US009270646B2

(12) **United States Patent**
Shelest

(10) **Patent No.: US 9,270,646 B2**
(45) **Date of Patent: Feb. 23, 2016**

(54) **SYSTEMS AND METHODS FOR GENERATING A DNS QUERY TO IMPROVE RESISTANCE AGAINST A DNS ATTACK**

(75) Inventor: **Art Shelest**, Coral Springs, FL (US)

(73) Assignee: **Citrix Systems, Inc.**, Fort Lauderdale, FL (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 1321 days.

(21) Appl. No.: **12/426,330**

(22) Filed: **Apr. 20, 2009**

(65) **Prior Publication Data**

US 2010/0269174 A1 Oct. 21, 2010

(51) **Int. Cl.**

G06F 21/10	(2013.01)
H04L 29/06	(2006.01)
H04L 29/12	(2006.01)
H04L 9/32	(2006.01)

(52) **U.S. Cl.**

CPC ***H04L 63/04*** (2013.01); ***H04L 9/3236*** (2013.01); ***H04L 29/12066*** (2013.01); ***H04L 61/1511*** (2013.01); ***H04L 63/14*** (2013.01); *H04L 2209/56* (2013.01)

(58) **Field of Classification Search**

CPC G06F 21/10
USPC 705/50–79; 726/26
See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

7,296,155	B1 *	11/2007	Trostle et al.	713/170
2003/0135625	A1 *	7/2003	Fontes	H04L 63/12 709/228
2004/0083306	A1 *	4/2004	Gloe	H04L 29/12066 709/245
2010/0121981	A1 *	5/2010	Drako	709/245
2011/0022675	A1 *	1/2011	Bayles	709/206

FOREIGN PATENT DOCUMENTS

JP 2003208371 A * 7/2003

OTHER PUBLICATIONS

Hubert Netherlabs Computer Consulting BV., R. van Mook Equinix. 01CMeasures for Making DNS More Resilient against Forged Answers,01D >, Internet Enginerering Task Force, IETF; Standardworkingdraft, Internet Society (ISOC), Jan. 1, 2009.

(Continued)

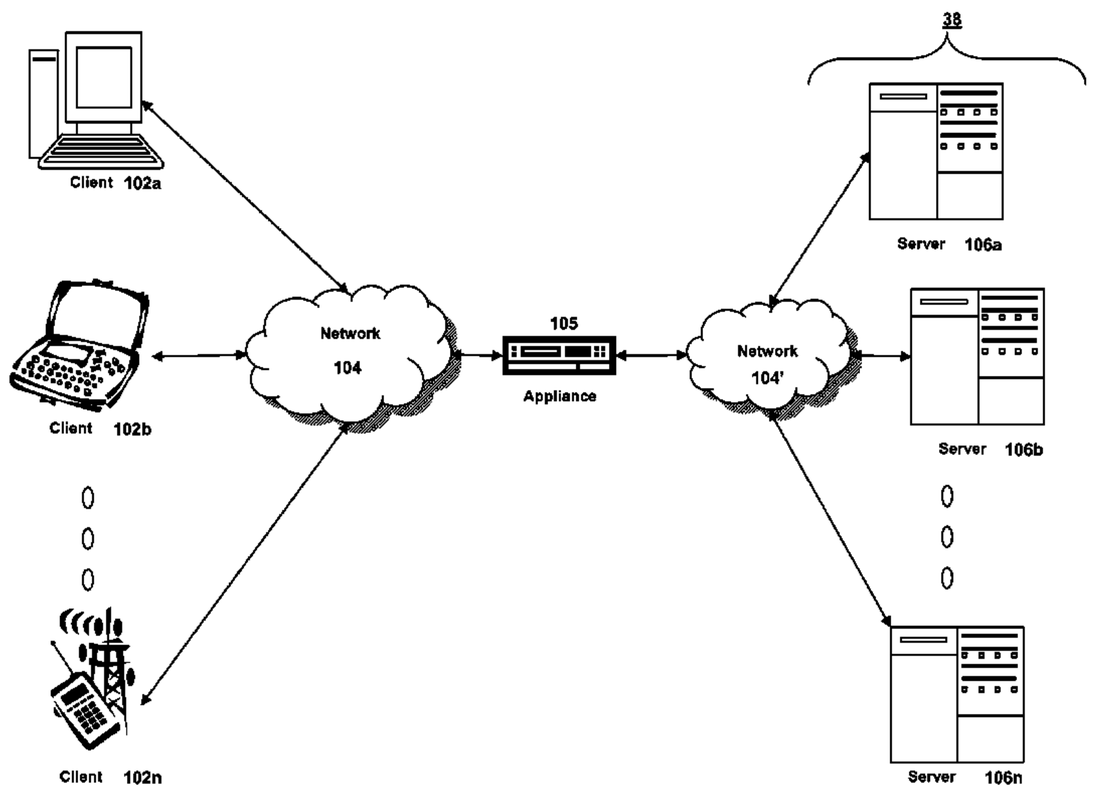
Primary Examiner — James A Reagan

(74) *Attorney, Agent, or Firm* — Christopher J. McKenna; Foley & Lardner LLP

(57) **ABSTRACT**

The present solution provides systems and methods for generating DNS queries that are more resistant to being compromised by attackers. To generate the transaction identifier, the DNS resolver uses a cryptographic hash function. The inputs to the hash function may include a predetermined random number, the destination IP address of the name server to be queried, and the domain name to be queried. Because of the inclusion of the name server's IP address in the formula, queries for the same domain name to different name servers may have different transaction identifiers, preventing an attacker from observing a query and predicting the identifiers for other queries. Additional entropy may be provided for generating transaction identifiers by including the port number of the name server and/or a portion of the domain name as inputs to the hash function. If it is determined that the responding server may preserve capitalization in its responses, the upper and lower case characters may be salted within the domain name to provide additional entropy in generating transaction identifiers.

20 Claims, 6 Drawing Sheets



(56) **References Cited**

OTHER PUBLICATIONS

International Preliminary Report on Patentability on PCT/US2010/027132 dated Feb. 16, 2012.
International Search Report on PCT/US2010/027132 dated Jan. 31, 2012.
Menezes A J et al., 01CHandbook of Applied Cryptography, Chapter 5—Pseudorandom Bits and Sequences,01D Jan. 1, 1997.
Vixie P. et al., 01CUse of Bit 0×20 in DNS Labels to Improve Transaction Identity,01D <ft-vixie-dnsext-dns0x20-00.txt>, Internet Enginerering Task Force, IETF; Standardworkingdraft, Internet Society (ISOC), Mar. 17, 2008.
Written Opinion on PCT/US2010/027132 dated Jan. 31, 2012.
Chinese Office Action for Chinese Application No. 201080026895.6 dated Aug. 12, 2014.
Hubert et al, Measures for Making DNS More Resilient against Forged Answers, Netherlands Computer Consulting BV., Network Work Group, Jan. 2009 (18 pages).
CN Office Action for Application No. 201080026895.6 dated Jan. 30, 2014.

* cited by examiner

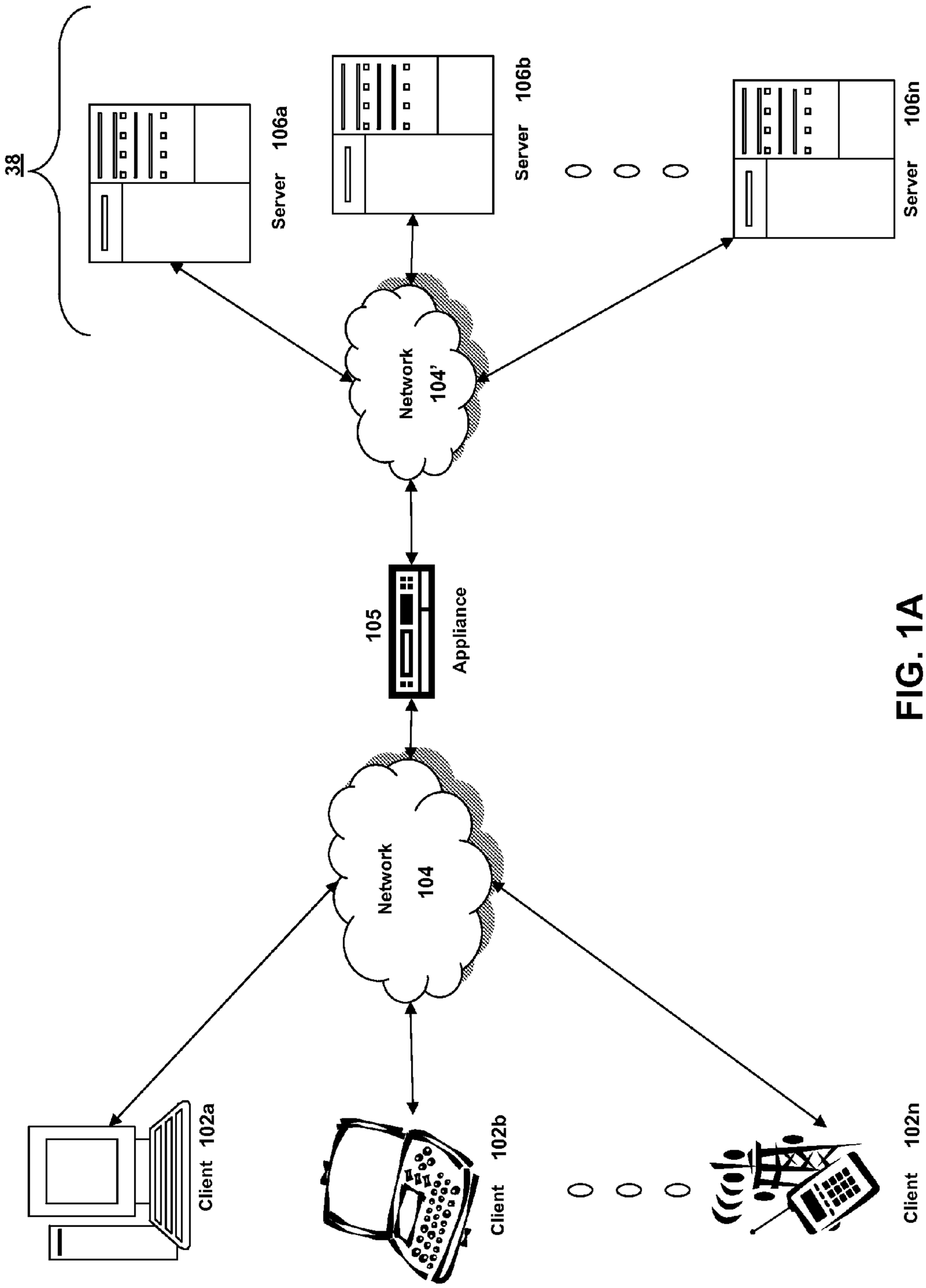

38
Server 106a
Server 106b
Server 106n
Network 104'
105
Appliance
Network 104
Client 102a
Client 102b
Client 102n

FIG. 1A

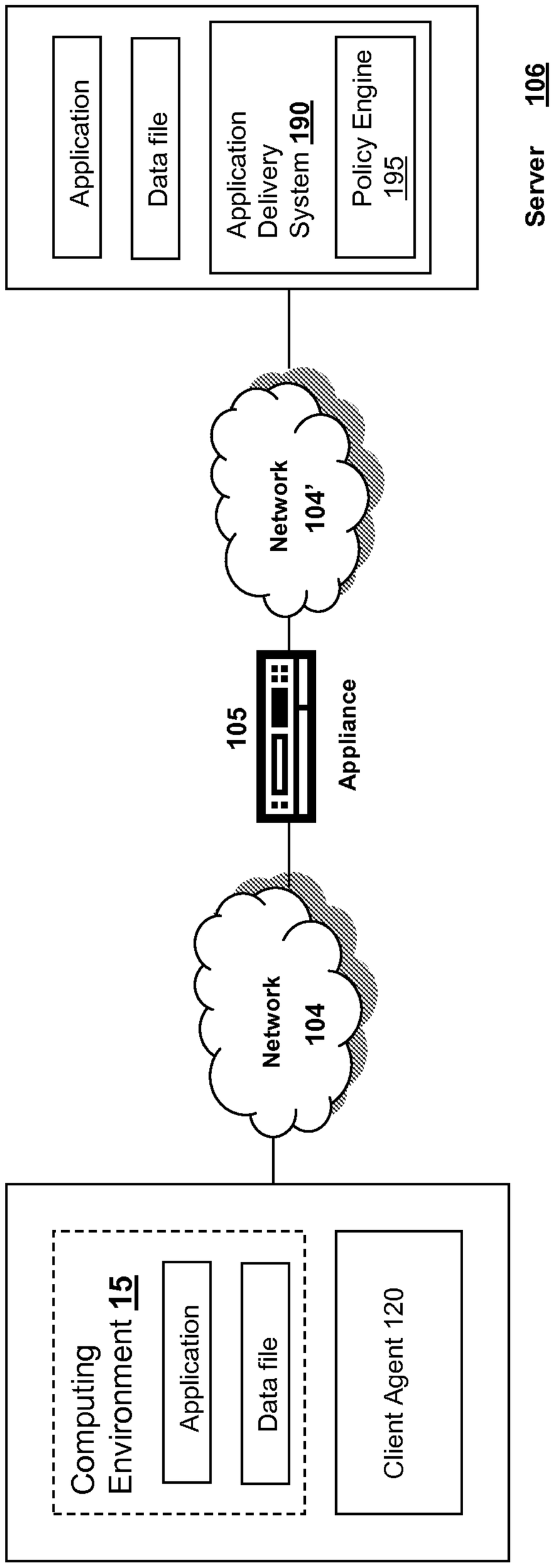

Application
Data file
Application Delivery System 190
Policy Engine 195
Server 106
Network 104'
105
Appliance
Network 104
Computing Environment 15
Application
Data file
Client Agent 120
Client 102

FIG. 1B

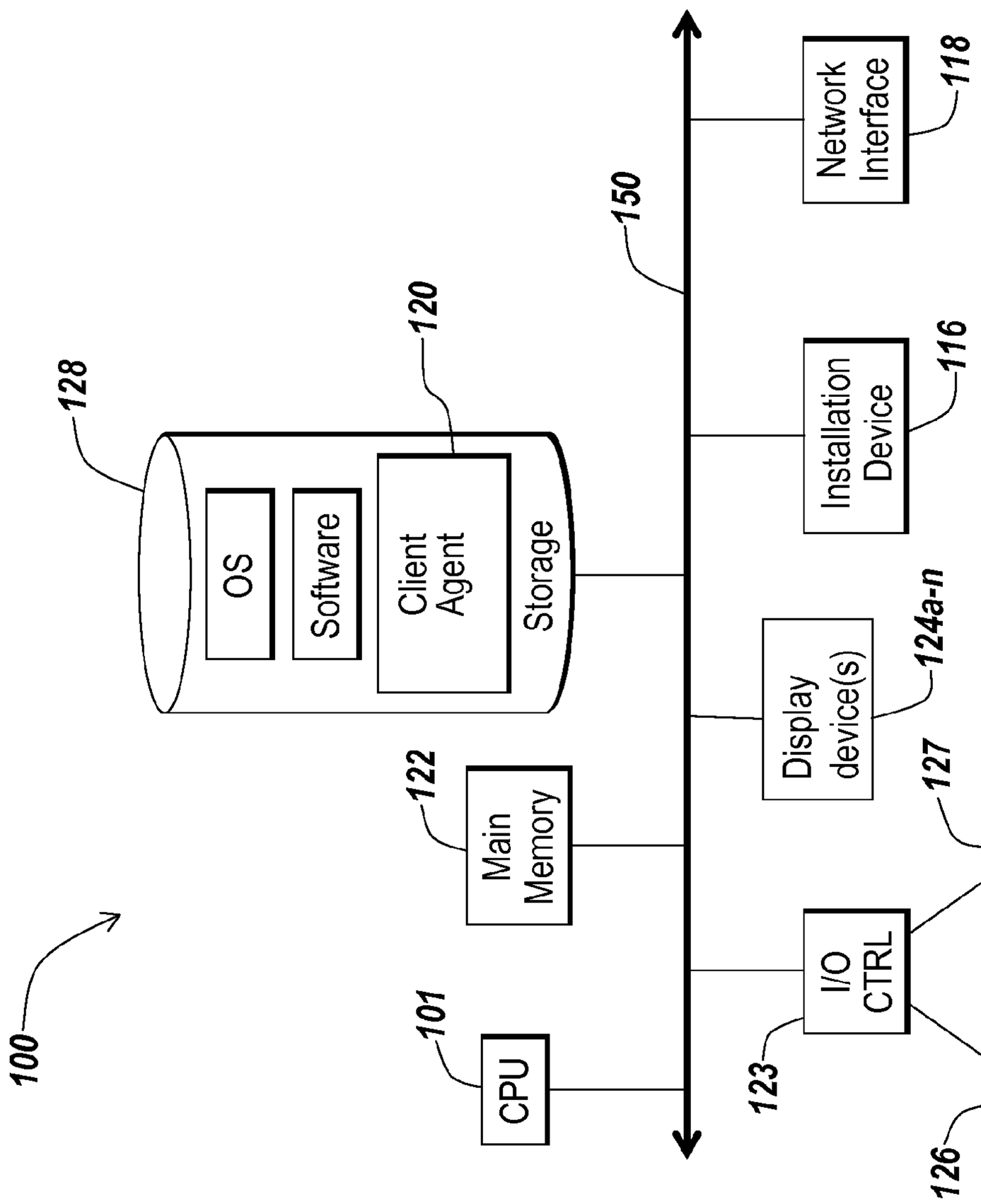

100
101
CPU
122
Main Memory
128
OS
Software
Client Agent
120
Storage
150
Network Interface
118
Installation Device
116
Display device(s)
124a-n
127
Pointing Device
I/O CTRL
123
Keyboard
126

FIG. 1C

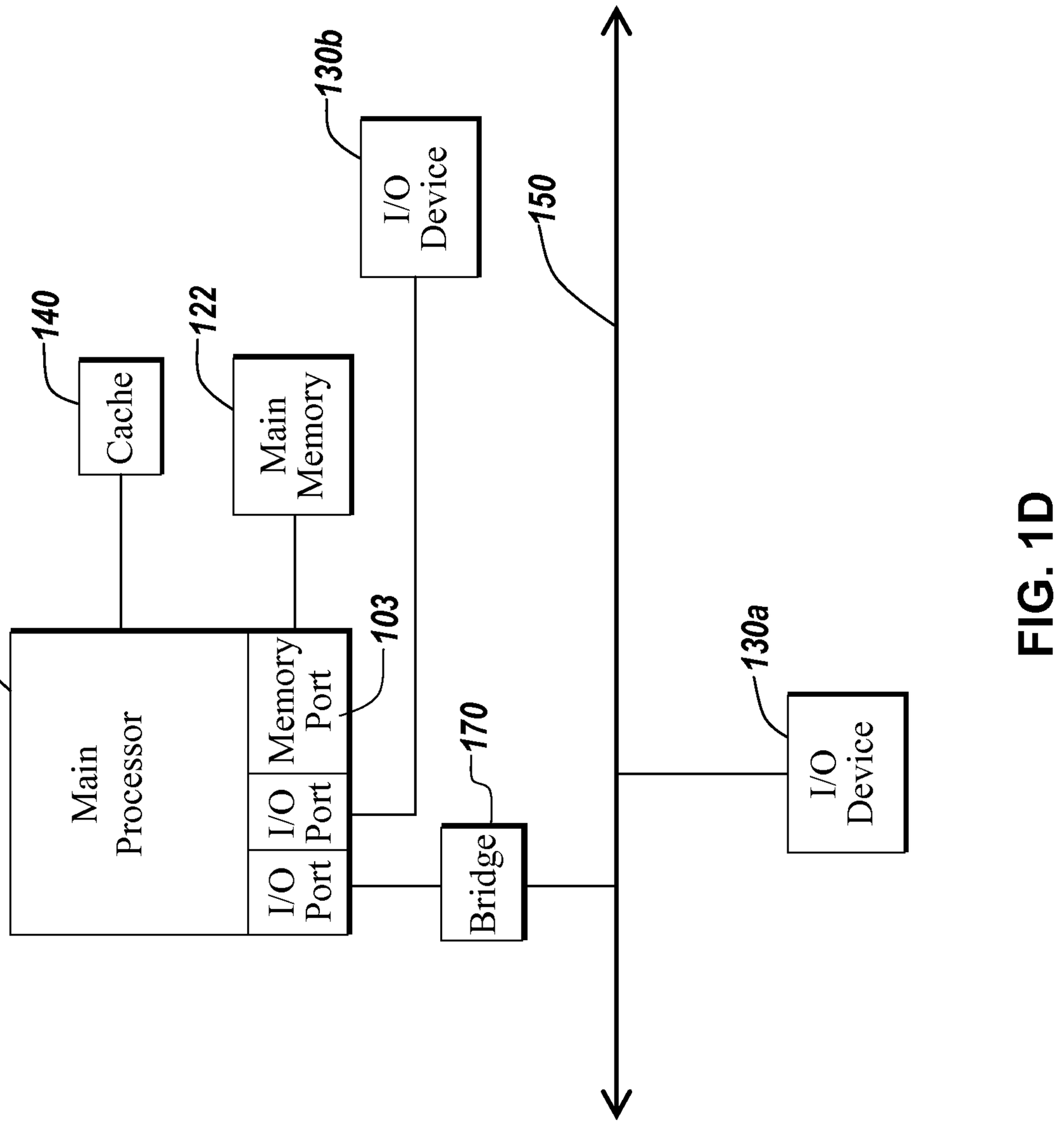

101
Main Processor
I/O Port
I/O Port
Memory Port
103
Cache
140
Main Memory
122
Bridge
170
I/O Device
130b
I/O Device
130a
150

FIG. 1D

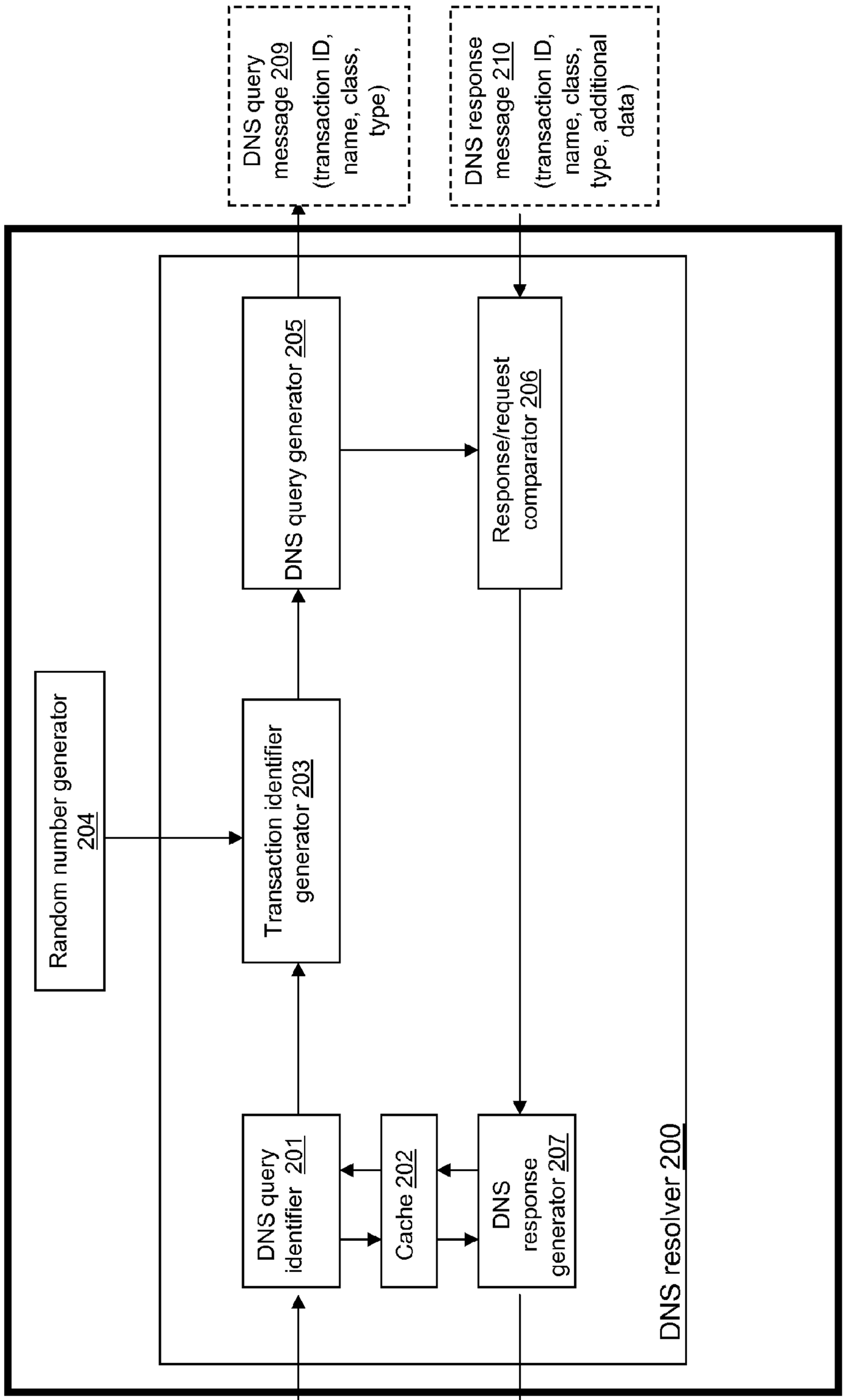

DNS query message 209 (transaction ID, name, class, type)
DNS response message 210 (transaction ID, name, class, type, additional data)
Random number generator 204
Transaction identifier generator 203
DNS query generator 205
Response/request comparator 206
DNS query identifier 201
Cache 202
DNS response generator 207
DNS resolver 200
Client, Server, or Intermediary
DNS resolution request 208 (name, class, type)
DNS resolution response 211 (name, IP address)

FIG. 2

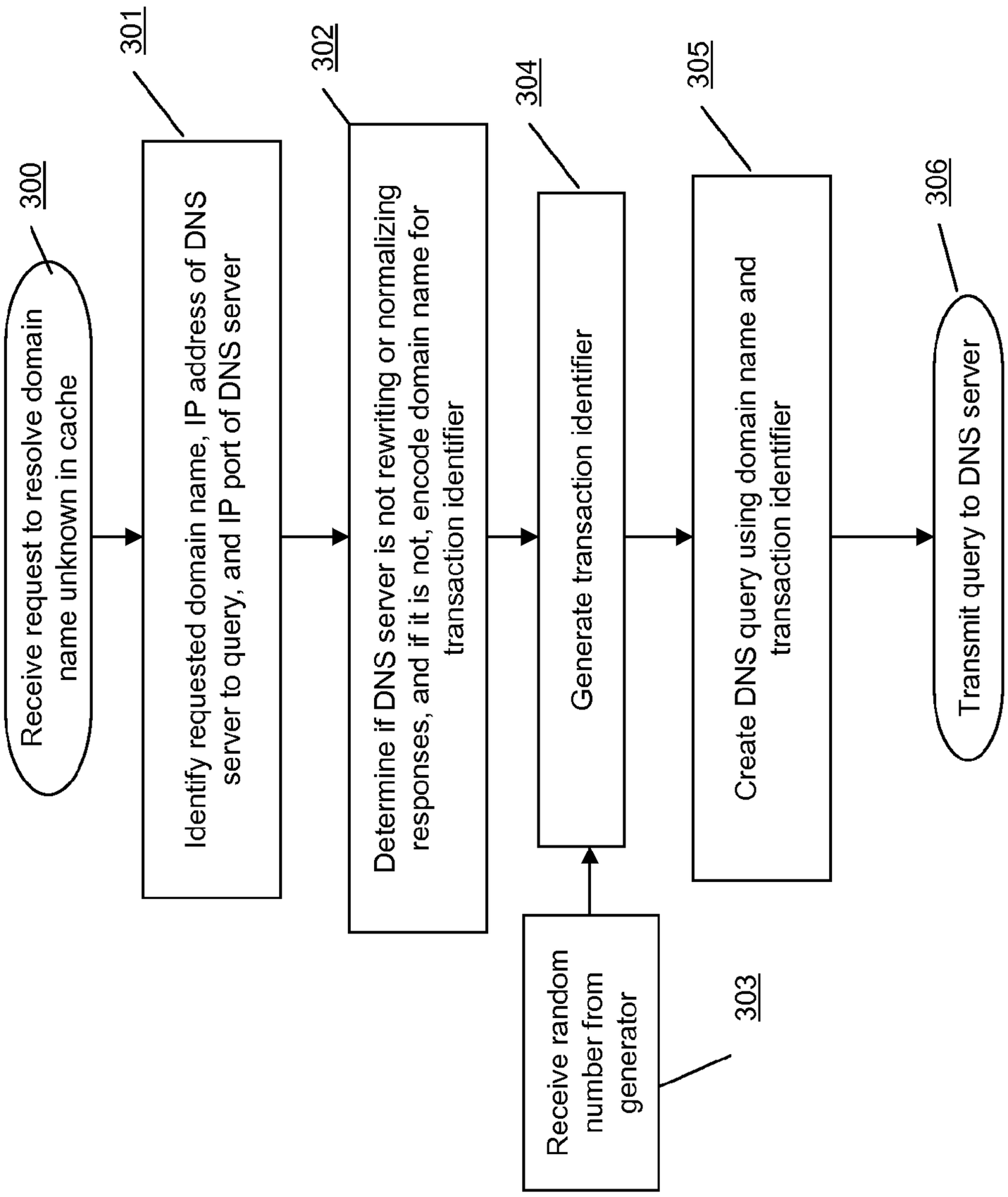

300
Receive request to resolve domain name unknown in cache
301
Identify requested domain name, IP address of DNS server to query, and IP port of DNS server
302
Determine if DNS server is not rewriting or normalizing responses, and if it is not, encode domain name for transaction identifier
304
Generate transaction identifier
305
Create DNS query using domain name and transaction identifier
306
Transmit query to DNS server
303
Receive random number from generator

FIG. 3

SYSTEMS AND METHODS FOR GENERATING A DNS QUERY TO IMPROVE RESISTANCE AGAINST A DNS ATTACK

FIELD OF THE INVENTION

The present application generally relates to data communication networks. In particular, the present application relates to systems and methods for generating a Domain Name System ["DNS"] query to improve resistance against a DNS attack.

BACKGROUND OF THE INVENTION

The Domain Name System ["DNS"] allows human meaningful names to be associated with the numerical internet protocol ["IP"] addresses of clients, servers, or other resources on the internet. For example, the domain name www.example.com may be associated with 208.77.188.166. Domain names are mapped and indexed by name servers. Each name server is authoritative or responsible for indexing clients, servers, or other resources within its zone of authority. When a user requests a resource by domain name, a DNS resolver identifies the request. If the IP address for the requested resource is not available in its cache, the resolver initiates a query to a name server. The DNS resolver's query includes a transaction identifier. The name server's reply may also include the transaction identifier to identify the response as having come from the name server queried by the DNS resolver. If a malicious attacker can respond to a DNS resolver's request before the real name server can, the malicious attacker can direct the user to a different client, server, or resource than was intended. This opens possibilities of identity or data theft or other malicious activities.

BRIEF SUMMARY OF THE INVENTION

The present solution provides systems and methods for generating DNS queries that are more resistant to being compromised by attackers. To generate the transaction identifier, the DNS resolver uses a cryptographic hash function. The inputs to the hash function may include a predetermined random number, the destination IP address of the name server to be queried, and the domain name to be queried. Because of the inclusion of the name server's IP address in the formula, queries for the same domain name to different name servers may have different transaction identifiers, preventing an attacker from observing a query and predicting the identifiers for other queries. Additional entropy may be provided for generating transaction identifiers by including the port number of the name server and/or a portion of the domain name as inputs to the hash function. If it is determined that the responding server may preserve capitalization in its responses, the upper and lower case characters may be salted within the domain name to provide additional entropy in generating transaction identifiers.

In one aspect, the present invention features a method for generating a DNS query to improve resistance against a DNS attack. The method includes a DNS resolver receiving a request to resolve a domain name. The method also includes the DNS resolver identifying the domain name and an IP address of a DNS server. The method further includes generating a transaction identifier for a DNS query by applying a one-way hash function to an input of a predetermined random number, the IP address of the DNS server and the domain name. The method also includes the DNS resolver transmitting a DNS query for the domain name to the DNS server, the DNS query identified by the generated transaction identifier.

In some embodiments, the method includes the DNS resolver identifying the IP port number of the DNS server. In further embodiments, the method includes generating the transaction identifier for the DNS query by applying the one-way hash function to the input of a predetermined random number, the IP address and the port of the DNS server, and the domain name. In yet further embodiments, the domain name input to the one-way hash function may comprise a portion of the domain name to be resolved.

In one embodiment, the method includes changing the predetermined random number input to the one-way hash function at a predetermined frequency. In another embodiment, the method includes changing the predetermined random number in response to an event. In other embodiments, the method includes generating the same transaction identifier for DNS queries to resolve the same domain name transmitted to the same DNS server. In still other embodiments, the method includes encoding one or more fields of the DNS request and using the encoded one or more fields as input to the one-way hash function to generate the transaction identifier. In other embodiments, the method includes encoding the domain name by capitalizing one or more characters of the domain name and generating the transaction identifier by using the encoded domain name as the input of the domain name to the one-way hash function. In still other embodiments, the method further comprises encoding the domain name input to the one-way hash function by using a punycode or a RACE encoding scheme.

In another embodiment, the method further comprises the DNS resolver determining that the DNS server rewrites or normalizes responses. In response to the determination, the DNS resolver may not encode a portion of the DNS query. In other embodiments, the method further comprises the DNS resolver determining that the destination is not rewriting responses. In response to the determination, the DNS resolver may encode a portion of the DNS query and include the encoded portion in the transaction identifier. In yet other embodiments, the method further includes the DNS resolver communicating the input of the IP address of the destination and the domain name to a transaction identifier generator.

In another aspect, the present invention features a system for generating a DNS query to improve resistance against a DNS attack. The system includes a DNS resolver and a transaction identifier generator. The DNS resolver receives a request to resolve a domain name and identifies the domain name and an IP address of a destination of the request. The transaction identifier generator that generates a transaction identifier by applying a one-way hash function to an input of a predetermined random number, the IP address of the destination and the domain name. The DNS resolver forms the DNS query using the generated transaction identifier and transmits the DNS query for the domain name to the destination.

In one embodiment, the DNS resolver identifies a port of the destination of the request. In further embodiments, the transaction identifier generator may generate the transaction identifier by applying the one-way hash function to the input of the predetermined random number, the internet protocol address and the port of the destination and the domain name. In still further embodiments, the domain name input to the one-way hash function may comprise a portion of the domain name to be resolved.

In another embodiment, the transaction identifier generator changes the predetermined random number at a predetermined frequency. In yet another embodiment, the transaction

identifier generator changes the predetermined random number in response to an event. In still another embodiment, the transaction identifier generator generates the same transaction identifier for inputs identifying the same domain name and the same destination.

In other embodiments, the DNS resolver encodes one or more fields of the DNS request and communicates the encoded one or more fields as input to the transaction identifier generator to generate the transaction identifier. In another embodiment, the DNS resolver encodes the domain name by capitalizing one or more characters of the domain name and communicates the encoded domain name as the input of the domain name to the transaction identifier generator. In still another embodiment, the DNS resolver encodes the domain name by using a punycode or a RACE encoding scheme. In other embodiments, the DNS resolver may determine that the destination rewrites or normalizes responses, and in response to the determination the DNS resolver may not encode a portion of the DNS query. In yet other embodiments, the DNS resolver may determine that the destination does not rewrite responses, and in response to the determination the DNS resolver may encode a portion of the DNS query and communicate the encoded portion as input to the transaction identifier generator to generate the transaction identifier. In still other embodiments, the DNS resolver may reside on a client, a server, or an intermediary.

The details of various embodiments of the invention are set forth in the accompanying drawings and the description below.

BRIEF DESCRIPTION OF THE FIGURES

The foregoing and other objects, aspects, features, and advantages of the invention will become more apparent and better understood by referring to the following description taken in conjunction with the accompanying drawings, in which:

FIG. **1**A is a block diagram of an embodiment of a network environment for a client to access a server via an appliance;

FIG. **1**B is a block diagram of another embodiment of an environment for delivering a computing environment from a server to a client via;

FIGS. **1**C and **1**D are block diagrams of embodiments of a computing device;

FIG. **2** is a block diagram of an embodiment of a domain name resolver; and

FIG. **3** is a flow diagram of an embodiment of steps of a method for generating a DNS query with improved resistance against a DNS attack.

The features and advantages of the present invention will become more apparent from the detailed description set forth below when taken in conjunction with the drawings, in which like reference characters identify corresponding elements throughout. In the drawings, like reference numbers generally indicate identical, functionally similar, and/or structurally similar elements.

DETAILED DESCRIPTION OF THE INVENTION

For purposes of reading the description of the various embodiments below, the following descriptions of the sections of the specification and their respective contents may be helpful:

Section A describes a network environment and computing environment which may be useful for practicing embodiments described herein;

Section B describes embodiments of systems and methods for responding to DNS name resolution requests, transmitting requests to name servers, receiving responses from name servers, and transmitting responses to DNS name resolution requests; and

Section C describes embodiments of systems for and methods of generating DNS queries with improved resistance to DNS attacks.

A. Network and Computing Environment

Prior to discussing the specifics of embodiments of the systems and methods of an appliance and/or client, it may be helpful to discuss the network and computing environments in which such embodiments may be deployed. Referring now to FIG. **1**A, an embodiment of a network environment is depicted. In brief overview, the network environment comprises one or more clients **102***a*-**102***n* (also generally referred to as local machine(s) **102**, or client(s) **102**) in communication with one or more servers **106***a*-**106***n* (also generally referred to as server(s) **106**, or remote machine(s) **106**) via one or more networks **104**, **104'** (generally referred to as network **104**). In some embodiments, a client **102** communicates with a server **106** via an appliance **105**.

Although FIG. **1**A shows a network **104** and a network **104'** between the clients **102** and the servers **106**, the clients **102** and the servers **106** may be on the same network **104**. The networks **104** and **104'** can be the same type of network or different types of networks. The network **104** and/or the network **104'** can be a local-area network (LAN), such as a company Intranet, a metropolitan area network (MAN), or a wide area network (WAN), such as the Internet or the World Wide Web. In one embodiment, network **104'** may be a private network and network **104** may be a public network. In some embodiments, network **104** may be a private network and network **104'** a public network. In another embodiment, networks **104** and **104'** may both be private networks. In some embodiments, clients **102** may be located at a branch office of a corporate enterprise communicating via a WAN connection over the network **104** to the servers **106** located at a corporate data center.

The network **104** and/or **104'** be any type and/or form of network and may include any of the following: a point to point network, a broadcast network, a wide area network, a local area network, a telecommunications network, a data communication network, a computer network, an ATM (Asynchronous Transfer Mode) network, a SONET (Synchronous Optical Network) network, a SDH (Synchronous Digital Hierarchy) network, a wireless network and a wireline network. In some embodiments, the network **104** may comprise a wireless link, such as an infrared channel or satellite band. The topology of the network **104** and/or **104'** may be a bus, star, or ring network topology. The network **104** and/or **104'** and network topology may be of any such network or network topology as known to those ordinarily skilled in the art capable of supporting the operations described herein.

As shown in FIG. **1**A, the appliance **105**, which also may be referred to as an interface unit **105** or gateway **105**, is shown between the networks **104** and **104'**. In some embodiments, the appliance **105** may be located on network **104**. For example, a branch office of a corporate enterprise may deploy an appliance **105** at the branch office. In other embodiments, the appliance **105** may be located on network **104'**. For example, an appliance **105** may be located at a corporate data center. In yet another embodiment, a plurality of appliances **105** may be deployed on network **104**. In some embodiments, a plurality of appliances **105** may be deployed on network **104'**. In other embodiments, a plurality of appliances **105** may be deployed on both networks **104** and **104'**. In other embodi-

ments, the appliance **105** could be a part of any client **102** or server **106** on the same or different network **104**, **104'** as the client **102**. One or more appliances **105** may be located at any point in the network or network communications path between a client **102** and a server **106**.

In some embodiments, the appliance **105** comprises any of the network devices manufactured by Citrix Systems, Inc. of Ft. Lauderdale Fla., such as the Citrix NetScaler™, Citrix WANScaler™, Citrix Repeater™, Citrix Branch Repeater™, or Citrix Branch Repeater™ with Windows Server®. In other embodiments, the appliance **105** includes any of the product embodiments referred to as WebAccelerator and BigIP manufactured by F5 Networks, Inc. of Seattle, Wash. In yet another embodiment, the appliance **105** includes any application acceleration and/or security related appliances and/or software manufactured by Cisco Systems, Inc. of San Jose, Calif., such as the Cisco ACE Application Control Engine Module service software and network modules, and Cisco AVS Series Application Velocity System.

In one embodiment, the system may include multiple, logically-grouped servers **106**. In these embodiments, the logical group of servers may be referred to as a server farm **38**. In some of these embodiments, the servers **106** may be geographically dispersed. In some cases, a farm **38** may be administered as a single entity. In other embodiments, the server farm **38** comprises a plurality of server farms **38**. In one embodiment, the server farm executes one or more applications on behalf of one or more clients **102**.

The servers **106** within each farm **38** can be heterogeneous. One or more of the servers **106** can operate according to one type of operating system platform (e.g., WINDOWS NT, manufactured by Microsoft Corp. of Redmond, Wash.), while one or more of the other servers **106** can operate on according to another type of operating system platform (e.g., Unix or Linux). The servers **106** of each farm **38** do not need to be physically proximate to another server **106** in the same farm **38**. Thus, the group of servers **106** logically grouped as a farm **38** may be interconnected using a wide-area network (WAN) connection or medium-area network (MAN) connection. For example, a farm **38** may include servers **106** physically located in different continents or different regions of a continent, country, state, city, campus, or room. Data transmission speeds between servers **106** in the farm **38** can be increased if the servers **106** are connected using a local-area network (LAN) connection or some form of direct connection.

Servers **106** may be referred to as a file server, application server, web server, proxy server, or gateway server. In some embodiments, a server **106** may have the capacity to function as either an application server or as a master application server. In one embodiment, a server **106** may include an Active Directory. The clients **102** may also be referred to as client nodes or endpoints. In some embodiments, a client **102** has the capacity to function as both a client node seeking access to applications on a server and as an application server providing access to hosted applications for other clients **102***a*-**102***n*.

In some embodiments, a client **102** communicates with a server **106**. In one embodiment, the client **102** communicates directly with one of the servers **106** in a farm **38**. In another embodiment, the client **102** executes a program neighborhood application to communicate with a server **106** in a farm **38**. In still another embodiment, the server **106** provides the functionality of a master node. In some embodiments, the client **102** communicates with the server **106** in the farm **38** through a network **104**. Over the network **104**, the client **102** can, for example, request execution of various applications hosted by the servers **106***a*-**106***n* in the farm **38** and receive output of the results of the application execution for display. In some embodiments, only the master node provides the functionality required to identify and provide address information associated with a server **106'** hosting a requested application.

In one embodiment, the server **106** provides functionality of a web server. In another embodiment, the server **106***a* receives requests from the client **102**, forwards the requests to a second server **106***b* and responds to the request by the client **102** with a response to the request from the server **106***b*. In still another embodiment, the server **106** acquires an enumeration of applications available to the client **102** and address information associated with a server **106** hosting an application identified by the enumeration of applications. In yet another embodiment, the server **106** presents the response to the request to the client **102** using a web interface. In one embodiment, the client **102** communicates directly with the server **106** to access the identified application. In another embodiment, the client **102** receives application output data, such as display data, generated by an execution of the identified application on the server **106**.

Referring now to FIG. **1**B, a network environment for delivering and/or operating a computing environment on a client **102** is depicted. In some embodiments, a server **106** includes an application delivery system **190** for delivering a computing environment or an application and/or data file to one or more clients **102**. In brief overview, a client **102** is in communication with a server **106** via network **104**, **104'** and appliance **105**. For example, the client **102** may reside in a remote office of a company, e.g., a branch office, and the server **106** may reside at a corporate data center. The client **102** comprises a client agent **120**, and a computing environment **15**. The computing environment **15** may execute or operate an application that accesses, processes or uses a data file. The computing environment **15**, application and/or data file may be delivered via the appliance **105** and/or the server **106**.

In some embodiments, the appliance **105** accelerates delivery of a computing environment **15**, or any portion thereof, to a client **102**. In one embodiment, the appliance **105** accelerates the delivery of the computing environment **15** by the application delivery system **190**. For example, the embodiments described herein may be used to accelerate delivery of a streaming application and data file processable by the application from a central corporate data center to a remote user location, such as a branch office of the company. In another embodiment, the appliance **105** accelerates transport layer traffic between a client **102** and a server **106**. The appliance **105** may provide acceleration techniques for accelerating any transport layer payload from a server **106** to a client **102**, such as: 1) transport layer connection pooling, 2) transport layer connection multiplexing, 3) transport control protocol buffering, 4) compression and 5) caching. In some embodiments, the appliance **105** provides load balancing of servers **106** in responding to requests from clients **102**. In other embodiments, the appliance **105** acts as a proxy or access server to provide access to the one or more servers **106**. In another embodiment, the appliance **105** provides a secure virtual private network connection from a first network **104** of the client **102** to the second network **104'** of the server **106**, such as an SSL VPN connection. In yet other embodiments, the appliance **105** provides application firewall security, control and management of the connection and communications between a client **102** and a server **106**.

In some embodiments, the application delivery management system **190** provides application delivery techniques to deliver a computing environment to a desktop of a user,

remote or otherwise, based on a plurality of execution methods and based on any authentication and authorization policies applied via a policy engine **195**. With these techniques, a remote user may obtain a computing environment and access to server stored applications and data files from any network connected device **100**. In one embodiment, the application delivery system **190** may reside or execute on a server **106**. In another embodiment, the application delivery system **190** may reside or execute on a plurality of servers **106***a*-**106***n*. In some embodiments, the application delivery system **190** may execute in a server farm **38**. In one embodiment, the server **106** executing the application delivery system **190** may also store or provide the application and data file. In another embodiment, a first set of one or more servers **106** may execute the application delivery system **190**, and a different server **106***n* may store or provide the application and data file. In some embodiments, each of the application delivery system **190**, the application, and data file may reside or be located on different servers. In yet another embodiment, any portion of the application delivery system **190** may reside, execute or be stored on or distributed to the appliance **200**, or a plurality of appliances.

The client **102** may include a computing environment **15** for executing an application that uses or processes a data file. The client **102** via networks **104**, **104'** and appliance **105** may request an application and data file from the server **106**. In one embodiment, the appliance **105** may forward a request from the client **102** to the server **106**. For example, the client **102** may not have the application and data file stored or accessible locally. In response to the request, the application delivery system **190** and/or server **106** may deliver the application and data file to the client **102**. For example, in one embodiment, the server **106** may transmit the application as an application stream to operate in computing environment **15** on client **102**.

In some embodiments, the application delivery system **190** comprises any portion of the Citrix Access Suite™ by Citrix Systems, Inc., such as the MetaFrame or Citrix Presentation Server™; any portion of the Citrix Delivery Center™ by Citrix Systems, Inc., such as the XenDesktop™, XenApp™, XenServer™, or NetScaler™; and/or any of the Microsoft® Windows Terminal Services manufactured by the Microsoft Corporation. In one embodiment, the application delivery system **190** may deliver one or more applications to clients **102** or users via a remote-display protocol or otherwise via remote-based or server-based computing. In another embodiment, the application delivery system **190** may deliver one or more applications to clients or users via steaming of the application.

In one embodiment, the application delivery system **190** includes a policy engine **195** for controlling and managing the access to, selection of application execution methods and the delivery of applications. In some embodiments, the policy engine **195** determines the one or more applications a user or client **102** may access. In another embodiment, the policy engine **195** determines how the application should be delivered to the user or client **102**, e.g., the method of execution. In some embodiments, the application delivery system **190** provides a plurality of delivery techniques from which to select a method of application execution, such as a server-based computing, streaming or delivering the application locally to the client **120** for local execution.

In one embodiment, a client **102** requests execution of an application program and the application delivery system **190** comprising a server **106** selects a method of executing the application program. In some embodiments, the server **106** receives credentials from the client **102**. In another embodiment, the server **106** receives a request for an enumeration of available applications from the client **102**. In one embodiment, in response to the request or receipt of credentials, the application delivery system **190** enumerates a plurality of application programs available to the client **102**. The application delivery system **190** receives a request to execute an enumerated application. The application delivery system **190** selects one of a predetermined number of methods for executing the enumerated application, for example, responsive to a policy of a policy engine. The application delivery system **190** may select a method of execution of the application enabling the client **102** to receive application-output data generated by execution of the application program on a server **106**. The application delivery system **190** may select a method of execution of the application enabling the local machine **10** to execute the application program locally after retrieving a plurality of application files comprising the application. In yet another embodiment, the application delivery system **190** may select a method of execution of the application to stream the application via the network **104** to the client **102**.

A client **102** may execute, operate or otherwise provide an application, which can be any type and/or form of software, program, or executable instructions such as any type and/or form of web browser, web-based client, client-server application, a thin-client computing client, an ActiveX control, or a Java applet, or any other type and/or form of executable instructions capable of executing on client **102**. In some embodiments, the application may be a server-based or a remote-based application executed on behalf of the client **102** on a server **106**. In one embodiments the server **106** may display output to the client **102** using any thin-client or remote-display protocol, such as the Independent Computing Architecture (ICA) protocol manufactured by Citrix Systems, Inc. of Ft. Lauderdale, Fla. or the Remote Desktop Protocol (RDP) manufactured by the Microsoft Corporation of Redmond, Wash. The application can use any type of protocol and it can be, for example, an HTTP client, an FTP client, an Oscar client, or a Telnet client. In other embodiments, the application comprises any type of software related to VoIP communications, such as a soft IP telephone. In further embodiments, the application comprises any application related to real-time data communications, such as applications for streaming video and/or audio.

In some embodiments, the server **106** or a server farm **38** may be running one or more applications, such as an application providing a thin-client computing or remote display presentation application. In one embodiment, the server **106** or server farm **38** executes as an application, any portion of the Citrix Access Suite™ by Citrix Systems, Inc., such as the MetaFrame or Citrix Presentation Server™; any portion of the Citrix Delivery Center™ by Citrix Systems, Inc., such as the XenDesktop™, XenApp™, XenServer™, or NetScaler™; and/or any of the Microsoft® Windows Terminal Services manufactured by the Microsoft Corporation. In one embodiment, the application is an ICA client, developed by Citrix Systems, Inc. of Fort Lauderdale, Fla. In other embodiments, the application includes a Remote Desktop (RDP) client, developed by Microsoft Corporation of Redmond, Wash. Also, the server **106** may run an application, which for example, may be an application server providing email services such as Microsoft Exchange manufactured by the Microsoft Corporation of Redmond, Wash., a web or Internet server, or a desktop sharing server, or a collaboration server. In some embodiments, any of the applications may comprise any type of hosted service or products, such as GoToMeeting™, GoToWebinar™, GoToMyPC™, or GoToAssist™ provided by Citrix Online Division, Inc. of

Santa Barbara, Calif., WebEx™ provided by WebEx, Inc. of Santa Clara, Calif., or Microsoft Office Live Meeting provided by Microsoft Corporation of Redmond, Wash.

The client **102**, server **106**, and appliance **105** may be deployed as and/or executed on any type and form of computing device, such as a computer, network device or appliance capable of communicating on any type and form of network and performing the operations described herein. FIGS. **1**C and **1**D depict block diagrams of a computing device **100** useful for practicing an embodiment of the client **102**, server **106** or appliance **105**. As shown in FIGS. **1**C and **1**D, each computing device **100** includes a central processing unit **101**, and a main memory unit **122**. As shown in FIG. **1**C, a computing device **100** may include a visual display device **124**, a keyboard **126** and/or a pointing device **127**, such as a mouse. Each computing device **100** may also include additional optional elements, such as one or more input/output devices **130***a*-**130***b* (generally referred to using reference numeral **130**), and a cache memory **140** in communication with the central processing unit **101**.

The central processing unit **101** is any logic circuitry that responds to and processes instructions fetched from the main memory unit **122**. In many embodiments, the central processing unit is provided by a microprocessor unit, such as: those manufactured by Intel Corporation of Mountain View, Calif.; those manufactured by Motorola Corporation of Schaumburg, Ill.; those manufactured by Transmeta Corporation of Santa Clara, Calif.; the RS/6000 processor, those manufactured by International Business Machines of White Plains, N.Y.; or those manufactured by Advanced Micro Devices of Sunnyvale, Calif. The computing device **100** may be based on any of these processors, or any other processor capable of operating as described herein.

Main memory unit **122** may be one or more memory chips capable of storing data and allowing any storage location to be directly accessed by the microprocessor **101**, such as Static random access memory (SRAM), Burst SRAM or Synch-Burst SRAM (BSRAM), Dynamic random access memory (DRAM), Fast Page Mode DRAM (FPM DRAM), Enhanced DRAM (EDRAM), Extended Data Output RAM (EDO RAM), Extended Data Output DRAM (EDO DRAM), Burst Extended Data Output DRAM (BEDO DRAM), Enhanced DRAM (EDRAM), synchronous DRAM (SDRAM), JEDEC SRAM, PC100 SDRAM, Double Data Rate SDRAM (DDR SDRAM), Enhanced SDRAM (ESDRAM), SyncLink DRAM (SLDRAM), Direct Rambus DRAM (DRDRAM), or Ferroelectric RAM (FRAM). The main memory **122** may be based on any of the above described memory chips, or any other available memory chips capable of operating as described herein. In the embodiment shown in FIG. **1**C, the processor **101** communicates with main memory **122** via a system bus **150** (described in more detail below). FIG. **1**C depicts an embodiment of a computing device **100** in which the processor communicates directly with main memory **122** via a memory port **103**. For example, in FIG. **1**D the main memory **122** may be DRDRAM.

FIG. **1**D depicts an embodiment in which the main processor **101** communicates directly with cache memory **140** via a secondary bus, sometimes referred to as a backside bus. In other embodiments, the main processor **101** communicates with cache memory **140** using the system bus **150**. Cache memory **140** typically has a faster response time than main memory **122** and is typically provided by SRAM, BSRAM, or EDRAM. In the embodiment shown in FIG. **1**C, the processor **101** communicates with various I/O devices **130** via a local system bus **150**. Various busses may be used to connect the central processing unit **101** to any of the I/O devices **130**, including a VESA VL bus, an ISA bus, an EISA bus, a MicroChannel Architecture (MCA) bus, a PCI bus, a PCI-X bus, a PCI-Express bus, or a NuBus. For embodiments in which the I/O device is a video display **124**, the processor **101** may use an Advanced Graphics Port (AGP) to communicate with the display **124**. FIG. **1**D depicts an embodiment of a computer **100** in which the main processor **101** communicates directly with I/O device **130** via HyperTransport, Rapid I/O, or InfiniBand. FIG. **1**D also depicts an embodiment in which local busses and direct communication are mixed: the processor **101** communicates with I/O device **130** using a local interconnect bus while communicating with I/O device **130** directly.

The computing device **100** may support any suitable installation device **116**, such as a floppy disk drive for receiving floppy disks such as 3.5-inch, 5.25-inch disks or ZIP disks, a CD-ROM drive, a CD-R/RW drive, a DVD-ROM drive, tape drives of various formats, USB device, hard-drive or any other device suitable for installing software and programs such as any client agent **120**, or portion thereof. The computing device **100** may further comprise a storage device **128**, such as one or more hard disk drives or redundant arrays of independent disks, for storing an operating system and other related software, and for storing application software programs such as any program related to the client agent **120**. Optionally, any of the installation devices **116** could also be used as the storage device **128**. Additionally, the operating system and the software can be run from a bootable medium, for example, a bootable CD, such as KNOPPIX®, a bootable CD for GNU/Linux that is available as a GNU/Linux distribution from knoppix.net.

Furthermore, the computing device **100** may include a network interface **118** to interface to a Local Area Network (LAN), Wide Area Network (WAN) or the Internet through a variety of connections including, but not limited to, standard telephone lines, LAN or WAN links (e.g., 802.11, T**1**, T**3**, 56 kb, X.25), broadband connections (e.g., ISDN, Frame Relay, ATM), wireless connections, or some combination of any or all of the above. The network interface **118** may comprise a built-in network adapter, network interface card, PCMCIA network card, card bus network adapter, wireless network adapter, USB network adapter, modem or any other device suitable for interfacing the computing device **100** to any type of network capable of communication and performing the operations described herein.

A wide variety of I/O devices **130***a*-**130***n* may be present in the computing device **100**. Input devices include keyboards, mice, trackpads, trackballs, microphones, and drawing tablets. Output devices include video displays, speakers, inkjet printers, laser printers, and dye-sublimation printers. The I/O devices **130** may be controlled by an I/O controller **123** as shown in FIG. **1**C. The I/O controller may control one or more I/O devices such as a keyboard **126** and a pointing device **127**, e.g., a mouse or optical pen. Furthermore, an I/O device may also provide storage **128** and/or an installation medium **116** for the computing device **100**. In still other embodiments, the computing device **100** may provide USB connections to receive handheld USB storage devices such as the USB Flash Drive line of devices manufactured by Twintech Industry, Inc. of Los Alamitos, Calif.

In some embodiments, the computing device **100** may comprise or be connected to multiple display devices **124***a*-**124***n*, which each may be of the same or different type and/or form. As such, any of the I/O devices **130***a*-**130***n* and/or the I/O controller **123** may comprise any type and/or form of suitable hardware, software, or combination of hardware and software to support, enable or provide for the connection and

use of multiple display devices **124***a*-**124***n* by the computing device **100**. For example, the computing device **100** may include any type and/or form of video adapter, video card, driver, and/or library to interface, communicate, connect or otherwise use the display devices **124***a*-**124***n*. In one embodiment, a video adapter may comprise multiple connectors to interface to multiple display devices **124***a*-**124***n*. In other embodiments, the computing device **100** may include multiple video adapters, with each video adapter connected to one or more of the display devices **124***a*-**124***n*. In some embodiments, any portion of the operating system of the computing device **100** may be configured for using multiple displays **124***a*-**124***n*. In other embodiments, one or more of the display devices **124***a*-**124***n* may be provided by one or more other computing devices, such as computing devices **100***a* and **100***b* connected to the computing device **100**, for example, via a network. These embodiments may include any type of software designed and constructed to use another computer's display device as a second display device **124***a* for the computing device **100**. One ordinarily skilled in the art will recognize and appreciate the various ways and embodiments that a computing device **100** may be configured to have multiple display devices **124***a*-**124***n*.

In further embodiments, an I/O device **130** may be a bridge **170** between the system bus **150** and an external communication bus, such as a USB bus, an Apple Desktop Bus, an RS-232 serial connection, a SCSI bus, a FireWire bus, a FireWire 800 bus, an Ethernet bus, an AppleTalk bus, a Gigabit Ethernet bus, an Asynchronous Transfer Mode bus, a HIPPI bus, a Super HIPPI bus, a SerialPlus bus, a SCI/LAMP bus, a FibreChannel bus, or a Serial Attached small computer system interface bus.

A computing device **100** of the sort depicted in FIGS. **1**C and **1**D typically operate under the control of operating systems, which control scheduling of tasks and access to system resources. The computing device **100** can be running any operating system such as any of the versions of the Microsoft® Windows operating systems, the different releases of the Unix and Linux operating systems, any version of the Mac OS® for Macintosh computers, any embedded operating system, any real-time operating system, any open source operating system, any proprietary operating system, any operating systems for mobile computing devices, or any other operating system capable of running on the computing device and performing the operations described herein. Typical operating systems include: WINDOWS 3.x, WINDOWS 95, WINDOWS 98, WINDOWS 2000, WINDOWS NT 3.51, WINDOWS NT 4.0, WINDOWS CE, WINDOWS XP, WINDOWS VISTA, and WINDOWS 7, all of which are manufactured by Microsoft Corporation of Redmond, Wash.; MacOS, manufactured by Apple Computer of Cupertino, Calif.; OS/2, manufactured by International Business Machines of Armonk, N.Y.; and Linux, a freely-available operating system distributed by Caldera Corp. of Salt Lake City, Utah, or any type and/or form of a Unix operating system, among others.

In other embodiments, the computing device **100** may have different processors, operating systems, and input devices consistent with the device. For example, in one embodiment the computer **100** is a Treo **180**, **270**, **1060**, **600** or **650** smart phone manufactured by Palm, Inc. In this embodiment, the Treo smart phone is operated under the control of the PalmOS operating system and includes a stylus input device as well as a five-way navigator device. In another embodiment, the computer **100** is an iPhone smart phone manufactured by Apple Computers, Inc. In this embodiment, the iPhone is operated under the control of the iPhone OS operating system and includes a multi-touch screen interface. Moreover, the computing device **100** can be any workstation, desktop computer, laptop or notebook computer, server, handheld computer, mobile telephone, any other computer, or other form of computing or telecommunications device that is capable of communication and that has sufficient processor power and memory capacity to perform the operations described herein.

B. DNS Resolver Architecture

FIG. **2** describes an embodiment of a DNS resolver **200** residing on a server, client, or intermediary. As shown in FIG. **2**, the DNS resolver **200** includes a DNS query identifier **201**, a memory cache **202**, a transaction identifier generator **203**, a DNS query generator **205**, a comparator **206** for comparing requests and responses, and a DNS response generator **207**. The DNS resolver **200** may also receive input numbers from a random number generator **204**. The DNS resolver **200** receives DNS resolution requests **208** from hardware or software on the client, server, or intermediary, or from any hardware or software on another client, server, or intermediary. The DNS resolver **200** transmits DNS query messages **209** to DNS name servers, and in turn DNS receives response messages **210**. The DNS response messages **210** can also arrive from different sources than the name servers. The DNS resolver **200** also transmits DNS resolution responses **211** to the hardware or software on the same or different client, server, or intermediary that sent the DNS resolution request **208**. The simplified architecture shown is provided for illustration purposes only and is not intended to be limiting.

A DNS resolver **200** comprises any type or form of logic, operations or functions to resolve a domain name. The DNS resolver **200** may comprise any combination of software and hardware. The DNS resolver **200** may comprise a library, service, daemon, process, function, or subroutine. Although the random number generator **204** shown in FIG. **2** is external to the DNS resolver **200**, in some embodiments the DNS resolver **200** may also include the random number generator **204**. In other embodiments, the random number generator **204** may be on another software or hardware system. The DNS resolver **200** may include functionality for transmitting and receiving data in any format or protocol, such as Internet Protocol. As such, in some embodiments, the DNS resolver **200** may include hardware or communicate with hardware capable of performing this functionality. In other embodiments, the DNS resolver **200** may operate within a virtual machine and may include or communicate with virtual hardware.

The DNS query identifier **201** comprises one or more programs, tasks, services, processes or executable instructions to provide logic, rules, functions, or operations for receiving and handling a DNS resolution request **208**. The DNS query identifier **201** checks the resolver cache **202** to determine if a previously-received DNS response message corresponding to the DNS resolution request **208** has been stored in the resolver cache **202**. If so, the DNS response generator **207** transmits a DNS resolution response **211** to the requester using the previously-received DNS response message in the resolver cache **202**. If the answer is unknown, the DNS query identifier **201** checks that the DNS resolution request **208** is a fully qualified domain name query or unqualified multi-label domain name query. If the DNS query identifier **201** determines that the DNS resolution request **208** is not a fully qualified domain name query or unqualified multi-label query, the DNS query identifier **201** consults the cache **202** for a suffix search list. If a suffix search list does not reside in the cache **202**, the DNS query identifier **201** appends a global DNS suffix to the DNS resolution request **208**. If a suffix search list does reside in the cache **202**, the DNS query

identifier **201** appends a primary DNS suffix to the DNS resolution request **208**. The DNS query identifier **201** consults the cache **202** to determine the IP address or addresses of a domain name server or servers from which to request an answer. In some embodiments, the DNS query identifier **201** consults the cache **202** to determine the port or ports of the domain name server or servers.

In some embodiments, the domain name requested is an ASCII name. In other embodiments, the domain name requested is part of the international domain name system and is encoded in Row-based ASCII Compatible Encoding (RACE) or punycode. The international domain name may be encoded by the DNS resolver **200** or may be already encoded when received by the DNS query identifier **201**. In some embodiments, the DNS query identifier **201** consults the cache **202** to determine if each domain name server to be contacted is compliant with IETF RFC 4343 (Domain Name System Case Insensitivity Clarification). In some embodiments if a domain name server is RFC 4343 compliant, the DNS query identifier **201** may retain mixed capitalization of the domain name as received in the DNS resolution request **208**. In other embodiments where a domain name server is RFC 4343 compliant, the DNS query identifier **201** may encode random capitalization in the domain name.

The cache **202** may comprise any type and form of data structure, implemented in any combination of hardware and software. In some embodiments, the cache **202** may comprise a database, a flat file, dictionary, registry, index, lookup table, or any other repository capable of storing DNS resource records in any format. The cache **202** may include any associated logic and control functions for recording and obtaining DNS resource records. Once DNS resource records are stored in the cache **202**, the DNS resolver **200** can use the cached copy rather than re-transmitting a DNS query message for the resource, thereby reducing access time and use of network bandwidth. In some embodiments, the cache **202** may include an associated memory element, including RAM, Flash memory, or a portion of a disk drive. In other embodiments, the cache **202** may comprise a data object in main memory unit **122** or cache memory **140**, discussed above in connection with FIGS. **1**C and **1**D, or any combination thereof. In still other embodiments, the cache **202** may comprise any type of integrated circuit, such as a Field Programmable Gate Array (FPGA) or a Programmable Logic Device (PLD). In some embodiments, the cache **202** may have a fixed maximum size. In other embodiments, there may be no such limitation. Furthermore, the cache **202** may include logic or functionality for invalidating or removing cached DNS resource records based on the expiration of a time period or upon receipt of an invalidation command from the DNS resolver **200**. The logic or functionality may allow invalidation or removal of single records, groups of records, or all records residing in the cache **202**.

The random number generator **204** may comprise any type and form of software or hardware, or any combinations thereof, for generating random or pseudo-random numbers. In some embodiments, the random number generator **204** is within the DNS resolver **200**. In other embodiments, the random number generator **204** is in the same client, server, or intermediary as the DNS resolver **200**. In still other embodiments, the random number generator **204** is separate from the client, server, or intermediary that includes the DNS resolver **200**. In these embodiments, the random number generator **204** may communicates with the DNS resolver over any type and form of network or communications device or protocol. The random number generator **204** generates and transmits random or pseudo-random numbers to the transaction identifier generator **203** in the DNS resolver **200**. The random or pseudo-random numbers can be of any length. In some embodiments, the number length is at least as long as the length of the requested domain name. For example, under IETF RFC 1034, the maximum length of a fully qualified domain name is 255 octets, or 2040 bits. In other embodiments, the random or pseudo-random number may be shorter or longer. In some embodiments, the random number generator **204** may generate a new random or pseudo-random number for each new DNS resolution request received by the DNS resolver **200**. In other embodiments, the same random or pseudo-random number may be used for multiple transaction identifiers, reducing the need for new random or pseudo-random numbers for each request. In some embodiments, the random number generator **204** may generate a new random or pseudo-random number at a predetermined frequency. Higher frequencies may result in higher computation costs, while lower frequencies may result in less resistance to attack. In other embodiments, the random number generator **204** may generate a new random or pseudo-random number in response to an event. For example, the random number generator **204** may generate a new random number in response to every fifth or tenth DNS resolution request **208** received by the DNS resolver **200**. For another example, the random number generator **204** may generate a new random number in response to every invalidation of a DNS response record in the cache **202**. In these embodiments, the event that causes the random number generator **204** to generate a new random or pseudo-random number may be any event capable of triggering such functionality within the random number generator **204**, such as closing or opening a switch, changing a value in a memory register, executing a function call, or accessing a memory location.

The transaction identifier generator **203** comprises a process, logic, function, service, task, subroutine, or executable instructions for creating or providing a transaction identifier, such as for a DNS query. As defined by IETF RFC 1035, the DNS transaction identifier is a 16-bit field in the header of DNS query message **209**. However, in other protocols, the transaction identifier may be of different lengths or formats. In some embodiments, the queried domain name server responds to a DNS query message **209** with a DNS response message **210** including an identical DNS transaction identifier, associating the response with the query. In some embodiments, however, there may be multiple DNS query messages **209** sent by the DNS resolver **200** in response to multiple DNS resolution requests **208** before any DNS response message **210** is received. The transaction identifier allows the DNS resolver **200** to identify which outstanding request is associated with which response. The sent and received transaction identifiers are compared by the response/request comparator **206**, and if the received transaction identifier matches no outstanding DNS query message **209**, the DNS response message **210** is discarded by the DNS resolver **200**. If the received transaction identifier does match an outstanding DNS query message **209**, the DNS response message **210** is passed to the DNS response generator **207**. To generate a transaction identifier, the transaction identifier generator **203** may perform a cryptographic hashing function using inputs comprising the random or pseudo-random number received from the random number generator **204**, the IP address of the domain name server to be queried, and the domain name or a portion of the domain name requested. In some embodiments, the inputs to the transaction identifier generator **203** hash function may include the port number of the domain name server to be queried. In further embodiments, the inputs to the hash function may include the domain name encoded with

capitalization. In yet further embodiments, the inputs to the hash function may include the domain name encoded in punycode or RACE. The cryptographic hash function used to create the transaction identifier may be any hash function, with or without collision resistance, such as MD5, SHA-1, SHA-256, or any other known or currently unknown hash function or combination of hash functions. In some embodiments, the output of the cryptographic hash function may be compressed to 16 bits. In other embodiments, the output may be truncated or shortened through any other means to 16 bits. In still other embodiments, the output of the cryptographic hash function may be shortened, truncated, or extended in any means to achieve the length of the transaction identifier required by the protocol format of the DNS query message **209**. In further embodiments, the cryptographic hashing function may be collision resistant or collision free.

The DNS query generator **205** comprises a process, logic, function, service, task, subroutine, or executable instructions for creating any type and form of DNS query message **209**. The DNS query message **209** may be created in compliance with the standard DNS query message format defined in IETF RFC 1035, or may be created in any other desired format. The DNS query message **209** may include a transaction identifier received from the transaction identifier generator **203**. The DNS query message **209** may include a question entry of the domain name queried received from the DNS query identifier **201**. The question entry may include a domain name, class, and type, as described in RFC 1035, or may contain more or fewer information as dictated by the query message format. The DNS query message **209** may be transmitted by the DNS resolver **200** to the address or addresses of the domain name server or servers selected by the DNS query identifier **201** from the cache **202**. In some embodiments, the DNS query message **209** may be stored in a memory element associated with the response/request comparator **206**. In other embodiments, the DNS query message **209** may be stored in a memory element associated with the DNS query generator **205**, the cache **202**, or any other memory element associated with or accessible by the DNS resolver **200**. The stored DNS query message may be marked, flagged or recorded in such a way as to identify the query as an outstanding query. In such embodiments, the DNS resolver **200** may include functionality for invalidating or removing markings or flags or deleting the stored DNS query once it is no longer outstanding. In some embodiments, the DNS query message **209** may be transmitted by the DNS resolver **200** multiple times, such as in response to expiration of a time-to-live period.

The response/request comparator **206** comprises a process, logic, function, service, task, subroutine, or executable instructions for comparing a DNS response message **210** with a DNS query message **209**. For example, the response/request comparator **206** may compare a DNS response message **210** received by the DNS resolver **200** with a DNS query message **209** sent by the DNS resolver **200**. In some embodiments, the response/request comparator **206** may compare a received DNS response message **210** with a DNS query message marked as an outstanding query as discussed above. The response/request comparator **206** may check if the DNS response message **210** has apparently come from the same IP address and port of the domain name server to whom the DNS query message **209** was sent. Furthermore, in some embodiments, the response/request comparator **206** may check if the DNS response message **210** has the same transaction identifier as the DNS query message **209** that was sent. In these embodiments, if the response/request comparator **206** determines that the DNS response message **210** does not match the DNS query message **209**, the comparator **206** disregards the DNS response message **210**. In further embodiments, where the domain name server queried is RFC 4343 compliant, the response/request comparator **206** may compare capitalization of the domain name in the transmitted DNS query message **209** and received DNS response message **210**. In these embodiments, the response/request comparator **206** may instruct the DNS resolver **200** to disregard the DNS response message **210**, responsive to a determination that the capitalization does not match. If the response/request comparator **206** identifies the DNS response message **210** as matching an outstanding DNS query message **209**, it may, in some embodiments, pass the DNS response message **210** as a validated response to the DNS response generator **207**. The response/request comparator **206** may also invalidate or remove markings or flags or delete memory entries identifying the DNS query message **209** as outstanding, or instruct another process to do so.

The DNS response generator **207** comprises a process, logic, function, service, task, subroutine, or executable instructions for generating and/or sending a DNS resolution response **211**. The DNS response generator **207** may receive a validated DNS response message **210** from the response/request comparator **206**. In some embodiments, the DNS response generator **207** may inspect the DNS response message **210** to determine if it is fully responsive to the DNS resolution request **208**. A fully responsive message contains the final address sought. For example, a request for the address of www.example.com may return a response that the domain name www.example.com is located at 208.77.188.166. A non-fully responsive message to the query for the address of www.example.com may return a response that the domain name server for example.com is located at 208.77.188.1, but be silent on the address of www.example-.com. In some embodiments, if the DNS response message **210** is not fully responsive to the DNS resolution request **208**, the DNS response generator **207** may record any partial response or additional name server information in the cache **202**. Furthermore, the DNS query identifier **201** may create a new iteration of the query using the partial response or additional name server information. In some embodiments, if the DNS response message **210** is fully responsive to the DNS resolution request **208**, the DNS response generator **207** may record the response in the cache **202**. The DNS response generator **207** may send a DNS resolution response **211** to the originally requesting software or hardware on the same or different client, server, or intermediary.

The DNS resolution request **208** is a data packet or packets comprising a DNS name to be resolved. The DNS resolution request **208** may include a fully qualified domain name or a portion of a domain name; a DNS query type; and a DNS query class. In some embodiments, the DNS resolution request **208** comes from a software or hardware system on the same client, server, or intermediary. In other embodiments, the DNS resolution request **208** comes from a software or hardware system on another client, server, or intermediary.

The DNS query message **209** is a data packet or packets comprising a request to a name server to identify a domain name. The DNS query message **209** may include a transaction identifier and a question entry, as discussed above in connection with the DNS query generator **205**. The DNS query message **209** may be transmitted over TCP, UDP, or any other protocol known or currently unknown for allowing communication over a network.

The DNS response message **210** is a data packet or packets comprising a response to a DNS query message **210**. The DNS query message **210** may include a transaction identifier. The transaction identifier may be identical, similar, or differ-

ent from the transaction identifier of an outstanding DNS query message **209**. The DNS query message **210** may include a response entry. The response entry may include a domain name, class, or type as described in RFC 1035, or may contain more or fewer information as dictated by the query message format. The response entry may also include additional information, such as the address of an authoritative name server for the domain name requested. The DNS query message **210** may be transmitted over TCP, UDP, or any other type and form of protocol for allowing communication over a network.

The DNS resolution response **211** is a data packet or packets comprising a response to a DNS resolution request **208**. The DNS resolution response **211** may include a domain name and IP address corresponding to the domain name requested in the DNS resolution request **208**. In some embodiments, the DNS resolution response **211** may include a message that the domain name could not be located.

C. DNS Query Generation

FIG. **3** describes an embodiment of steps for a method of generating a transaction identifier for a DNS query message. In brief overview, at step **300**, a request is received to resolve a domain name. At step **301**, the request is parsed and the IP address and port of a domain name server with the domain in its zone of authority or a delegated zone is determined. At step **302**, the cache is consulted to determine if the domain name server to be queried rewrites or normalizes mixed-capitalization domain names; if the domain name server does not do so, capitalization shifts may be made in the characters of the requested domain name. At step **303**, a random or pseudo-random number is generated for a salt input to a cryptographic hashing function. At step **304**, a transaction identifier is created as an output of the cryptographic hashing function. At step **305**, the DNS query message is created. At step **306**, the DNS query message is transmitted to the domain name server to be queried. In some embodiments, this process may be repeated multiple times for the same request to query multiple domain name servers. In further embodiments, this process may be repeated iteratively or recursively where DNS answers fail to fully answer the DNS request, but indicate a more authoritative name server to query.

In further details, at step **300**, the DNS resolver may receive a request to resolve a domain name. In some embodiments, this request may come from a web browser or similar application. In other embodiments, this request may come from a kernel service, function, daemon, or other executable code residing in hardware, software, or any combination thereof. In some embodiments, the source of the request may be on the same client, server, or intermediary as the DNS resolver. In other embodiments, the source of the request may be from a different client, server, or intermediary on the same or a different network. The DNS request may be for a full or partial domain name. The DNS request may include a domain class and domain type. In some embodiments, the DNS request may include wildcard characters in the name or class or type, signifying that all records relevant to a domain name or partial domain name or class or type are being requested. If a relevant fully-responsive DNS record resides in the DNS resolver's cache, a response containing the information in the DNS record may be returned to the requestor. In such a case, no further steps of generating a DNS query may need to be taken.

At step **301**, the DNS resolver may select a domain name server to query from the index of name servers in the cache. In some embodiments, the DNS resolver may select a preferred name server. In other embodiments, the DNS resolver may select the name server with the narrowest zone of authority containing the requested domain listed in the index of name servers in the cache. In further embodiments, the DNS resolver may select the name server for the root zone. Once a name server to be queried has been selected, the DNS resolver retrieves the name server's IP address and port number from the index of name servers in the cache.

At step **302**, the DNS resolver may consult the cache to determine if the domain name server to be queried rewrites or normalizes responses. In some embodiments, the DNS resolver may determine that the name server rewrites responses by the presence and content of additional data fields in the cached resource record. In other embodiments, the DNS resolver may determine that the name server rewrites responses by comparing prior mixed capitalization domain name queries to the name server with responses from the same name server for preservation of capitalization. If the domain name server to be queried does not rewrite or normalize responses, in some embodiments the DNS resolver may shift any or all of the characters of the domain name between upper and lower case.

At step **303**, the random number generator generates a random or pseudo-random number. In one embodiment, the random number generator passes the random or pseudo-random number to the transaction identifier generator. In other embodiments, the transaction identifier generator obtains or retrieves the random or pseudo-random number from a memory element associated with the random number generator. In some embodiments, a new random or pseudo-random number is passed to the cryptographic hashing function for each new DNS query. In other embodiments, a random or pseudo-random number may be reused for multiple DNS queries. In further embodiments, the random or pseudo-random number may be updated at a predetermined frequency. In other embodiments, the random or pseudo-random number may be updated in response to an event, as discussed above in connection the random number generator **204** and FIG. **2**.

At step **304**, the transaction identifier generator performs a cryptographic hashing function on inputs comprising the random or pseudo-random number, the IP address of the domain name server to be queried, and the domain name or a portion of the domain name requested. The cryptographic hashing function may be any hash function or combination of hash functions, including MD5, SHA-1, SHA-256, or any other hash function or functions currently known or unknown. For example, in one embodiment, the transaction identifier generator may append the input bits of the random number, the IP address of the domain name server and the domain name to create a string of bits. In this example embodiment, the transaction identifier generator may use a combination of addition, exclusive-or (XOR), and constant rotations of adjacent bits or groups of bits to convert the string of bits into a cryptographic hash of length specified by the size of the transaction identifier field of the protocol. For instance, the IETF RFC 1035 standard specifies a 16-bit transaction identifier for DNS queries. However, the transaction identifier generator may be configured to output a hash of any length required by the network protocol in use. Although the illustrative example describes appending the input bits to create a string, the transaction identifier generator may use appending, multiplying, adding, subtracting, XORing, or any other method known to those skilled in the art.

At step **305**, the DNS query generator creates a DNS query using the requested domain name, domain class and/or domain type received at step **300**, and the transaction identifier generated at step **304**. As mentioned above at step **300**, the DNS request may be for a full or partial domain name, and wildcard characters may be present in the name or class or

type, signifying that all records relevant to a domain name or partial domain name or classes or types are being requested. In one embodiment, the DNS query generator may create an RFC 1035 standard DNS query, comprising a header followed by a question. In this embodiment, the header may include the transaction identifier generated at step **304**, a query flag, an opcode specifying the type of query, and a recursion flag. The question may include the domain name, the domain type and domain class. In some embodiments, the DNS query generator may also create a message compliant with the relevant protocol, such as TCP or UDP, with the DNS query as a payload. In other embodiments, the DNS query generator may pass the query as a payload to another process or service acting at the application or transport layer.

At step **306**, the DNS resolver transmits the DNS query to the domain name server selected at step **301**. The domain name server may be on the same network as the DNS resolver or on a different network. In some embodiments, the DNS resolver transmits the DNS query directly. In other embodiments, the DNS resolver passes the DNS query to another process or service responsible for handling network communications, such as a network driver.

In many embodiments, the DNS query generated at step **305** complies with IETF standard DNS protocol. In these embodiments, the domain name server may recognize the generated DNS query as a standard DNS query. In such embodiments, the functionality of DNS name resolution may be performed without alteration to hardware or software on the domain name server, client, or intermediary. In some embodiments, the DNS protocol used or supported by the client and/or server do not need to change to support any of the functionality or operations described herein. Such compliance with DNS protocols, such as with IETF standards, may prevent compatibility issues between the client, server, and intermediary. In other embodiments, the DNS query may be generated to comply with other standards, including Extended DNS (described in RFC 2671) and DNS Security Extensions (described in RFC 2535). In yet other embodiments, the DNS query may be encrypted. In one such embodiment, the encryption protocol may be DNSCurve. In other such embodiments, any other encryption protocol or combinations of protocols may be used. In still other embodiments, the DNS query may be generated in a proprietary format, and may require hardware or software changes to the client, server, or intermediary.

While the invention has been particularly shown and described with reference to specific embodiments, it should be understood by those skilled in the art that various changes in form and detail may be made therein without departing from the spirit and scope of the invention as defined by the following claims.

What is claimed:

1. A method for generating a Domain Name Service (DNS) query to improve resistance against a DNS attack, the method comprising:

a) receiving, by a DNS resolver configured on a device, a request to resolve a domain name;

b) identifying, by the DNS resolver, the domain name, an internet protocol address of a DNS server, and a port of the DNS server;

c) generating a transaction identifier for a DNS query by applying a one-way hash function to an input of a predetermined random number, the internet protocol address of the DNS server, the port of the DNS server, and the domain name, the input of the domain name comprising a portion of the domain name to be resolved; and

d) transmitting, by the DNS resolver, the DNS query for the domain name to the DNS server, the DNS query identified by the generated transaction identifier.

2. The method of claim **1**, wherein step (c) further comprises changing the predetermined random number at a predetermined frequency.

3. The method of claim **1**, wherein step (c) further comprises changing the predetermined random number in response to an event.

4. The method of claim **1**, wherein step (c) further comprises generating by the one-way hash function the same transaction identifier for DNS queries to resolve the same domain name transmitted to the same DNS server.

5. The method of claim **1**, wherein step (c) further comprises encoding one or more fields of the DNS request and using the encoded one or more fields as input to the one-way hash function to generate the transaction identifier.

6. The method of claim **1**, wherein step (c) further comprises encoding the domain name by capitalizing one or more characters of the domain name and generating the transaction identifier by using the encoded domain name as the input of the domain name to the one-way hash function.

7. The method of claim **1**, wherein step (c) further comprises encoding the domain name by using one of a punycode and a RACE encoding scheme.

8. The method of claim **1**, further comprising determining, by the DNS resolver, that the DNS server is one of rewriting or normalizing responses and in response to the determination not encoding a portion of the DNS query.

9. The method of claim **1**, further comprising determining, by the DNS resolver, that the destination is not rewriting responses and in response to the determination encoding a portion of the DNS query and including the encoded portion in the transaction identifier.

10. The method of claim **1**, wherein step (c) further comprises communicating by the DNS resolver the input of the internet protocol address of the destination and the domain name to a transaction identifier generator.

11. A system for generating a Domain Name Service (DNS) query to improve resistance against a DNS attack, the system comprising:

a computing device, comprising a processor executing a DNS resolver and a transaction identifier generator,

wherein the DNS resolver is configured to receive a request to resolve a domain name and identify the domain name, an internet protocol address of a destination of the request, and a port of the destination of the request;

wherein the transaction identifier generator is configured to generate a transaction identifier by applying a one-way hash function to an input of a predetermined random number, the internet protocol address of the destination, the port of the destination, and the domain name, the input of the domain name comprising a portion of the domain name to be solved; and

wherein the DNS resolver is further configured to form the DNS query using the generated transaction identifier and transmit the DNS query for the domain name to the destination.

12. The system of claim **11**, wherein the transaction identifier generator is further configured to change the predetermined random number at a predetermined frequency.

13. The system of claim **11**, wherein the transaction identifier generator is further configured to change the predetermined random number in response to an event.

14. The system of claim **11**, wherein the transaction identifier generator is further configured to generate the same

transaction identifier for inputs identifying the same domain name and the same destination.

15. The system of claim **11**, wherein the DNS resolver is further configured to encode one or more fields of the DNS request and communicate the encoded one or more fields as input to the transaction identifier generator to generate the transaction identifier.

16. The system of claim **11**, wherein the DNS resolver is further configured to encode the domain name by capitalizing one or more characters of the domain name and communicate the encoded domain name as the input of the domain name to the transaction identifier generator.

17. The system of claim **11**, wherein the DNS resolver is further configured to encode the domain name by using one of a punycode and a RACE encoding scheme.

18. The system of claim **11**, wherein the DNS resolver is further configured to determine that the destination is one of rewriting or normalizing responses and in response to the determination does not encode a portion of the DNS query.

19. The system of claim **11**, wherein the DNS resolver is further configured to determine that the destination is not rewriting responses and in response to the determination encodes a portion of the DNS query and communicate the encoded portion as input to the transaction identifier generator to generate the transaction identifier.

20. The system of claim **11**, wherein the computing device executing the DNS resolver is one of a client, a server and an intermediary.

* * * * *