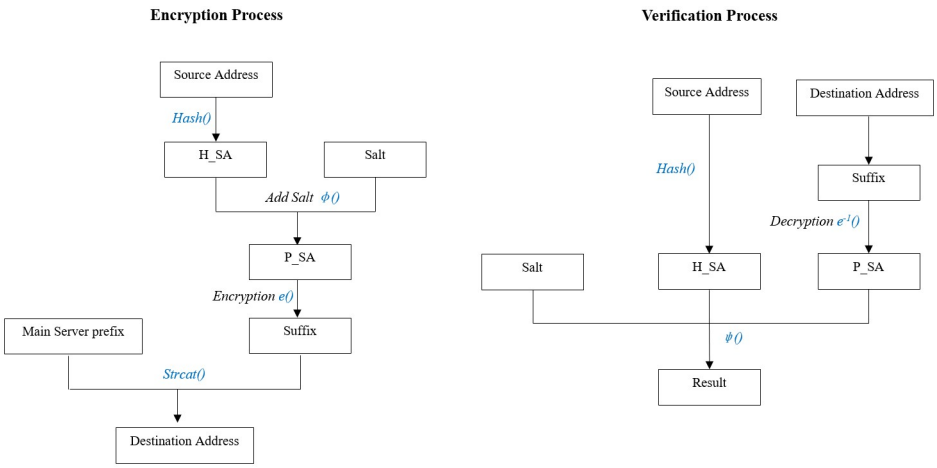

Fig 3. Encryption process and verification process.

https://doi.org/10.1371/journal.pone.0246293.g003

In Eqs (4) and (8), *Hash*() is a hash function to convert the source address into a 64-bit sequence. A hash function guarantees that the sequence is uniformly distributed in the entire range space. Any hash function is feasible here, including cryptographic hash functions such as md5 [49] and SHA-128 [65], or string hash functions such as DJB and BDKR. The string hash function is a better choice here because of the higher efficiency. DJB algorithm is used in our prototype.

In Eqs (5) and (10), *salt* is used as the time-varying factor. We add the salt in function Φ(), which is discussed in the next subsection. Function Ψ() is the verification function determined by Φ().

In Eqs (6) and (9), $e()$ is the encryption function, and $e^{-1}()$ is the decryption function. *key* is the encryption key.

Fig 3 shows the encryption and verification process briefly.

Theoretically, the encryption algorithm $e()$ can be arbitrary, as long as the ciphertext can be held in the suffix space in some way. However, we should consider it from two perspectives: security and efficiency. Considering security, a mainstream encryption algorithm should be used here. There are two categories of encryption algorithms: the symmetric encryption algorithm and the asymmetric encryption algorithm. The symmetric encryption algorithm has the following advantages: to achieve the same security level, it has shorter ciphertext length and key length. While the advantage of the asymmetric encryption algorithm is that it allows the public key not to be secret. In our model, the key is used only by the entrance module and the main server module, which are both under the control of the service owner. There is no need to distribute the public key, and it is not a challenge to keep the encryption key secret, thus there is no need for the asymmetric encryption algorithm. Considering efficiency, the symmetric encryption algorithm is also better for faster encryption and decryption processes. As a result, the symmetric encryption algorithm is a better choice in our model, which can ensure a faster encryption/decryption process and a higher level of security for a given ciphertext length.

The most widely used symmetric encryption algorithms are DES [66], 3DES [66], AES [67], etc. Considering the 64-bit length of the IPv6 suffix, to make the encryption easier, DES or 3DES is better here. Although DES is often considered insecure in the modern network environment, it can provide a sufficient level of security in our scenario (discussed in the Security Analysis Section) while it is much faster than 3DES. As a result, we use DES as $e()$ in our

prototype implementation. Nevertheless, our model does not place restrictions on exactly which encryption algorithm is used; that is up to the choice of the server owner.

Salting algorithm

We discuss the generation of the time-varying factor (salt) in this subsection. To make the generated addresses unpredictable and the replay attacks ineffective, adding salt is necessary in the address generation.

Stateful salt is the common choice in many salting scenarios. First, we consider if we can use stateful salt in our model. In this case, the entrance module and the main service module save the same state sequence. When an address is generated by the entrance module, it uses the current state as the salt, then hops to the next state. It is similar in the verification process in the main service module. However, it is not a good choice here because synchronization is a challenge. As shown in Fig 4, first client A visits the entrance module and obtains the redirect address $Address_1$ generated with salt i, later client B obtains the address $Address_2$ generated with salt i+1. But because of different delay, B visits the main service module earlier. At this time, the state in the main service module is still the salt i, thus the packets of B will fail verification and get wrongly dropped. This situation will happen frequently since a public server is generally visited simultaneously by a high volume of clients all over the world who face very different network conditions. Synchronization of states between the entrance module and the main server module thus become very challenging.

To solve this problem, we introduce a novel stateless salting algorithm. In this algorithm, the entrance module and the main service module do not save any state. Public information is used instead. System timestamp is a desirable choice among the various public information because it changes over time naturally and is extremely easy to obtain. We calculate the salt using the timestamp as equation Eq (11)

$$Salt = (SystemTime - T_0)/X \tag{11}$$

This equation is similar to the one-time key generation algorithm specified in RFC 6238 [68]. T_0 is the initial value and X is the step size. These two parameters are the same and kept secret in the entrance module and the main service module. Even if the attacker speculates

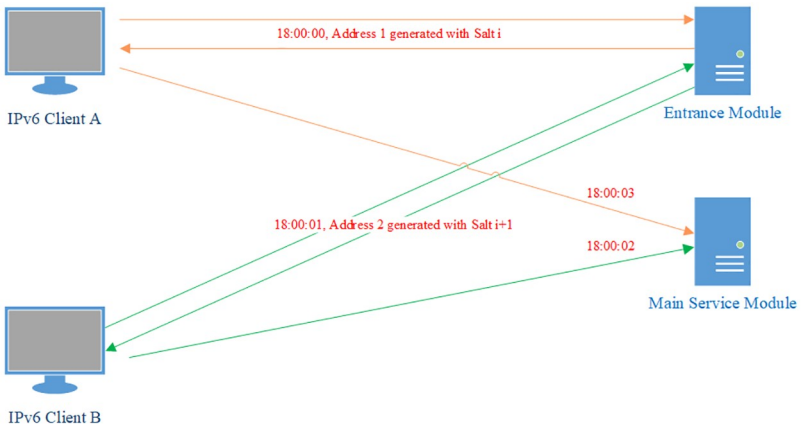

Fig 4. Stateful salt is NOT a good choice. Client B visits the entrance module later than client A and gets an address generated with salt i+1. Due to different delay, B visits the main service module earlier than A and fails verification because the main service module is expecting salt i.

https://doi.org/10.1371/journal.pone.0246293.g004

about the possible timestamps based on the current time, he cannot obtain the salt because of the confidentiality of T_0 and X, which further enhances the security of encryption.

The salting process is described by Eq (12)

$$P\_SA = XOR(Salt, H\_SA) \tag{12}$$

In Eq (12), P_SA is described in Eq (5).

XOR() is used as the function to add the salt in Eq (12). Here *XOR*() can be replaced by any operations, with the only requirements that: (1) the result should be different if the salt changes and (2) the operation is reversible. The complexity of *XOR*() is extremely low, so it is used as the salting function here. This does not introduce any additional security risks.

The server's verification process is described as the following equations:

$$P\_SA = e^{-1}(DA_{65-128})) \tag{13}$$

$$Salt = XOR(P\_SA, H\_SA) \tag{14}$$

$$T_s = Salt * X + T_0 \tag{15}$$

$$Result = (SystemTime - T_s) \in (0, threshold)?True : False \tag{16}$$

In the following, we discuss the value of *threshold* in Eq (16). The time required in general occasions and the error redundancy should be considered here. We assume that the timestamp used for encryption in the entrance module is T_{s1} and the system time for verification in the main service module is T_{s2}, then the time difference ΔT can be described by Eq (17):

$$\Delta T = T_{trans\_en} + T_{trans\_ma} + T_{pro\_cl} + T_{pro\_en} + T_{pro\_ma} + T_{syn} \tag{17}$$

In Eq (17):

T_{trans_en} is the transmission delay of the packet from the entrance module to the client;

T_{trans_ma} is the transmission delay from the client to the main service module;

T_{pro_cl} is the processing time of the client between receiving the redirect message and sending the new request to the main service module;

T_{pro_en} is the processing time of the entrance module from time stamping to sending the message;

T_{pro_ma} is the processing time of the main service module from receiving the message to verifying the timestamp.

T_{syn} is the system time difference between the main service module and the entrance module.

It is shown in Fig 5.

In Eq (17), T_{pro_en} and T_{pro_ma} are usually negligibly small. The entrance module and the main service module are both under the control of the server operator, thus the system time difference can be minimized and T_{syn} should also be negligible. In this case, Eq (17) is:

$$\Delta T = T_{trans\_en} + T_{trans\_ma} + T_{pro\_cl} \tag{18}$$

That is, the threshold should be at least greater than the sum of the delay from the entrance module to the client, the delay from the client to the main service module, and the time required by the client to process the redirect message. This usually varies from several milliseconds to several seconds. Considering redundancy, a threshold of about 10 seconds is generally

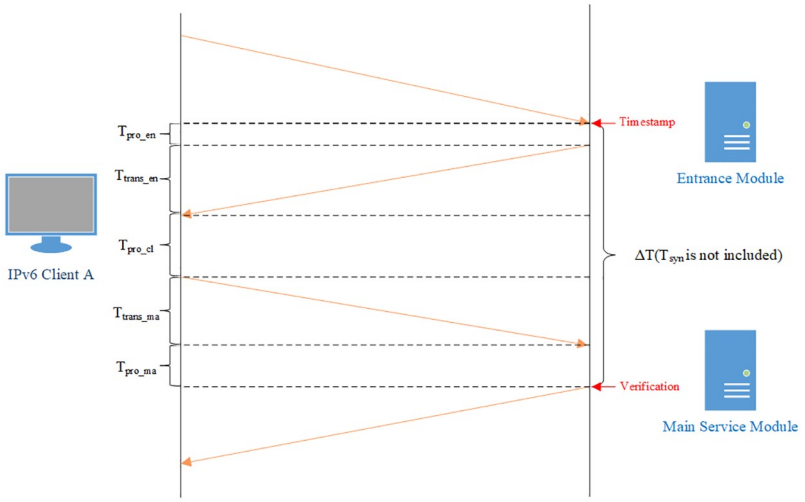

Fig 5. Time elapsed between timestamp of encryption and verification. It consists of two transmission delays (entrance module to client, and client to main service module) and three processing times (entrance module, client, and main service module). Note the system time difference T_{syn} is not included.

https://doi.org/10.1371/journal.pone.0246293.g005

appropriate. A larger threshold brings higher security risks, while a smaller threshold means less redundancy.

If the synchronization between the entrance module and the main service module is hard, which means T_{syn} in Eq (17) cannot be neglected, then the verification function Eq (16) should be modified as Eq (19):

$$Result = (SystemTime - T_s) \in (-threshold_1, threshold_2)?True : False \tag{19}$$

Here the $-threshold_1$ is negative, because the system time of the main service module when it performs verification can be earlier than the system time of the entrance module when it performs timestamp. Since the difference in system time synchronization is usually stable, system operators can adjust $threshold_1$ and $threshold_2$ accordingly.

However, considering that all modules are controlled by the server owner, time synchronization should not be a challenge. So on most occasions, we can regard T_{syn} negligible.

The salting algorithm is described in Fig 6.

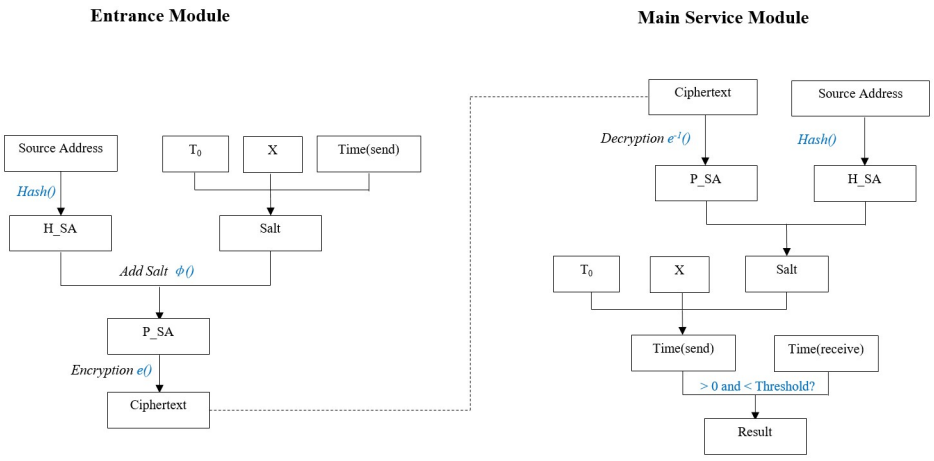

Fig 6. Salting process.

https://doi.org/10.1371/journal.pone.0246293.g006

Prefix length consideration

Generally speaking, according to RFC 4291 [46] and related RFCs, an IPv6 address has a 64-bit prefix and a 64-bit interface identification. Although this is not mandatory, it is consistent with our algorithm. In the above discussion, a 64-bit prefix is allocated to the server which is used for routing, and a 64-bit IID is used to carry the encrypted information.

However, our model does not require the prefix length to be 64-bit. Assigning a shorter prefix such as /56 is also feasible. The prefix space is used for routing, so in some cases, the server has to be allocated a longer prefix. For example, the server is built in a /64 subnet, and there are more than one devices in the subnet. In this case, a /68 or /72 prefix can be assigned.

A longer prefix means a shorter suffix, thus less space to carry the encryption information. The suffix length cannot be too short, otherwise, security will be jeopardized. As an extreme example, if the prefix is /120 and the suffix is only 8-bit long, then an attacker can trivially traverse the entire range space of the ciphertext and hit a legal address with only 256 trials. And some encryption algorithms need to be modified to shorten the length of the ciphertext when the suffix is shorter than 64-bit.

Further, the prefix length is a flexibly adjustable configuration in our model, because it is only known by the entrance module and the main service module. It is easy to cooperatively adjust the prefix length by the service operator when the network configuration changes.

Load balancing and high availability

In the above discussion, there is only one entrance module and one main service module by default. In fact, our model supports multiple main service modules and/or multiple entrance modules without any modification.

Multiple main service modules can be deployed smoothly and transparently. For example, in Fig 7, four main service modules are deployed to jointly serve the /64 address space. In this case, the entire server cluster shares a /64 prefix, but the prefix of each device may have any length longer than /64, and each can be of different lengths. The only requirement is that the prefixes of all the devices should cover the entire /64 address space, otherwise routing will fail and some packets will not be responded. Note that overlap is allowed, since the current routing rule of longest prefix matching will ensure proper routing. The existence of multiple devices, the number of the devices, and the respective prefix length of each device are all imperceptible

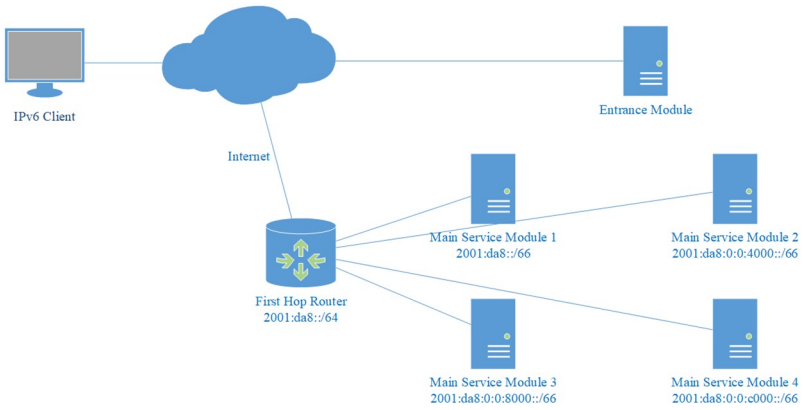

Fig 7. Multiple main service modules and random load balancing. In this example, four main service modules jointly serve the /64 prefix of the server. Because the generated address is uniformly distributed and random enough, the load is evenly distributed among them automatically.

https://doi.org/10.1371/journal.pone.0246293.g007

to the outside world, and cannot be obtained by any measurement approaches. This invisibility obviously helps to protect server security.

Similarly, multiple entrance modules can also be distributed in different territories and different ISP networks to optimize performance. Because the entrance module only provides the function of address generation which is very simple and stateless, it is even easier to be arbitrarily stacked in parallel. These entrance modules carry the same key and the same logic, therefore they generate the same address for the same client at the same time.

This support for multiple main service modules and/or multiple entrance modules goes beyond capacity expansion. Further, it provides a new and flexible framework to achieve load balancing and high availability related features.

Random load balancing. Our model achieve random load balancing automatically without any further modification or configuration. Because the addresses generated by the algorithm in the entrance module are uniformly distributed and random enough, the load is evenly distributed among the suffix space. If the devices all have the same prefix length, then the load is evenly distributed among them. For example in Fig 7, each device has a /66 prefix, so the load will be evenly distributed among these four devices. If the devices have different prefix lengths, then the load is distributed among the devices proportional to the size of the suffix space that is served by each of them. For example in Fig 7, if we delegate 2001:da8::/67, 2001:da8:0:0:2000::/67, 2001:da8:0:0:4000::/66, and 2001:da8:0:0:8000::/65 to the 4 devices respectively, then each will share 1/8, 1/8, 1/4, and 1/2 of the load respectively.

The advantages of the random load balancing feature in our model include:

1. Simplicity. Random load balancing in our model is completely stateless and requires no scheduling. The devices of the main service module can be arbitrarily stacked, and the load will be automatically distributed among them. To distribute the load in proportion to device capacity is easy. Operators only need to delegate different prefix lengths to the devices and configure routes accordingly.

2. Security. Our address generation algorithm is random. The configuration of devices of the main service module is completely imperceptible and transparent to the outside world. Thus attacks against load balancing become ineffective.

However random load balancing has the limitation of lack of control. The load is distributed completely random, thus cannot be controlled according to the status of the devices. Precisely, it is the number of requests instead of the load itself that is distributed, and it is statistically even instead of strictly even at any time. Resources consumed by different connections may vary greatly, and random allocation may not be absolutely uniform. While some devices are temporarily overloaded, others can be idle, resulting in a waste of resources and possible service failure on some devices.

Dynamic load balancing. To overcome the above limitation of random load balancing, our model supports dynamic load balancing through two schemes: routing configuration and entrance module strategy.

One way to achieve dynamic load balancing is based on routing, and all the devices of the main service module share one /64 prefix. The delegation of sub-prefixes to each device under this prefix and the related routes can be dynamically configured. In this way, the load of different devices can be adjusted in real-time to balance the utilization of each device. Caution that during the adjustment the sub-prefixes of all the devices should always cover the entire /64 address space.

High availability cluster can be naturally achieved in this scheme, such as active-active cluster and hot standby. When a device is detected to be failing, it can go offline simply by

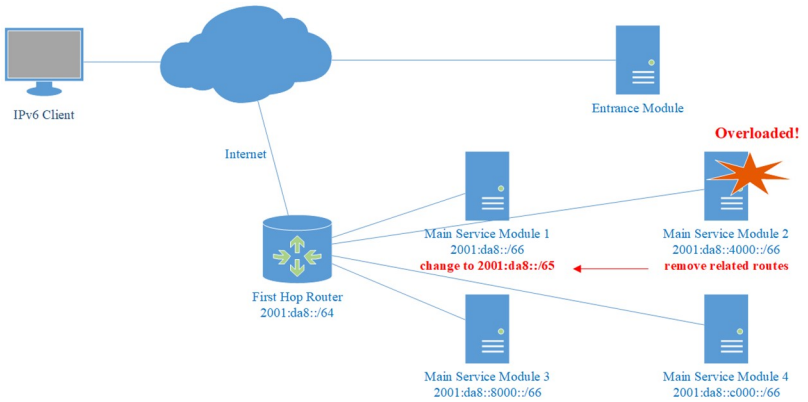

Fig 8. Active-active high availability cluster in dynamic load balancing based on routing. When a device fails, it can go offline by simply redistributing its related prefixes and routes to other devices.

https://doi.org/10.1371/journal.pone.0246293.g008

immediately redistributing its related prefixes and routes to other devices, so that the service will not be affected. A self-illustrated example is given in Fig 8.

Another way to achieve dynamic load balancing is based on the entrance module strategy, and each of the devices of the main service module is delegated one /64 prefix instead of sharing one /64 prefix. This means the entire server needs a shorter prefix. Nevertheless, for a server that needs dynamic load balancing, a /48 or even shorter prefix is not a problem [69]. In this case, the entrance module needs to add extra function when generating destination addresses. The 64-bit suffix is still generated as described above, but the 64-bit prefix is no longer the fixed server prefix. Instead, the prefix is selected among the prefixes of the cluster devices using load balancing algorithms, such as round-robin algorithm, least connections algorithm, etc.

An example is given in Fig 9. The main service module is composed of 4 devices. Each is configured with an /64 prefix, which is 2001:da8::/64, 2001:da8:0:1::/64, 2001:da8:0:2::/64, and 2001:da8:0:3::/64 respectively. Thus the prefix of the entire server is 2001:da8::/62. The load balancer resides in the entrance module, and it selects among these 4 prefixes according to real-time load using a load balancing algorithm. The entrance module generates the suffix using the encryption algorithm $f(SA)$ described in previous sections, then combine it with the selected prefix to generate the destination address.

In this case, the devices of the main service module do not need to be connected to the same first-hop router, and can be distributed deployed (Note the difference between Figs 8 and 9). Similar to CDN that is based on DNS redirection, the main service modules can be

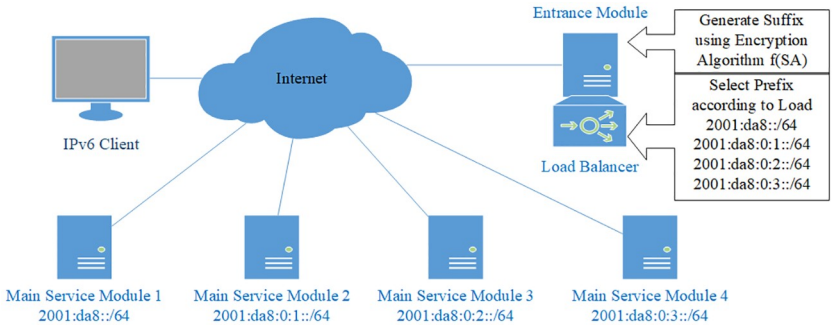

Fig 9. Dynamic load balancing based on entrance module strategy. The load balancer resides on entrance module and selects among the prefixes of multiple main service modules according to load.

https://doi.org/10.1371/journal.pone.0246293.g009

distributed all over the world, and the entrance module is responsible for server selection. The optimal main service module can be selected in a fully controllable manner using information such as geographic location, network capacity, and the ISP of the client.

All in all, our model provides a new and flexible framework that naturally supports various load balancing, high availability, and CDN features. For specific network scenarios like data center network, the network topology, structure, and routing configuration can be further optimized. This framework provides great potential for future work.

Security analysis

The primary goal of our model is to prevent the server from being scanned. Therefore, we first discuss its anti-scanning feature, then other security features. Finally, we analyze the big data characteristics of the addresses generated in our model and its ability to resist big data analysis.

Network scanning

As introduced in the Related Work Section, recent advances of IPv6 scanning can be divided into two types:

1. Scan by collecting active IPv6 address records.
2. Scan by generating a hitlist using pattern recognition algorithms.

Scanning by collecting active address records does not pose a threat to our model. The main service module is imperceptible to outside. If one of the addresses of the main service module is collected, as soon as the flow terminates, the address will no longer be active. This makes the collected address records completely useless.

Scanning by generating hitlists using pattern recognition algorithms does not pose a threat to our model either. According to the discussion in Subsection Big Data Analysis, the addresses generated by our algorithm do not have any pattern. This makes scanning through the generated hitlists no different from brute force scanning.

Another way to scan is to collect active addresses inside the subnet. However, as described in previous sections, the main service module does not use any global unicast address in the subnet. Attackers cannot get any useful addresses from the subnet. Therefore, this approach cannot pose a threat to our model as well.

As a result, for our model the above techniques are no different from brute force scanning, which is generally unworkable in IPv6.

However, things are complicated by our salting algorithm, which allows more than one destination addresses to pass the verification corresponding to the same source address at the same time. For example, if the step size X in Eq (11) is 5 milliseconds, and the threshold in Eq (16) is 10 seconds, then 2000 legitimate addresses can pass the verification at the same time. Assuming that the suffix length of the main service module is N, there are P legal timestamps within the threshold, then the probability that a random address can pass the verification is enlarged from $\frac{1}{2^N}$ to $\frac{P}{2^N}$. Fig 10 shows an illustration, where all addresses from Address 1 to Address 4 can pass the verification.

With the same scanning efficiency of Zmap under IPv4, which takes 45 minutes to scan the 2^{32} address space, the expected time T of the scanning will be

$$T = \frac{3 * 2^{N-32}}{4P} \tag{20}$$

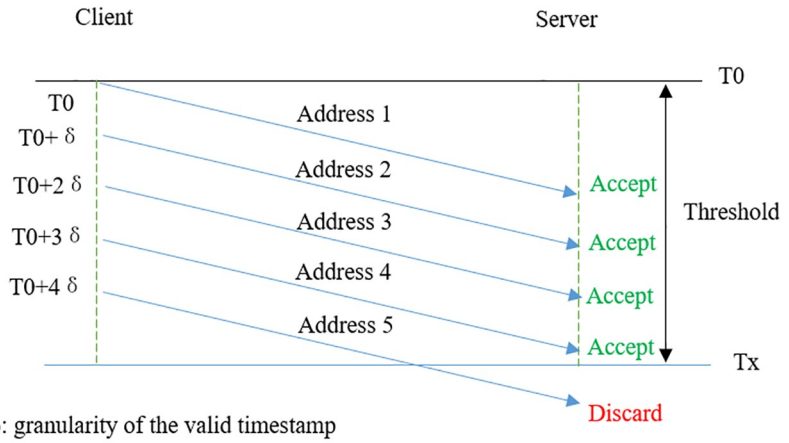

Fig 10. More than one legitimate destination addresses can pass verification. This is due to the *threshold* introduced in Eq (16).

https://doi.org/10.1371/journal.pone.0246293.g010

To protect the server from being scanned, T should be long enough, for example, longer than 1 year. Then P and N should satisfy Eq (21)

$$N - log_2 P \geq 46 \tag{21}$$

Eq (21) is easy to satisfy. For example, if a threshold of 10 seconds and a step size of 1ms are used, the expected scanning time T is about 9 years, which is too long to be worried about. Therefore, the complexity introduced by the salting algorithm does not harm the anti-scanning feature of our model.

Our above discussion focus on preventing the main service module from being scanned. Note that the entrance module is inevitably exposed to scanning, because some sort of entry address must be configured into the DNS system. However, first, scanning the entrance module does not pose a threat to the main server, because the address that provides the main service has been separated from the entrance module. Second, unnecessary exposure is eliminated through this separation, and the main service module, as the main body of the server, is well protected. Third, the entrance module is somehow similar to a bastion host. It provides no business logic, and the consequences of it being scanned are much more limited. Thus risks are isolated and controlled to a smaller scope.

DoS attack

DoS attack is one of the major threats faced by high-profile public servers. We discuss two typical types of DoS attack here: TCP Syn-Flood and UDP Flood.

Syn-Flood does not pose a threat to our model. All messages received by the main server are verified, and the ones with inappropriate source address-destination address pairs will be discarded. This means that the server does not need to send SYN+ACK messages, nor allocate CPU time or memory to maintain a queue waiting for subsequent packets.

Similarly, for UDP Floods, all the illegal UDP packets will be dropped immediately, and no resources will be allocated. Therefore, neither Syn-Flood nor UDP Flood poses a threat to our model.

Admittedly, the mitigation of DoS attacks is also limited. Our model cannot prevent the types of attacks that do not target the server directly, such as bandwidth attacks. Nor can it prevent attacks in which each of the malicious requests first gets a legitimate address from the

entrance module. In this case, an intrusion detection system can be deployed on the entrance module to detect and respond to abnormal traffic.

Again, the entrance module is still exposed to DoS attacks. However, the entrance provides limited service with little traffic load, so its ability to withstand DoS attacks is much higher. And its function is very simple, so it is easy to be duplicated, stacked, and distributed deployed, which further enhances its resistance to DoS attacks.

Application vulnerability attack

To prevent application vulnerability attacks generally requires software and hardware updates in time. However, our model can mitigate this threat by applying stricter authentication in the entrance module. The entrance module can be configured with authentication or admission strategies, so that only an authorized user can get the legitimate address, and the other requests are discarded. In this way, adversaries cannot even obtain the address of the main service module, let alone exploiting application or operating system vulnerabilities to launch attacks such as SQL injection attacks.

Replay attack and session hijack

A salting algorithm is introduced in our model to prevent replay attacks. The destination address generated by the entrance module is different each time. If an attacker launches replay attacks using previously intercepted packets, the address will already become illegal.

However, there is a complication introduced by the *threshold* in Eq (16). Though the entrance module dutifully generates a different address each time, a given address can pass the verification of the main service module in a time window of length *threshold*. Thus if an attacker intercepts a packet and launches replay attacks fast enough within *threshold*, the address will remain valid and pass the verification.

It can be solved by requiring the main service module to cache the addresses used in connections that have just ended within a time window of length *threshold*, and to reject connection requests with destination addresses that have been used recently. However, this caching is memory intensive for a high-traffic server. New vulnerabilities like cache exhaustion attack can be introduced, though the attacker needs to contact the entrance module first to get a huge amount of legitimate addresses. This strategy, accompanied by an intrusion detection system deployed on the entry module, can be an option for servers with high security requirements but not too much traffic.

Similar to fast replay attacks, the attacker can monitor the connection and launch session hijacking attacks. This is because verification is conducted only once for each flow in our model for performance considerations. Once the connection is established, the client and the main service module communicate directly without verification, which can be taken advantage of.

These two kinds of attacks can both be prevented using encryption at higher layers. Considering that our model is a network layer model, it is orthogonal thus compatible with encryption protocols at the transport or application layer, such as TLS [39], DTLS [70], and HTTPs [71]. These protocols can encrypt the packet payload and invalidate replay attacks and session hijacking attacks.

Key cracking

For breaking symmetric encryption, the plaintext-ciphertext pair needs to be obtained to guess the key. Even if an attacker intercepts the messages, the plaintext of the function $e()$ in Eq (6) is confidential and cannot be inferred from the source address. This is because the salting

algorithm in our model makes the plaintext time-varying. The precise timestamp is difficult to get, let alone the parameters T_0 and X are confidential. As a result, although DES is considered insecure in the modern Internet, it has a sufficient security level in our model.

ND related attack

ND attacks are unique in IPv6 because the ND protocol [72] is designed to replace the ARP [73] protocol of IPv4. There are many kinds of ND related attacks, which can be roughly divided into DAD attacks [74, 75], spoofing attacks (such as RA spoofing, malicious redirection), and cache exhaustion attacks.

Our model allocates a prefix to the main service module instead of an address, and there is only one device under this prefix. A lot of ND packets used for neighbor communication are no longer needed. Thus related threats are eliminated, including rouge NA and NS messages and DAD DoS Attacks [76].

On the other hand, ND messages are still used in the prefix allocating process by some mechanisms such as DHCP-PD. Thus RA spoofing attacks still pose a threat. And link-local addresses are used to communicate with the first-hop router in the subnet in our model, so ND attacks on the link-local addresses are still effective. Meanwhile, the entrance module still has a fixed address that can be attacked.

Therefore, although our model protects the main service module from a lot of ND attacks, it is recommended to deploy ND-related security policies such as SEND [38, 77], RA-guard [78, 79].

Big data analysis

In our model, one concern is that the attacker obtains a lot of source addresses-destination addresses pairs by intercepting a large number of network traffic data, and learns patterns of the generated addresses to launch attacks and scans. However, if the generated addresses have sufficiently good statistical characteristics, that is, random and evenly distributed enough, attackers cannot crack, scan, or attack by big data analysis.

We use simulations to demonstrate that the destination addresses generated using our algorithm have good statistical characteristics. In the simulation, we generate a large number of destination address suffixes then analyze them. Our simulation is conducted in 2 groups, and the results are shown in Fig 11.

The figure is plotted in the following manner. Since our algorithm generates address suffixes, we show the scatter plot of the generated suffixes. The longitudinal coordinate of each point is determined by the 65-96 bits of the address. For example, if the 65-96 bits are 0000:0000, then the vertical coordinate is 0; and if the 65-96 bits are FFFF:FFFF, then the vertical coordinate is 1. Similarly, the horizontal coordinate of a point is determined by the 97-128 bits of the address in the same way.

In Group 1, we use a fixed address as the source address, and generate 1000 destination address suffixes using it. The salt used in the process naturally varies over time, so the generated addresses are different. The distribution of the generated suffixes is shown in Fig 11(a). It shows that suffixes generated using the same source address over time are sufficiently evenly distributed and have no obvious pattern.

In Group 2, we collect 1000 active addresses in real-world network traffic from Tsinghua Campus Network. Then we use these 1000 source addresses to generate 1000 destination address suffixes. The results are shown in Fig 11(b). The red points are the suffixes of the source addresses, while the blue points are the suffixes of the generated addresses. An interesting observation is that the collected IPv6 addresses have an obvious pattern. A huge number of

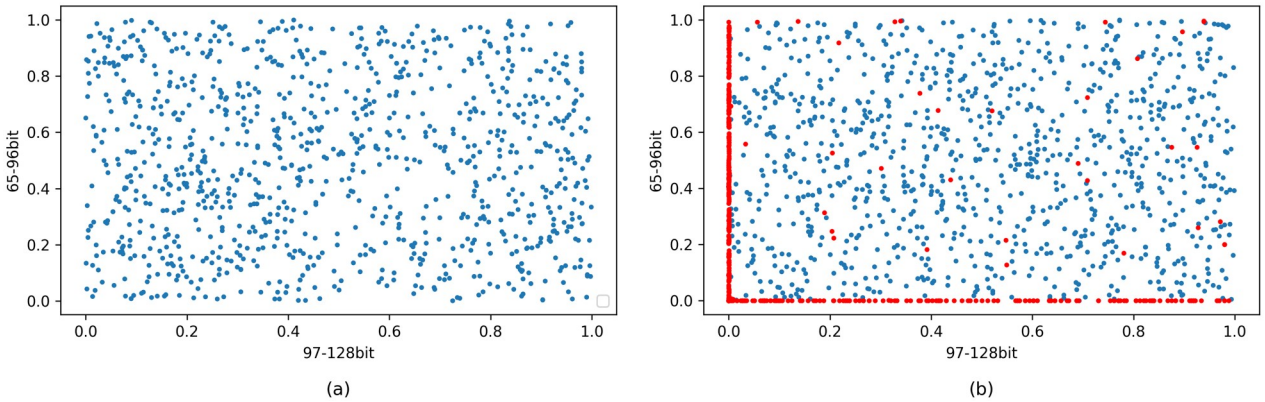

Fig 11. Distribution of the generated suffixes. (a) is the result of Group 1, where 1000 destination addresses are generated using one fixed source address at different times. (b) is the result of Group 2, where 1000 destination addresses (blue points) are generated using 1000 source addresses (red points) collected in real-world campus network traffic. The generated suffixes are evenly distributed in both groups.

https://doi.org/10.1371/journal.pone.0246293.g011

addresses set all of the 65-96 bits to zero, while another huge number of addresses set all of the 97-128 bits to zero. Nevertheless, the suffixes generated by our algorithm using these collected addresses are still evenly distributed.

Furthermore, to evaluate the randomness of the generated addresses, we calculated the entropy of them. This approach is often used to evaluate the randomness of IP addresses [16–19]. The entropy is calculated for each nybble (a nybble is 4-bit), so there are 16 entropy values in total for a 64-bit suffix. In probability theory, for a discrete random variable X with possible values $\{x_1, \ldots, x_k\}$ and probability mass function $P(X)$, entropy is defined as

$$H(X) = -\sum_{i=1}^{k} P(x_i) log P(x_i) \tag{22}$$

For each nybble in our case, there are 16 possible hex characters. From probability theory we know that $H(X)$ is maximum if $P(X)$ is uniform, so here the maximum entropy is log16. For the k-th nybble, if character c_i occurs N_i times (assume there are N samples totally), the entropy normalized using the maximum entropy is

$$e_k = -\frac{1}{4}\sum_{i=1}^{16} \frac{N_i}{N} log\left(\frac{N_i}{N}\right) \tag{23}$$

The results are shown in Fig 12. Fig 12(a) shows the results of Group 1, and Fig 12(b) shows the results of Group 2. The red line is the entropy of the source address suffixes, while the blue line is the entropy of the generated destination address suffixes. It is interesting yet reasonable that the collected addresses are not so random, especially for higher nybbles. In both groups, the entropy of the generated suffixes for each nybble is almost 1. Since the entropy is normalized using the maximum entropy, this means that no matter the source addresses are random or not, the generated address suffixes are sufficiently random in all bits.

In short, our model can withstand big data analysis attacks launched by obtaining and analyzing large volumes of traffic data.

Prototype implementation and experiments

In this section, we introduce our prototype implementation and experiments based on the prototype.

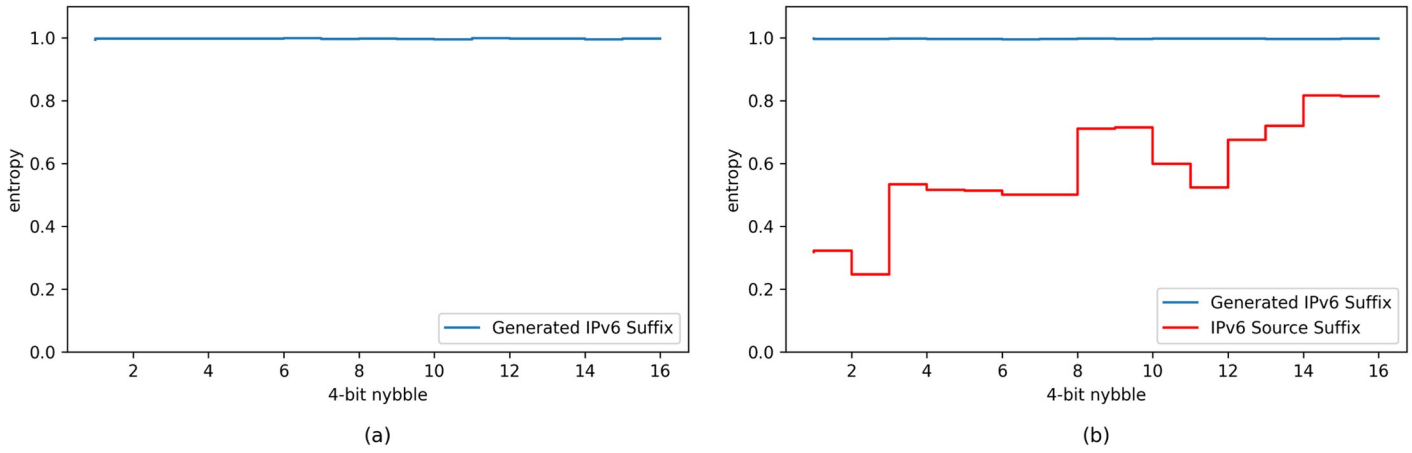

Fig 12. Entropy of the generated suffixes. (a) is the result of Group 1, where 1000 destination addresses are generated using one fixed source address at different times. (b) is the result of Group 2, where 1000 destination addresses are generated using 1000 source addresses collected in real-world campus network traffic. Blue line shows the entropy of the suffixes of the generated destination addresses, while red line shows the entropy of the suffixes of the source addresses. In both groups, the entropy of the generated suffixes is almost 1, which is the max entropy. The generated address suffixes are sufficiently random in all bits.

https://doi.org/10.1371/journal.pone.0246293.g012

Prototype implementation

Our prototype is implemented based on Linux, specifically, Raspbian OS based on Debian. We make modifications to the NetFilter Linux kernel module to implement the mechanism. In our prototype, the following features are implemented:

1. The server is divided into two modules. The entrance module is configured with an IPv6 address while the main service module is assigned a prefix using DHCP-PD. The main service module listens on all the addresses under the prefix.
2. The entrance module and the main service module save the same key. When the entrance module receives a request, it generates an address under the main service module prefix and redirects the request to that address. When the main service module receives a request, it conducts verification on the flow.
3. The encryption and the verification use DES as e() in Eq (6), and the salt is added as Eq (11) to prevent replay attacks.

The algorithm implemented in the prototype is described as follows. The encryption algorithm executed in the entrance module is Algorithm 1, while the verification algorithm executed in the main service module is Algorithm 2.

Algorithm 1 Encryption Algorithm Executed by the Entrance module

```
Input: SourceAddress,Key,Prefix,T0,X
Output: DestinationAddress
1: H_SA = DJB(SourceAddress)
2: T_current = time.currenttime()
3: Salt = (T_current - T0) / X
4: P_SA = XOR(Salt, H_SA)
5: Suffix = DES(P_SA,Key)
6: DestinationAddress = strcat(Prefix, Suffix)
7: return DestinationAddress
```

Algorithm 2 Verification Algorithm Executed by the Main Service module

```
Input: SourceAddress,DestinationAddress,Key,T0,X,T_threshold
Output: True/False
1: H_SA = DJB(SourceAddress)
```

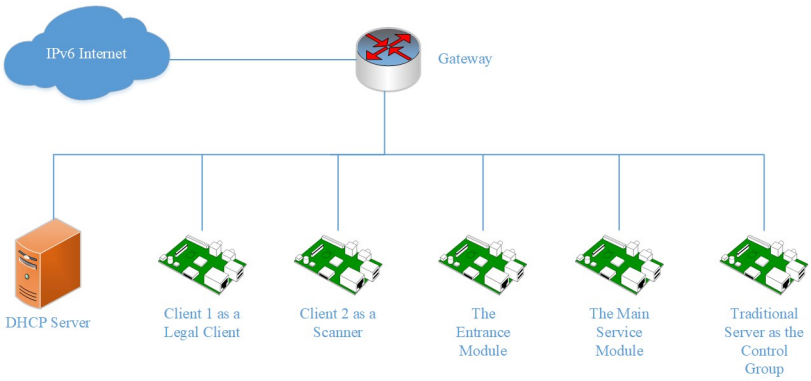

Fig 13. Topology of the experiment subnet.

https://doi.org/10.1371/journal.pone.0246293.g013

```
2: Suffix = DestinationAddress [64:128]
3: P_SA = DES^-1(Suffix, Key)
4: T_current = time.currenttime()
5: Salt = XOR(H_SA, P_SA)
6: T_send = Salt*X + T0
7: T_delta = T_current - T_send
8: if T_delta < T_threshold and T_delta > 0 then
9:   return True
10: end if
11: return False
```

Experiment environment

Our experiment is built on a /48 subnet in CERNET (China Education and Research Network). The entrance module and the main service module are configured in the same subnet. We use DHCP-PD to allocate a prefix to the main service module while the addresses of the client and the entrance module are configured manually. We use Kea for the DHCP server.

We use Raspberry Pis for all the devices in the experiment, and the operating system of each Pi is Raspbian, which comes with the Raspberry Pi and is based on Debian.

The experiment topology is described in Fig 13.

Five devices are configured in the experiment subnet. One is a legal client, one is a scanner, one is the entrance module of the server, and one is the main service module. To offer a baseline for the experiment, we configure the last one a traditional server as the control group.

Experiment on defense of scanning

In the experiment, the server is configured as a public HTTP web server based on the apache and Django framework. The demo web page displays the source address and destination address of the visit for the demonstration. We use a client to access the server through the entrance module first, and the result shows that the client can visit the main server smoothly, while the client cannot get any response if it tries to visit the main server directly. This shows that the main server cannot be perceived by the outside world directly.

To test the scanning defense feature of the main server, a client is used as the attacker. Three types of scans are tested in the experiment. First, we perform a 100-hour brute-force scan on the main service module. The result shows that the scan does not hit any address.

Then, we generate 1 million addresses under the /64 prefix using Entropy/IP and 6gen, and use them as the hitlists to scan the server. The scan does not hit any address as well.

Finally, ND information for other devices in the subnet is collected to compose a hitlist to conduct the scan. Similarly, no address can be accessed as well. Therefore, our model can prevent the main service module from being scanned by existing IPv6 scanning approaches.

Experiment on performance

We conduct two sets of experiments on performance in this subsection. First on the delay in the connection establishment phase, then on the RTT, bandwidth, and jitter after the connection is established.

Because our model introduces additional overhead only in the connection establishment phase which influences the delay most, we first conduct experiments on the delay in the connection establishment phase. We use Wireshark to monitor the delay. Compared with the control group, the extra delay introduced by our model equals the time elapses between the entrance module receiving the first packet and the main server module receiving the first packet. This value can be described by Eq (24).

$$T_{add} = T_{trans\_en} + T_{trans\_ma} + T_{process\_cl} + T_{encryption\_en} + T_{verification\_ma} \tag{24}$$

In Eq (24), T_{trans_en} is the transmission delay from the entrance module to the client; T_{trans_ma} is the transmission delay from the client to the main service module; $T_{process\_cl}$ is the processing time from the client receiving the redirect message to the client sending the new request to the main server; $T_{encryption\_en}$ is the processing time from the entrance module receiving the request to it sending the redirect message; while $T_{verification\_ma}$ is the processing time from the main service module receiving the request to it completing the verification. We test these five items separately to analyze to which degree each of them influences the delay.

First, T_{trans_en} and T_{trans_ma} typically vary from several to hundreds of milliseconds. The one-way transmission delay measured in our experiment environment is very small (several milliseconds), but this value is greatly affected by the specific network environment of each client, so our results are not representative and we will not display it here. However, considering that the additional cost caused by two one-way delays is usually a small number (generally at most a few hundred milliseconds) comparing with the total access time cost, it may not cause a user-perceivable impact.

Second, we test the effect of $T_{process\_cl}$. $T_{process\_cl}$ is dependent on the hardware and the application of the client. We conducted the experiments in three groups of client settings: Group (a) uses Linux and Wget, Group (b) uses Windows and IE 11 Browser, and Group (c) uses Windows and Chrome Browser. Each client launches access to our prototype website and $T_{process\_cl}$ is measured in each test using tcpdump/Wireshark. Each group launches access to our prototype website for five times, with an hour interval between each. The result is shown in Fig 14.

From Fig 14, we can see that $T_{process\_cl}$ is greatly affected by different hardware and software environment, but it is less than 10 milliseconds in all groups. It means that $T_{process\_cl}$ brings little additional delay and does not influence user experience at all.

Finally, $T_{encryption\_en}$ and $T_{verification\_ma}$ represent the execution time of the encryption and verification process, respectively. We conduct experiments on the encryption and verification time cost. The experiment is repeated 10 times. The result is shown in Fig 15.

From Fig 15, we can see that the execution time of a single encryption or verification operation does not exceed 0.05 milliseconds, which is negligible to the delay and will not affect the server performance.

In summary, the additional delay brought by our model is determined by the two one-way transmission delay, while the sum of the other time overhead will not exceed several milliseconds. The total additional delay is acceptable and will not affect user experience.

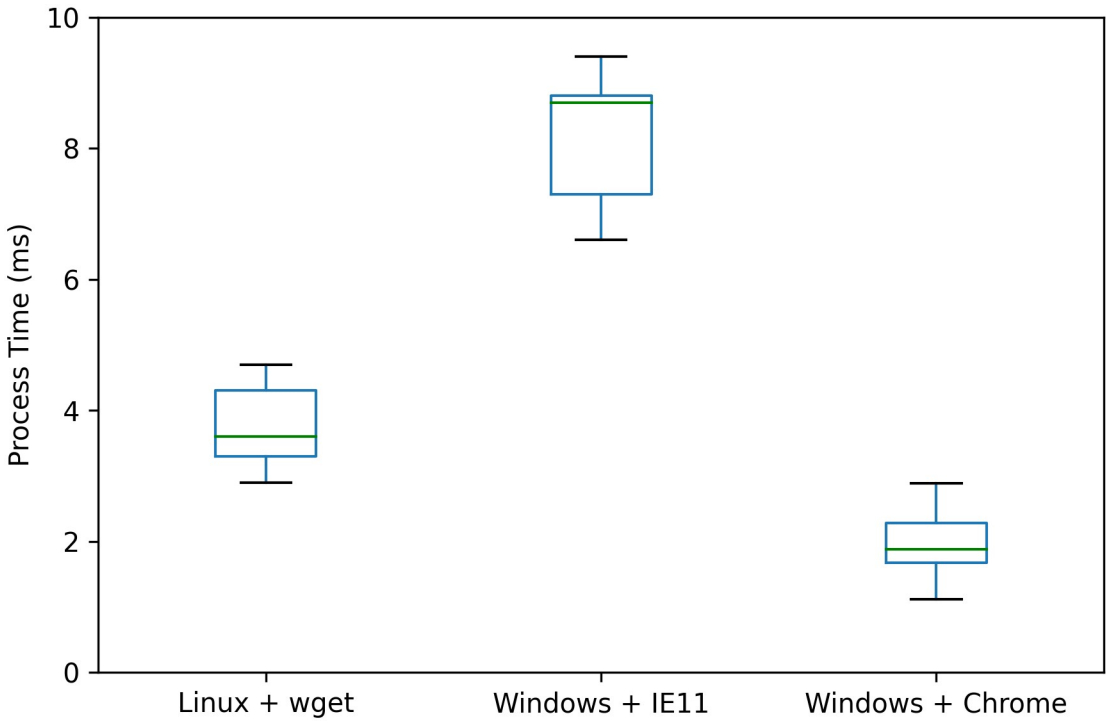

Fig 14. $T_{process\_ci}$ with different OSes and browsers. We measure the process time of the client from receiving the redirect message to it sending the new request to the main server. It is greatly affected by the environment, but will not exceeds 10 milliseconds.

https://doi.org/10.1371/journal.pone.0246293.g014

In order to test the impact of our model on network performance after the connection is established, we conduct experiments on the RTT, bandwidth, and jitter between the client and the server. The result is shown in Fig 16. Our model does not bring any additional cost in RTT, bandwidth, or jitter once the connection is established.

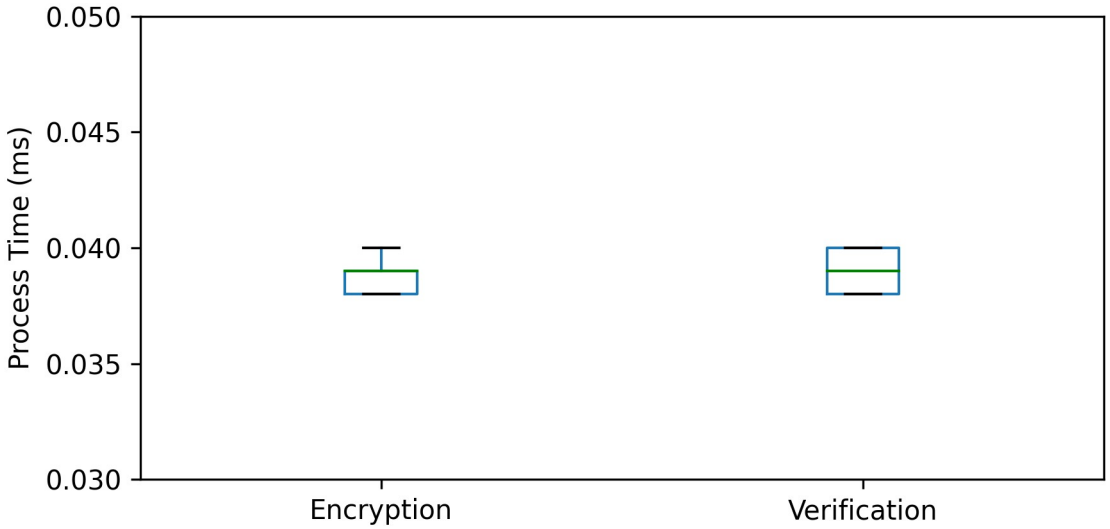

Fig 15. Encryption time and verification time. The time cost is less than 0.05 milliseconds, and its influence to server performance is negligible.

https://doi.org/10.1371/journal.pone.0246293.g015

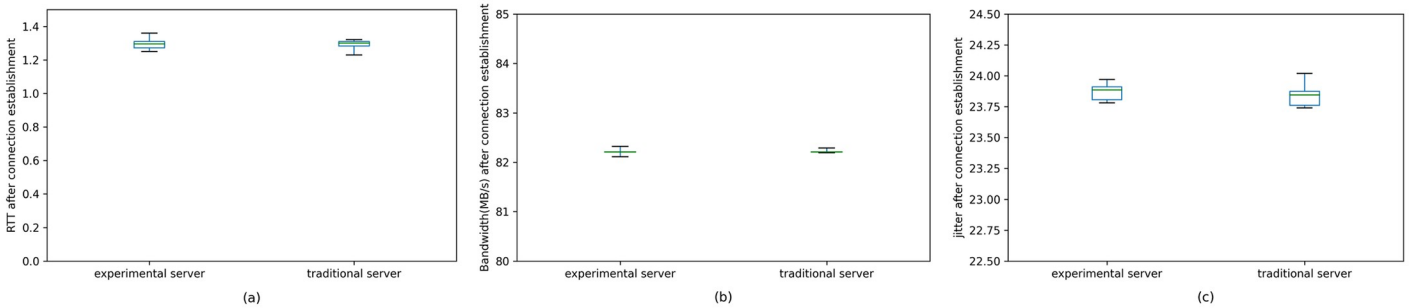

Fig 16. RTT, bandwidth, and jitter between client and server after connection establishment. There is no performance difference between our model and a traditional server. Our model brings no additional cost once the connection is established.

https://doi.org/10.1371/journal.pone.0246293.g016

Discussion

Compatibility, deployability, and evolvability

Our model has good compatibility with the current Internet. A client agnostic of our model can visit the server smoothly. And none of the other participants of the Internet need any modification, such as ISP or DNS servers. Moreover, the modification is limited within the network layer, thus the model transparently supports protocols of the transportation layer and application layer. All in all, our model requires no transition mechanism, is simple, and can be easily deployed.

Not only can our model provide a new perspective on many security and functional issues including the scanning problem, but also it provides a new and flexible framework for public servers. Various strategies can be formulated based on this framework, such as user authentication on the entrance module, distributed deployment, and load balancing and high availability of the server cluster, etc. We believe that various exciting models can be proposed based on this framework. Therefore, our model has good scalability and potential.

Consideration on IPv6 address space

Is it too extravagant that we assign an entire prefix to one server in our model? Although the number of prefixes that can be allocated is much fewer than the number of addresses in the IPv6 address space, the worry that the IPv6 addresses will be exhausted is unnecessary. All addresses under 2000::/3 are global unicast addresses. There are 8.6 billion global unicast /36 prefixes in total, which means that everyone in the world can be allocated a /36 prefix, not to mention that there are 268 million /64 prefixes under a single /36 prefix. Even if a server is allocated a /56 prefix or even a /48 prefix, the current IPv6 address space is still enough. Therefore, assigning a prefix to each server will not exhaust the IPv6 address space.

Consideration on IPv4

Is our model available for IPv4? Theoretically, yes. In IPv4, the server can also be divided into an entrance module and a main service module, the main service module can also be configured with a block of addresses while the entrance module can also generate a verifiable address using encryption algorithm and provide redirection. However, there are two problems in IPv4:

1. In IPv6, we use a 64-bit suffix length to carry the ciphertext. However, in IPv4, the entire address space is only 32-bit. It cannot provide sufficient security level in encryption, resulting in the encryption being easily cracked.

2. Different from the high redundancy of IPv6 address space, IPv4 addresses are very scarce. In IPv4, to allocate a block of addresses to one server is wasteful and literally too expensive [80]. Therefore, it is unrealistic.

Therefore, our model is only recommended to be used in IPv6.

Conclusion

In this paper, a novel Internet server model named addressless server is introduced. This model separates the entrance module from the main service module, and uses prefix delegation mechanism to allocate an IPv6 prefix instead of an IPv6 address to the main service module. When a user sends a request to the server, the entrance module generates an address under the main service module prefix by encrypting the user address, then redirects the request to the generated address. A time-varying salt is added in the encryption process to make the address different in each connection. The main service module verifies each request received and drops all packets that failed verification. In this way, our model can prevent the main server from being scanned and other security threats such as DoS attacks and replay attacks.

Our model takes advantage of the massive IPv6 address space to hide the address in use, and incorporates encryption into IPv6 addresses, which can fully utilize the redundant space of IPv6 address. By allocating a prefix to the main service module, our model eliminates the one-to-one correspondence between the server and the IP address. Our model not only shields the server from scanning, but also establishes a new network framework for servers that supports desirable features such as load balancing, active-active cluster, and CDN conveniently.

We implement a prototype of the addressless server and conduct simulations and experiments based on it. The results show that users can access the server smoothly while scan traffic cannot get any response. The addresses generated by the algorithm are sufficiently random and uniformly distributed, which prevents attackers from using big data analysis to scan the servers or crack the keys. Our experiments on the performance show that only during the establishment of the connection does it bring additional delay, and it does not affect user experience. Once the connection is established, our model will not affect delay, bandwidth, or jitter.

Supporting information

S1 Raw data.
(RAR)

Author Contributions

Conceptualization: Shanshan Hao, Renjie Liu, Congxiao Bao, Xing Li.

Data curation: Shanshan Hao, Renjie Liu.

Formal analysis: Shanshan Hao, Renjie Liu.

Investigation: Shanshan Hao, Renjie Liu.

Methodology: Renjie Liu, Deliang Chang, Xing Li.

Project administration: Xing Li.

Software: Zhe Weng, Deliang Chang.

Supervision: Congxiao Bao, Xing Li.

Validation: Shanshan Hao, Renjie Liu.

Visualization: Shanshan Hao.

Writing – original draft: Renjie Liu.

Writing – review & editing: Shanshan Hao, Renjie Liu.

References

1. Deering, S. and Hinden, R. "Internet protocol, version 6 (IPv6) specification". RFC 8200, 2017.
2. https://www.vyncke.org/ipv6status/
3. https://www.statista.com/statistics/471264/iot-number-of-connected-devices-worldwide
4. https://www.facebook.com/ipv6/
5. https://teamarin.net/2019/04/03/microsoft-works-toward-ipv6-only-single-stack-network/
6. https://www.iab.org/2016/11/07/iab-statement-on-ipv6/
7. Durumeric, Zakir, Eric Wustrow, and J. Alex Halderman. "ZMap: Fast Internet-wide scanning and its security applications". 22nd USENIX Security Symposium (USENIX Security 13). 2013.
8. Fiebig, Tobias, et al. "Something from nothing (There): collecting global IPv6 datasets from DNS". International Conference on Passive and Active Network Measurement. Springer, Cham, 2017.
9. Borgolte, Kevin, et al. "Enumerating active IPv6 hosts for large-scale security scans via DNSSEC-signed reverse zones". 2018 IEEE Symposium on Security and Privacy (SP). IEEE, 2018.
10. Fiebig, Tobias, et al. "In rDNS we trust: revisiting a common data-source's reliability". International Conference on Passive and Active Network Measurement. Springer, Cham, 2018.
11. Beverly, Robert, et al. "In the IP of the beholder: Strategies for active IPv6 topology discovery". Proceedings of the Internet Measurement Conference 2018. 2018.
12. Rohrer, Justin P., Blake LaFever, and Robert Beverly. "Empirical study of router IPv6 interface address distributions". IEEE Internet Computing 20.4 (2016): 36-45.
13. Rye, Erik C., and Robert Beverly. "Discovering the IPv6 Network Periphery". International Conference on Passive and Active Network Measurement. Springer, Cham, 2020.
14. Gasser, Oliver, et al. "Scanning the IPv6 internet: towards a comprehensive hitlist". arXiv preprint arXiv:1607.05179 (2016).
15. Gasser, Oliver, et al. "Clusters in the expanse: Understanding and unbiasing IPv6 hitlists". Proceedings of the Internet Measurement Conference 2018. 2018.
16. Foremski, Pawel, David Plonka, and Arthur Berger. "Entropy/ip: Uncovering structure in ipv6 addresses". Proceedings of the 2016 Internet Measurement Conference. 2016.
17. Ullrich, Johanna, et al. "On reconnaissance with IPv6: a pattern-based scanning approach". 2015 10th International Conference on Availability, Reliability and Security. IEEE, 2015.
18. Zhihao Zuo, et al. "Prediction algorithm of active IPv6 address prefix," Journal on Commmunications, 2018 S1: 7–14.
19. Murdock, Austin, et al. "Target generation for internet-wide IPv6 scanning". Proceedings of the 2017 Internet Measurement Conference. 2017.
20. Liu Zhizhu, et al. "6Tree: Efficient dynamic discovery of active addresses in the IPv6 address space". Computer Networks 155 (2019): 31–46. https://doi.org/10.1016/j.comnet.2019.03.010
21. Cui, Tianyu, Gaopeng Gou, and Gang Xiong."6GCVAE: Gated Convolutional Variational Autoencoder for IPv6 Target Generation". Pacific-Asia Conference on Knowledge Discovery and Data Mining. Springer, Cham, 2020.
22. Gont, F. "A Method for Generating Semantically Opaque Interface Identifiers with IPv6 Stateless Address Autoconfiguration (SLAAC)", RFC 7217, 2014.
23. Narten T, Draves R, Krishnan S. "Privacy Extensions for Stateless Address Autoconfiguration in IPv6". RFC 4941, 2007.
24. Gont, F. and Chown, T. "Network Reconnaissance in IPv6 Networks." RFC 7707, 2016.
25. M. Sifalakis, S. Schmid and D. Hutchison, "Network address hopping: a mechanism to enhance data protection for packet communications", IEEE International Conference on Communications, 2005.
26. Mavani Monali, and Krishna Asawa. "Privacy preserving ipv6 address auto-configuration for internet of things". Intelligent Communication and Computational Technologies. Springer, Singapore, 2018. 3–14.

27. Dunlop, Matthew, et al. "The blind man's bluff approach to security using IPv6". IEEE Security & Privacy 10.4 (2012): 35-43.

28. Judmayer, Aljosha, et al. "Lightweight address hopping for defending the IPv6 IoT". Proceedings of the 12th International Conference on Availability, Reliability and Security. 2017.

29. Jafarian, Jafar Haadi, et al. "Openflow Random Host Mutation: Transparent Moving Target Defense Using Software Defined Networking". Proceedings of the First Workshop on Hot Topics in Software Defined Networks, 2012, pp. 127-132.

30. Sharma, Dilli Prasad, et al. "FRVM: Flexible Random Virtual IP Multiplexing in Software-Defined Networks". 2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/ 12th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE), 2018, pp. 579-587.

31. Gleitz, Peter M., and Steven M. Bellovin. "Transient Addressing for Related Processes: Improved Firewalling by Using IPV6 and Multiple Addresses per Host". USENIX Security Symposium. 2001.

32. Aura, Tuomas, and Alf Zugenmaier. "Privacy, control and internet mobility". International Workshop on Security Protocols. Springer, Berlin, Heidelberg, 2004.

33. Lindqvist, Janne, and Juha-Matti Tapio. "Protecting privacy with protocol stack virtualization". Proceedings of the 7th ACM workshop on Privacy in the electronic society. 2008.

34. Sakurai, Atsushi, et al. "One-time receiver address in IPv6 for protecting unlinkability". Annual Asian Computing Science Conference. Springer, Berlin, Heidelberg, 2007.

35. Han, Seungyeop, et al. "Expressive privacy control with pseudonyms". ACM SIGCOMM Computer Communication Review 43.4 (2013): 291-302.

36. Trostle, Jonathan. "Applying network address encryption to anonymity and preventing data exfiltration". MILCOM 2008-2008 IEEE Military Communications Conference. IEEE, 2008.

37. Lee, Taeho, et al. "Communication based on per-packet One-Time Addresses". 2016 IEEE 24th International Conference on Network Protocols (ICNP). IEEE, 2016.

38. Gagliano, R., S. Krishnan, and A. Kukec. "Certificate Profile and Certificate Management for SEcure Neighbor Discovery (SEND)". RFC 6494, Feb, 2012.

39. Rescorla, E, and Dierks, T. "The transport layer security (TLS) protocol version 1.3". RFC 8446, 2018.

40. Ateniese, Giuseppe, and Stefan Mangard. "A new approach to DNS security (DNSSEC)". Proceedings of the 8th ACM conference on Computer and Communications Security. 2001.

41. Kent, Stephen, and Karen Seo. "Security architecture for the internet protocol" RFC 4301, 2005.

42. Aura, T. "Cryptographically Generated Addresses (CGA)". RFC 3972, 2005.

43. Mrugalski, T., et al. "Dynamic host configuration protocol for IPv6 (DHCPv6)". RFC 8415, 2018.

44. Lyon, Gordon Fyodor. "Nmap network scanning: The official Nmap project guide to network discovery and security scanning". Insecure, 2009.

45. Gasser, Oliver. "Evaluating network security using Internet-wide measurements." Diss. Technische Universität München, 2019.

46. Hinden, R., and S. Deering. "IP version 6 addressing architecture". RFC 4291, 2006.

47. Thomson, S., T. Narten, and T. Jinmei. "IPv6 Stateless Address Autoconfiguration" RFC 4862, 2007.

48. Crawford, Matt. "Transmission of IPv6 packets over ethernet networks". RFC 2464, December, 1998.

49. Rivest, R. "The MD5 message-digest algorithm". RFC 1321, 1992.

50. Bound, J., et al. "Dynamic host configuration protocol for IPv6 (DHCPv6)". RFC 3315, 2003.

51. Huitema, C., and T. Mrugalski. S. Krishnan, "Anonymity Profile for DHCP Clients". RFC 7844, 2016.

52. Gont, F. and Liu, W. "A Method for Generating Semantically Opaque Interface Identifiers (IIDs) with the Dynamic Host Configuration Protocol for IPv6 (DHCPv6)", RFC 7943, 2016.

53. Thaler, D. "Privacy Considerations for IPv6 Adaptation-Layer Mechanisms." RFC 8065, 2017.

54. Plonka, David, and Arthur Berger. "Temporal and spatial classification of active IPv6 addresses". Proceedings of the 2015 Internet Measurement Conference. 2015.

55. Li Fuliang, et al. "Characteristics analysis at prefix granularity: A case study in an IPv6 network". Journal of Network and Computer Applications 70 (2016): 156–170. https://doi.org/10.1016/j.jnca.2016.02.022

56. Fukuda, Kensuke, and John Heidemann. "Who knocks at the ipv6 door? detecting ipv6 scanning". Proceedings of the Internet Measurement Conference 2018. 2018.

57. Plonka, David, and Arthur Berger. "KIP: A measured approach to IPv6 address anonymization". arXiv preprint arXiv:1707.03900 (2017).

58. Brzozowski, J., and G. Van de Velde. "Unique IPv6 prefix per host." RFC 8273, 2017.

59. Antonatos S., Akritidis P., Markatos E.P. and Anagnostakis K.G., "Defending against hitlist worms using network address space randomization", Computer Networks, vol. 51, no. 12, pp. 3471–3490, 2007. https://doi.org/10.1016/j.comnet.2007.02.006

60. Herrmann, Dominik, Christine Arndt, and Hannes Federrath. "Ipv6 prefix alteration: An opportunity to improve online privacy". arXiv preprint arXiv:1211.4704 (2012).

61. Marx, Matthias, et al. "Context-Aware IPv6 Address Hopping". International Conference on Information and Communications Security. Springer, Cham, 2019.

62. Heydari, Vahid, and S. Yoo. "Moving target defense enhanced by mobile ipv6". 7th Annual Southeastern Cyber Security Summit (2015).

63. Zarif, Nasrin Sadat, et al. "A New Hybrid Method of IPv6 Addressing in the Internet of Things". 2019 Smart Grid Conference (SGC). IEEE, 2019.

64. Hinden, Robert, and Brian Haberman. "Unique local IPv6 unicast addresses." RFC 4193, October, 2005.

65. Eastlake 3rd, D., and Paul Jones. "Us secure hash algorithm 1 (sha1)". RFC 3174, 2001.

66. National Bureau of Standards, Data Encryption Standard, FIPS Publication 46, 1977.

67. Daemen, J. and Rijmen, V. "AES Proposal: Rijndael", Banksys/Katholieke Universiteit Leuven, Belgium, AES submission, Jun 1998.

68. M'Raihi, David, et al. "TOTP: Time-Based One-Time Password Algorithm," RFC 6238, 2011.

69. Narten, T., Huston, G., and Roberts, L. "IPv6 Address Assignment to End Sites". RFC 6177, 2011.

70. Rescorla, E. and Modadugu, N. "Datagram Transport Layer Security Version 1.2" RFC 6347, 2012.

71. Rescorla, E. "HTTP Over TLS". RFC 2818, 2000.

72. Narten T, Nordmark E, Simpson W. "Neighbor discovery for IP version" RFC 4861, 2007.

73. David, C. Plummer. "An Ethernet address resolution protocol". RFC 826, 1982.

74. Nikander, P., Kempf, J., and Nordmark, E. "IPv6 neighbor discovery (ND) trust models and threats". RFC 3756, May, 2004.

75. Goel, Jai Narayan, and B. M. Mehtre. "Dynamic IPv6 activation based defense for IPv6 router advertisement flooding (DoS) attack". 2014 IEEE International Conference on Computational Intelligence and Computing Research. IEEE, 2014.

76. Nikander, Pekka, James Kempf, and Erik Nordmark. "IPv6 neighbor discovery (ND) trust models and threats". RFC 3756, May, 2004.

77. Alsa'deh, Ahmad, and Christoph Meinel. "Secure neighbor discovery: Review, challenges, perspectives, and recommendations". IEEE Security & Privacy 10.4 (2012): 26-34.

78. Levy-Abegnoli, Eric, et al."IPv6 router advertisement guard". RFC 6105, 2011.

79. Gont, F. "Implementation advice for ipv6 router advertisement guard (ra-guard)". RFC 7113, 2014.

80. https://ipv4marketgroup.com/ipv4-pricing/