修士論文

トンネリング抑止を目的とした
分散ハッシュテーブルを利用したDNSに関する研究

高須賀 昌烈

2020年1月28日

奈良先端科学技術大学院大学
先端科学技術研究科

本論文は奈良先端科学技術大学院大学先端科学技術研究科に
修士 (工学) 授与の要件として提出した修士論文である。

高須賀 昌烈

審査委員：

門林 雄基 教授　（主指導教員）
笠原 正治 教授　（副指導教員）
林 優一 教授　（副指導教員）
妙中 雄三 准教授　（副指導教員）

トンネリング抑止を目的とした分散ハッシュテーブルを利用したDNSに関する研究*

高須賀 昌烈

内容梗概

巧妙化するサイバー攻撃の手法の中に，攻撃通信を無害な通信に偽装することで検知を迂回する手法がある．DNSトンネリングと呼ばれる手法は，そのような秘匿通信手法の中で最も広く利用されている．DNSトンネリングは，DNSクエリパケットのドメイン名あるいは応答パケットのリソースレコードにデータを含めることによって，データを転送するという単純な仕組みで動作する．この手法に対して，従来は，サブドメインの長さおよびエントロピー，トラフィック頻度などを特徴量とする閾値や悪性モデルを推定する検知アプローチが取られてきた．しかし，転送データ量の削減やパケット間のインターバルを長期化させるといった転送効率を下げる手法を用いることで，既存の検知アプローチを迂回される脅威がある．

この脅威に対して，本研究ではDNSトンネリングの発生を抑止する名前解決システムを提案する．システムに採用したアプローチは，権威サーバの機能を分離させることでスタブリゾルバからサーバへのクエリ透過を抑制し，このメカニズムによってDNSトンネリングの発生を抑止することができる．評価では，実装した提案システムのプロトタイプ上でトンネリング通信をシミュレーションさせることによって，DNSトンネリングの通信抑止に有効であることを示した．また，既存システムとの比較に基づいた特性の評価を行った結果から，提案システムにはトラフィック量の削減と高速な名前解決という優位性があることを示した．

*奈良先端科学技術大学院大学 先端科学技術研究科 修士論文, 2020年1月28日.

キーワード

ドメインネームシステム，DNS トンネリング，分散ハッシュテーブル，フルメッシュネットワーク

A Study of Domain Name System using Distributed Hash Table for Tunneling Deterrence*

Shoretsu Takasuka

Abstract

In well-organised cyber-attacks, camouflage techniques are used to bypassing the monitoring or analysis systems. DNS Tunneling is the most popular method in those covert network techniques. It's utilised a simple concept which transfers data through the Qname field in the DNS query packet. Many solutions have been proposed to counter against this tunneling technique. However, those previous research used some tactics such as subdomain's length or entropy, and frequency of traffic by analysing tunneling traffic. The previous countermeasure were based on the detection apploach estimated threshold or malicious models. However those detection apploachies are exposed to the menace of the being bypassed. For example, the malicious user can bypass by reducing the transfer data in a packet or leaving times between query packets and response packets.

In this study, I propose name resolution system based on distributed hash table for DNS tunneling deterrence. The core idea is the division of authorization server function into contents management and records cofiguration. Following this approach, the transparency from clients to servers which malicious user configured is restricted, tunneling queries don't reach the server in the end. I implemented prototype system of the proposed design for the evaluation, and verified the

*Master's Thesis, Graduate School of Information Science, Nara Institute of Science and Technology, January 28, 2020.

effectiveness for the tunneling deterrence by simulation tunneling test on the prototype. Besides the results, I evaluated the features of the proposed system shows my proposed system is expected for a reduction of name resolution traffic and faster name resolution.

Keywords:

Domain Name System(DNS), DNS Tunneling, Distributed Hash Table, Full Mesh Network

目 次

図 目 次

表 目 次

アルゴリズム目次

1. 序論

1.1 背景

増加し続けるサイバー攻撃に対して，現在多くの組織は，SIEM[*1]のようなネットワークトラフィックを監視するシステムから発せられるアラートを処理することでセキュリティの脅威に対処している．一方，機密情報の奪取や諜報活動を行う攻撃者は，そのような監視システムを迂回するために秘匿通信手法[*2]を用いることが知られている [1]．このような攻撃に対して，検知の閾値や悪性モデルを調整するアプローチが考えられるが，誤検知とのトレードオフの関係にある．

秘匿通信の代表的な手法に DNS トンネリングがある．DNS(Domain Name System) は，IP アドレスをはじめとしたドメイン名に関連づけられたリソースレコードを解決するシステムである．この名前解決の機能によって，ユーザは識別しづらい IP アドレスを直接利用することなくサーバのリソースにアクセスすることができる．インターネットの利活用において，名前解決の通信はメールの送受信やウェブ検索などの通信に先立って行われる．すなわち，DNS のトラフィックをフィルタリングすることはインターネットの利活用に大きな影響を及ぼすため，容易にフィルタリングを行うことが困難であるという特性がある．DNS トンネリングは，フィルタリングされにくいという DNS のこの特性を利用する．DNS トンネリングでは，クライアントからサーバ方向のデータ転送に Qname を用い，その逆方向のデータ転送にはリソースレコードを使用する．このように双方向にデータを転送できる DNS トンネリングは，ターゲット組織から取得したデータを外部に流出させる際の手段としてだけでなく，ターゲットネットワークに潜伏しているマルウェアに対する C2 通信の手段として，サイバー攻撃で広く利用されている [2–9]．

しかし DNS プロトコルは，その名前解決の仕組みは変更されずに現在まで使用され続けている．この DNS トンネリングに対して，同一ドメインへの時間あたりのトラフィック頻度や問い合わせられるドメイン名のサブドメインにおける文

[*1] SIEM(Security Information and Event Management)：複数のソフトウェアや機器からトラフィックなどのログ情報を一元的に管理し，異常や脅威が発生した時に管理者に通知する仕組み
[*2] 秘匿通信：情報を不正・秘密裏に転送するために使用される回避通信手法

字列の分布や長さといった特徴に基づく検知アプローチはこれまでに多数提案されている [10–15]．検知に基づくアプローチを取る先行研究では，DNS トンネリングの擬似通信として，Github[*3]などから入手可能なトンネリング実装プログラムが用いた評価が行われる．しかし，擬似通信の発生に広く使われる Iodine [16] や DNSCat2 [17] といった実装は，インタラクティブシェル機能を目的としているため，時間あたりのトラフィック量が多く，パケットサイズも大きいという性質がある．このような顕著な性質が現れるトンネリング実装に対して，パケットごとのインターバルを 1ヶ月間などトラフィック頻度を調整したり，正規の FQDN の平均の長さまで注入するデータ量を下げるなどのバイパス手法がある [13]．これらバイパス手法を利用した場合，トンネリング実装の特徴に基づいた既存の検知手法を用いて検知することは困難である．

一方，これまでに多数の次世代名前解決システムは提案されてきているが，DNS トンネリング抑止を目的としたシステムは筆者が知りうる限り提案されていない．そこで本研究では，秘匿通信手法である DNS トンネリングの発生を抑止する次世代の名前解決システムを提案する．

1.2 目的

本研究の目的は，従来の名前解決システムを保ちながら，秘匿通信手法の DNS トンネリングを抑止する名前解決システムを開発することである．既存の DNS による名前解決システムは，現在のインターネットの根幹技術であるため，移行を念頭に設計されている必要がある．例えば，期待されないシステムとしては，以下のようなものが予想される．

- DNS トンネリング抑止は実現されるが，既存の名前解決システムが実現されない
- DNS トンネリング抑止は実現されるが，既存システムからの大幅な変更が必要になり，未対応のコンピュータのインターネット接続に支障をきたす

[*3]Github : ソースコード共有プラットフォーム

以上のことを踏まえた既存システムとの互換性を確保しながら，目的の実現を目指す.

1.3 貢献

提案システムは，既存システムのクライアントサーバアーキテクチャに基づき，既存の再帰問い合わせの仕組みのみに改変処理を施すことで，DNS トンネリングを抑止する名前解決システムの実現させる．提案システムは，既存クライアントに変更を加えずに秘匿通信としての機能を抑止することができるため，セキュアな名前解決システムとして広く一般に利用されることが期待される．

1.4 本論構成

本稿の構成は次の通りである．第 2 章では，本論で対象とする脅威モデルである DNS トンネリングを説明し，これまでの検知に基づくアプローチの限界を示す．第 3 章では，提案手法を説明する．第 4 章では，提案手法の DNS トンネリングに対する機能評価と提案システムの特性評価の結果について説明する．第 5 章では，提案手法の課題について議論する．最後に，第 6 章にて結論を述べる．

2. 脅威モデル

本章では，はじめに DNS の概要について述べる．その次に，本研究における脅威モデルである DNS トンネリングについて説明する．

2.1 DNS の概要

DNS は，ドメイン名に関連づけられたリソースレコードを解決するシステムである [18, 19]．DNS がユーザから問い合わせられたドメイン名の IP アドレスを解決することで，ユーザは識別しづらい IP アドレス (IPv4:“93.184.216.34”, IPv6:“2606:2800:220:1:248:1893:25c8:1946”) を直接入力することなく，サーバにアクセスすることができる．このような利便性を実現する DNS による名前解決の機能は，ユーザがインターネットを利活用する上で極めて重要である．

2.1.1 名前空間

DNS における名前空間は，委譲の仕組みに基づいて，ルートのゾーンを頂点に上位のドメインが下位のドメインにゾーンを分けていくことによって階層的に分割された構成になっている．ドメイン名では，右から左方向に階層の序列が表現されている．例として，“example.com”を考える．ルートドメインは一般に省略され，TLD(Top Level Domain) と呼ばれるルートの 1 つ下の階層に位置づくドメインは，この場合 “com”である．階層間の区切りには，ドット (“.”) が使用される．TLD の 1 つ下に位置づく SLD(Second Level Domain) は，この場合 “example”である．それぞれの名前空間はゾーンと呼ばれ，上位のドメインがその下位ドメインにゾーンを委譲することによって分割される．委譲の仕組みによってドメインごとにゾーンが分割されるのは，図 1 の色で区別する通りである．このゾーンは権威サーバによって管理され，権威サーバはゾーンファイルにドメイン名とレコードデータを記述することでドメイン名にレコードデータを関連づけることができる．委譲に基づいたゾーン分割の仕組みは，初期の DNS における Root.hints を用いた中央集権的な管理に伴うデータの同期の課題に対処したもの

である．Root.hints による単一ファイルによる管理では，増加するドメイン名に対応することが困難であった．委譲の仕組みによって，管理するドメイン名の対象をゾーンで分割されるため，負荷分散が実現されている．

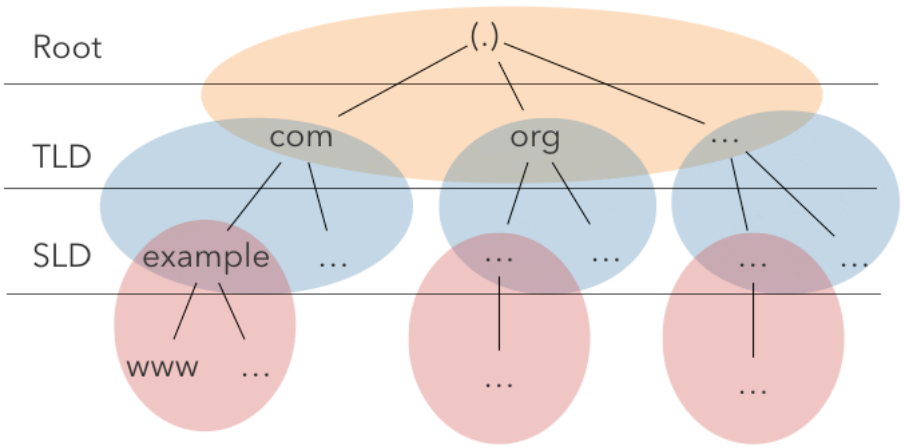


図 1 ゾーンごとに名前空間を色で区分した様子．Root 権威が管理するオレンジで表現されたゾーンには，com や org をはじめとしたドメインが含まれる．Root の直下に位置する TLD の一つである com ドメインは，青色で囲まれた example ドメインを含むゾーンを管理する．com に紐づけられる example ドメインは，www などを管理する．

アーキテクチャ

DNS は，クライアント・サーバアーキテクチャで構成され，機能に基づき 3 つのサービスに分類することができる．

- スタブリゾルバ
- フルサービスリゾルバ
- 権威サーバ

スタブリゾルバは，名前解決の問い合わせを行うクライアントノードである．フルサービスリゾルバ (キャッシュサーバやリカーシブサーバとも呼称される) は，スタブリゾルバに代わって，リソースレコードを保持する権威サーバに問い合わせるクライアントノードである．名前解決をする際には，ルートから順に TLD，

SLD という具合に権威サーバに再帰的に問い合わせることで，最終的に目的のドメイン名に関するリソースレコード情報を取得する．この時，はじめのルート権威サーバのアドレスは “Root.hints” と呼ばれるファイルに基づいて問い合わせるが，それより下位のドメインについては，上位の権威サーバが次の権威サーバのアドレスを応答することで名前解決のチェーンを繋げている．すなわち，ルート権威サーバが TLD の権威サーバのアドレスを応答し，TLD の権威サーバが SLD の権威サーバのアドレスを応答していく具合である．権威サーバは，リソースレコードを保持するサーバノードであり，フルサービスリゾルバからの問い合わせ依頼に応答する．

2.1.2 ドメイン名とリソースレコード

ルートからホストまでの階層構造の繋がりは，“label3.label2.label1.” のように右から左方向に連結することで表現される．この表記は，FQDN(Fully Qualified Domain Name) と呼ばれ，右端のドットがルート，“label1” が TLD，“label2” が SLD を表現し，ドメインごとの階層にはドットが使用される．ドメインの階層構造において，全てはルートを頂点としているため，ルートを意味するドットを略記した “label3.label2.label1” が，一般にはドメイン名として解釈される．次にドメイン名に関する長さおよび使用できる文字列について説明する．

$$(LabelD).(LabelC).(LabelB).(LabelA). \quad (1)$$

$$(Length) + (LabelD) + ... + (length) + (LabelB) + (length) + (LabelA) + 0 \quad (2)$$

$$1 + (Max63) + ... + 1 + (Max63) + 1 + (Max63) + 1 = (Max255) \quad (3)$$

(1) は，複数のラベルで構成されたドメイン名の例である．(2) は，Question ヘッダーに注入される際のそのドメイン名を表すデータである．Question セクションでは，ドメイン名を表す際にドットは省略され，ラベルの長さとラベル名，そしてドメイン名の終わりを意味する “0” で表現される．(3) は，ラベルの長さとラベル名のサイズを表す．ラベルの長さは，1 バイトのサイズで表現され，ラベル自体の最大長は 63 バイトである．Question セクションの最大長 255 バイトは，ラベルの長さとラベル，そしてドメイン終了を表す “0” を含めた長さである．この

ため，最初のラベル長を表す 1 バイトとドメイン名の終了を意味する “0” を表すための 1 バイトを差し引いた 253 バイトが，実際のドメイン長の最大長である．

ラベルには，数字とアルファベットおよびハイフン (“-”) を使用することができ，ラベル中に大文字・小文字の区別はない．他方で，アルファベットなどの ASCII 以外にも，国際化ドメイン名 (IDN: Internationalized Domain Name) を使用すると日本語やアラビア語なども使用することができる．IDN は，Punnycode[*4]などのエンコーディング手法に基づき，DNS クエリする際には ASCII コードに変換される [20]．

表 1　主要なリソースレコードの一覧

タイプ	目的
A	ホストの IPv4 アドレス
NS	権威サーバ
CNAME	別名
SOA	権威ゾーンの開始
NULL	NULL(実験用)
PTR	ドメイン名のポインター (逆引き)
MX	メール交換
TXT	任意文字列
AAAA	ホストの IPv6 アドレス
SRV	ドメイン名に対するサービスの場所
RRSIG	リソースレコード署名
NSEC	DNSSEC のための不在署名
DNSKEY	DNSSEC のための公開鍵
NSEC3	NSEC のバージョン 3
CAA	証明書を発行する認証局の指定

[*4]Punnycode: Unicode 文字列を一意かつ可逆的に ASCII 文字列に変換する符号化方式

ドメイン名に関連づけられる情報はリソースレコードと呼ばれ，目的に応じて複数のタイプが定義されている．例えば，ドメイン名にIPv4アドレスを関連づけることを考える．そのドメインの権威サーバは，関連づけたいIPv4アドレスをレコードタイプAとして，ゾーンファイルに記述する．クライアントは，ドメイン名とレコードタイプとしてAを指定しサーバに問い合わせることで，権威サーバが事前に登録したAレコードに記述されたアドレスを取得することができる．DNSでは，IPv4アドレス以外にも様々な情報をドメイン名に関連づけることができる．代表的なソースレコードのタイプを表1で示す．

表 2 DNSのパケットフォーマット

Header
Question
Answer
Authority
Additional

次に，DNSのパケットフォーマットについて説明する．表2で示すように，DNSのパケットは5つのセクションに分けられる．クライアントがDNSの名前解決をする際，解決したいレコードタイプとそのドメイン名は表3におけるNAMEフィールドに格納される．各フィールドはサイズが決まっており，Qnameフィールドが最大長255バイト，リソースレコードのタイプを表すQtypeフィールドとクエリクラスを表すQclassフィールドがそれぞれ2バイトとなっている．RDLENGTHは，応答データが含まれるRDATAの長さが格納される．

表 3 DNSのAnswerセクション (bits)

NAME (可変長)	TYPE (16)	CLASS (16)	TTL (32)	RDLENGTH (16)	RDATA (可変長)

2.1.3 名前解決の仕組み

例えば，クライアントが “www.example.com” の IPv4 アドレスを解決する場合を考える．はじめに，クライアントとなるスタブリゾルバは，スタブリゾルバと同一セグメント内のフルサービスリゾルバもしくは，ネットワークセグメントに依らないどこからでもアクセスできるフルサービスリゾルバ (オープンリゾルバ，パブリックリゾルバとも呼称される) に問い合わせる．フルサービスリゾルバは，その名前解決クエリが過去に解決したものでないかキャッシュデータを確認する．キャッシュにヒットした場合にはキャッシュの情報をクライアントに応答され，ヒットしなかった場合には，root.hints ファイルを参照しルート権威サーバにリクエストパケットを転送する．クエリ (問い合わせ) を受け取ったルート権威サーバは，“com” ドメインを委譲した権威サーバのアドレスを応答する．次に，フルサービスリゾルバは，“com” の権威サーバに対し同じクエリを転送する．“com” の権威サーバは，“example.com” ドメインを委譲した権威サーバのアドレスを応答する．フルリゾルバは，“example.com” の権威サーバに同じクエリを転送する．“example.com” の権威サーバは，保持するゾーンファイルからクエリされたドメインのリソースレコードについて探索し，探索の結果としてレコード情報をフルサービスリゾルバに応答する．フルサービスリゾルバは，権威サーバから応答された情報をスタブリゾルバに転送することで，問い合わせられた名前は解決される．DNS による名前解決の一連の動作を図 2 で示す．

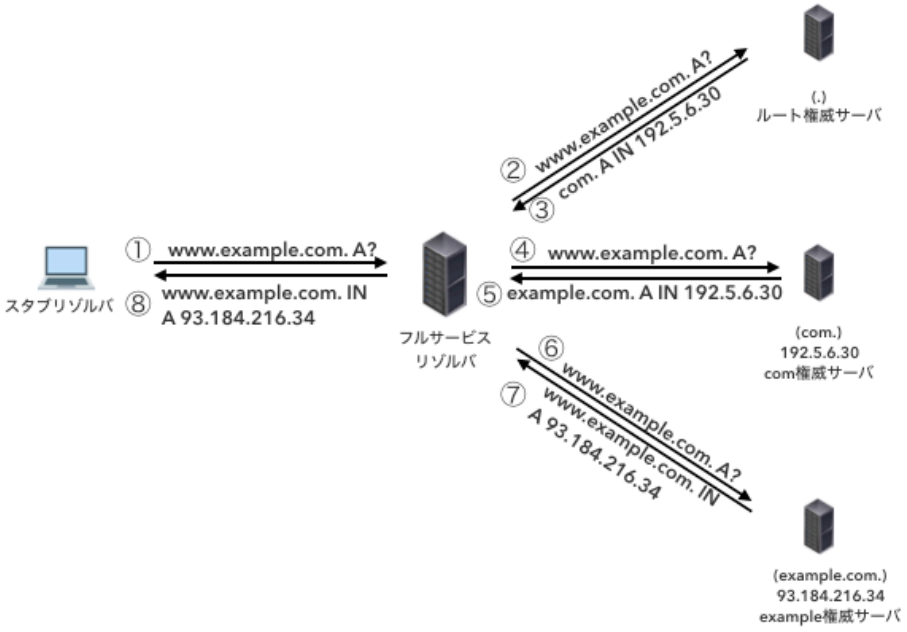


図 2 DNS による名前解決

2.2 DNS トンネリング

現在，情報流出を目的とするマルウェアのほとんどは，秘匿通信手法を利用しているとされている [13]．DNS トンネリングは，そのような秘匿通信の代表的な手法である．DNS トンネリングという名称は，DNS をデータ転送のメディアとした秘匿通信手法の総称であり，転送元と転送先の方向に基づき 2 つに分類することができる．一方がスタブリゾルバから権威サーバへのデータ転送手法である DNS Exfiltration，他方が権威サーバからスタブリゾルバへのデータ転送手法が DNS Infitration である．以下に示す DNS の特性を利用することによって，DNS トンネリングは機能する．

- パケットフォーマットの構造において，任意の文字列を注入できるフィールドを保持する
- 名前解決の通信はほとんどのサービスに先立って発生するため，
  - DNS のサービスポートが開かれ，フィルタリングされることが少ない
  - トラフィックが肥大化しやすく，長期でログを管理すること少ない

DNS トンネリングがデータ転送のキャリアとするフィールドは，クエリの Question セクションの Qname と，Answer セクションの Rdata である．Question セクションの Qname フィールドを利用することで，スタブリゾルバから権威サーバ方向にデータを転送できる．この方向の通信は，ビーコン通信やターゲットから取得した情報を外部に漏えいさせるといった攻撃の最終目的を達成するのに使われる．また，Answer セクションの Rdata フィールドを利用することで，データを転送することができる．この通信は，ターゲットネットワーク内のホストに潜伏したマルウェアなどへの命令コードを送信するのに使われる．さらに，この二つのキャリアを利用することが双方向の通信路を確保できるため，C2 通信を実施することも可能である．

DNS トンネリング手法が初めて一般に公開されたのは，1998 年に，ポートスキャンで知られる Nmap のメーリングリストだとされている [21, 22]．2004 年には，Dan Kaminsky が OzymanDNS [23] と呼ばれる DNS トンネリングの実装を公開した [24] ことをきっかけに広く知られるようになった．それ以降，数多くの DNS トンネリングの実装 [16, 17, 25–37] が公開され，実際のサイバー攻撃に悪用されるようになっている．

2.2.1 DNS Exfiltration

本項では，スタブリゾルバから権威サーバ方向にデータを転送する手法である DNS Exfiltration の詳細について説明する．DNS Exfiltration は，名前解決として問い合わせられるドメイン名が，そのドメインのゾーンを管理する権威サーバに転送される仕組みを利用した手法である．DNS では，ドメイン名に関連づけられるリソースレコードの情報は，そのドメインをゾーンとする権威サーバが保持しており，ルートから再帰的に問い合わせていくことでその権威サーバからの応答を受け取る．このため，問い合わせられたドメイン名が実在しない場合でも，再帰問い合わせの仕組みに従って，そのドメイン名の最後の権威サーバまで転送されることになる．権威サーバでは通常，クエリされたドメイン名の実在性の有無に依らず，問い合わせられたクエリ情報をログとして管理する．このような特性に踏まえて DNS を利用すると，DNS クエリのドメイン名のラベルに組織外ネッ

トワークに転送したい文字列を注入することで，組織外ネットワーク上に設置された権威サーバにそのデータを転送することができる．これが DNS Exfiltration の動作原理である．

このような仕組みの DNS Exfiltration を動作させるには，宛先となる権威サーバを用意する必要があり，グローバルなドメインを取得することを前提としている．第 2.1.3 項で述べるように，ドメイン名の最大長は 253 バイトであり，その内ラベルの最大長は 63 バイトという制約がある．そのため，DNS Exfiltration 手法を用いてデータを転送する際には，TLD のラベルと宛先権威サーバのラベルもしくは SLD ラベルと権威サーバのラベルを差し引いたサイズが実際に転送できる最大長となる．また，任意の文字列を DNS Exfiltration メソッドを用いて外部に転送するにあたり，転送キャリアであるドメイン名における文字列制約を満たすように転送したいデータに前処理を施す必要がある．ドメイン名に使用できる文字列は，第 2.1.3 項で述べるように，“a” から “z” までのアルファベットと “0” から “9” までの数字と先頭以外のハイフン “-” 記号である．この文字列制約については，転送したいデータをバイナリデータに変換し，そのバイナリデータをラベルとして印字可能な ASCII コードに変換することでその制約を満たすことができる．この前処理について，既存の DNS トンネリング実装の多くが Base Encoding [38] を用いている．この処理によって，転送データがバイナリデータである際にも転送効率上げたり，ラベルの文字列制約を満たさないデータも転送することができる．また，エンコーディングのラベルは，自然言語とは異なるため，メッセージの意味抽出を困難にすることにも機能する．

ここで，DNS Exfiltration を用いて，あるイントラネット内のホストからイントラネット外のホストにデータを転送することを考える．転送される宛先となるイントラネット外のホストには，“exfil.com” より下位の全ての名前空間をゾーンとする権威サーバ (“exfil.com”) を指定する．転送したい文字列にエンコーディング前処理を施した後，“用意した文字列.exfil.com” という具合に文字列をラベルとして含めることで，ドメイン名が用意できる．適当なリソースレコードタイプを指定し，DNS クエリとして転送すると，その権威サーバにはログとして，文字列を含んだドメイン名を取得する．最後に，受け取ったサーバサイドは，前処

理と逆のデコード処理を施すことで，オリジナルのデータを取得できる．以上のように再帰問い合わせとラベルという転送キャリア，エンコーディング処理を組み合わせることで，イントラネット内のホストから外部ネットワークに任意の情報を転送することができる．これが，DNS Exfiltration の動作メカニズムである．DNS Exfiltration のメカニズムは，図 3 で示す通りである．

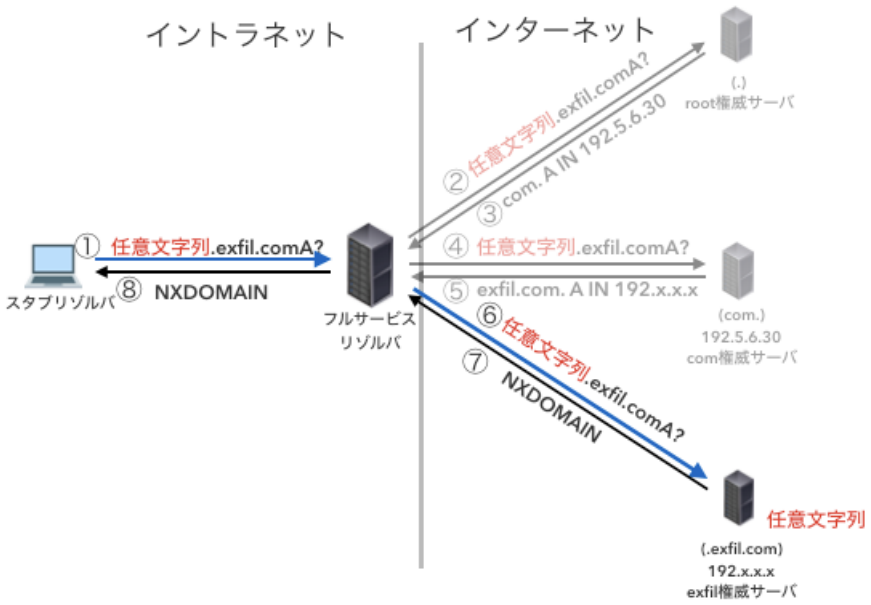


図 3　ドメイン名のラベル部に任意の文字列が注入された DNS クエリが，インターネット上の権威サーバ (“exfil.com”) に転送される様子．

2.2.2 DNS Infiltration

本項では，権威サーバからスタブリゾルバ方向にデータを転送する手法である DNS Infiltration の詳細について説明する．DNS Infiltration は，DNS における幾つかのリソースレコードが任意の文字列を記述できる設計を利用したデータ転送手法である．ドメイン名に関連づけられた情報を管理・提供する権威サーバは，ゾーンファイルに関連づけたい情報を記述する．リソースレコードには，レコード情報を検証する機構が備わっていないため，任意の文字列を登録することができる．特に，記法が決まっていない TXT タイプや NULL タイプなどもあり，DNS

Infiltration ではこのようなレコードタイプに転送したいデータを登録しておく．このようにして登録されたレコード情報について，名前解決問い合わせすることによって，インターネット (権威サーバ) からイントラネット (スタブリゾルバ) にデータを転送することができる．DNS Infiltration として利用され得るレコードタイプについて，これまでのトンネリング実装で使用されたものに基づいてまとめたのが，表 4 である．

表 4 DNS Infiltration として使用することができるレコードタイプの一覧

タイプ	最大サイズ (bytes)	説明	実装
A	4	ホストの IPv4 アドレス	
NS	4	権威サーバ	[17]
CNAME	253	別名	[16], [17], [31], [37]
NULL	255	NULL(実験用)	[16]
PTR	4	ドメイン名のポインター (逆引き)	
MX	253	メール交換	[16], [17]
TXT	255	任意文字列	[16], [17], [28], [29], [31], [32], [34]
AAAA	32	ホストの IPv6 アドレス	
SRV	180	ドメイン名に対するサービスの場所	[16]
DNSKEY	40	DNSSEC のための公開鍵	[36]

NS・CNAME・MX レコードでは，DNS Exfiltration と同じ要領でドメイン名のラベルに転送したい文字列を注入できる．また，NULL・TXT[*5]・SRV・DNSKEY を用いる場合には，レコード構文に指定がないため任意の文字列をそのまま注入できる．最後に，A・AAAA・PTR レコードを用いる場合には，転送したい文字列を数字に変換させた後に，ドット (.) 区切りもしくはコロン (:) 区切りで注入できる．

$$www.exfil.com. \quad IN \quad TXT \quad uname \ -ap \tag{4}$$

TXT レコードタイプを用いて DNS Infiltration することを考える．(4) は，TXT レコードを用いてホストで実行させる Linux コマンドを転送する例である．転送される uname プログラムは，システム情報を取得するプログラムである．実行結果を DNS Exfiltration 手法を用いて転送することで，ホストマシンにおけるシステムのカーネルバージョンやプロセッサの種類などの情報を取得することができる．次に，スタブリゾルバは，“www.exfil.com” の “TXT” レコードタイプを通常通り問い合わせる．再帰問い合わせの仕組みに基づいて，その DNS クエリは “exfil.com” まで転送され，ゾーンファイルの TXT レコードタイプの値がフルサービスリゾルバを経由したのち，スタブリゾルバまで応答される．DNS Infiltration の流入通信を図解した様子が，図 4 である．このようにして，DNS Infiltration では，正規の名前解決の方法を用いて，インターネットから組織内へとデータを取得することができる．

[*5] EDNS0 を使用する場合，65535bytes が最大長となる．

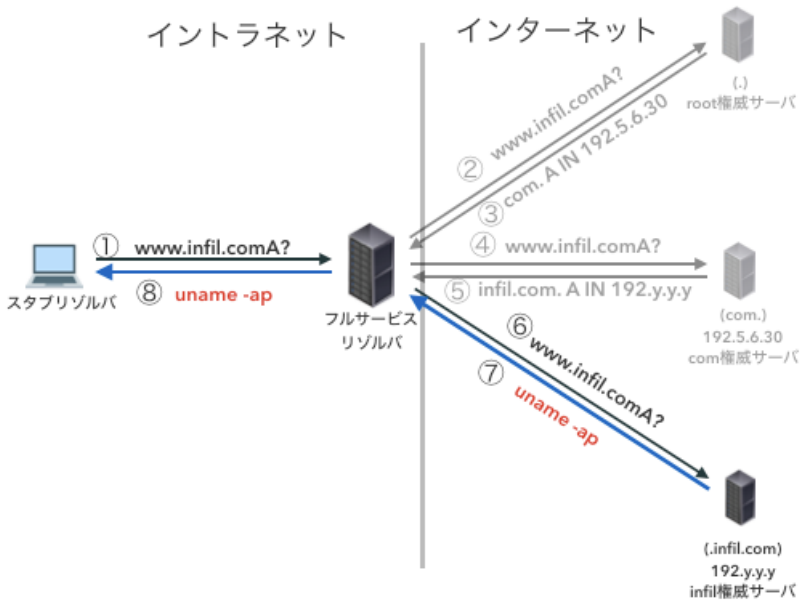


図 4 TXT レコードに登録された情報について，DNS クエリで問い合わせることで権威サーバから命令情報を取得している様子

2.3 DNS トンネリングへの既存対策

本節では，DNS トンネリングに対するこれまでの対策手法を紹介し，検知手法として用いられるアプローチについて説明する．最後に，それら検知アプローチを迂回する脅威モデルを示し，検知に基づくアプローチに限界があることを明らかにする．

DNS トンネリングに対して，森下らは提案されている DNS トンネリングへの対策アプローチを以下のようにまとめている [39]．

1. DNS クエリログの取得と保存・内容の調査
2. エンタープライズネットワークにおける OP53B[*6] の適用
3. DNS ファイヤーウォールの導入

[*6] OP53B：組織外に設置されたオープンリゾルバのようなフルサービスリゾルバや権威サーバとの通信を抑止するために，組織外ネットワークを宛先とする 53 番ポートの通信をブロックする仕組み．

クエリログの取得は解析・調査のために必要不可欠である一方で，DNS のログファイルは肥大化しやすく管理コストが課題になる．OP53B の適用は，組織内部からの名前解決トラフィックをイントラネットに設置されたフルサービスリゾルバに集約することができるため，オープンリゾルバならびに権威サーバと直接通信することに伴う悪性通信の発生抑止に寄与する．DNS ファイヤーウォールとは，ベンダーが独自に開発したトンネリングなどの悪性通信の統計データから閾値や機械学習モデルに基づき，悪性トラフィックを検知・ブロックする製品を指す．

2.3.1 特徴量

従来，DNS トンネリング通信の検知には，以下に示す特徴量について統計分析を用いた閾値の算出や機械学習を用いた悪性モデルが用いられてきた．トンネリング実装などによって発生する一般的な DNS トンネリング通信の検知にあたり，以下のような特徴を利用した手法がこれまでに多数提案されている．

ドメイン名の長さとクエリパケットのサイズ

クライアントからサーバ方向にデータを転送させる DNS Exfiltration 手法において，転送キャリアとなるドメイン名が注入されるデータ量に応じて長くなる [40]．例えば，DNS Exfiltration において，一回あたりのデータ転送量を増加させる場合，Qname フィールドのドメイン長もそれに比例して長くなり，結果としてパケットサイズも増加するという具合である．DNS Infiltration においても同様で，一回あたりのデータ転送量を増加させる場合，Rdata フィールド内のデータ量も大きくなり，応答パケットのサイズが増加する．

表 5 正規 DNS クエリと DNS トンネリングにおけるドメイン名の違い

種類	ドメイン名
正規	www.example.com
トンネリング	arbitrary-text.you-can-input-here-as-labels.example.com

同一ドメインあたりのトラフィック頻度

DNS トンネリングでは，一度に転送できるデータ量に限界があるため，目的のデータを全て転送するには分割する必要がある．トンネリング実装のように対話的にシェルコマンドを実行する通信の場合，トラフィック頻度は極めて高頻度になる．また，サイズの大きいデータを DNS Exfiltration を用いて転送する場合も同様に，複数のパケットに分割されたデータを転送するにあたって，多数のトラフィックが発生することになる．

リソースレコードのタイプ

理論的に全てのリソースレコードを用いてデータを転送することは可能であるが，第 2.2.2 項で示すように，使用するレコードタイプによって転送できるデータ量は大きく異なる．表 4 で示すように，実際のトンネリング実装における DNS Infiltaration を目的とする通信では，A や AAAA などは使われず，CNAME や TXT が主に使用される．A や AAAA などのレコードタイプが使用されない背景には，数字のみの文字列制約が厳しさと最大のデータサイズに小さいことが考えられる．TXT の最大サイズが 253bytes であるのに対して，A が 4bytes で AAAA が 32bytes なのは明らかに小さいことが確認できる．他方で，通常のインターネットの利活用において使用されるレコードタイプに極端な分布の偏りがあることが知られている．Herrymann ら [41] は， 2010 年 1 月 1 日から 6 月 30 日までの期間において，大学構内に設置されたフルサービスリゾルバによる DNS ログデータを収集した．収集の結果，ドメイン名に対する IPv4 と IPv6 のアドレス解決の通信が全体の大部分の 89.407%を占め，Infiltration として使用される CNAME や TXT は 1%に満たないことが明らかになった (表 17)．以上のことから，TXT や CNAME といった任意の文字列を注入できる反面，使用頻度が低いレコードタイプである特性から，データ転送効率と秘匿性がトレードオフであることが確認できる．

パケットの応答ステータス

DNS のヘッダーは，表 6 で示すようなフィールドを持っており，問い合わせに対して，表 7 のようなステータス情報を応答する．第 2.3.2 項で示すような検知迂回手法を使う場合を除いて，通常の DNS Exfiltration では，権威サーバが未知のデータがクラアントから転送される．そのため，クライアントからの問い合わせには，コンテンツ不在を意味する “NXDomain” が応答される．応答パケットのステータスが “NXDomain” であるとき，DNS Exfiltration の可能性がある．

表 6 DNS の Header セクション

Identification(16bits)									
QR (1)	Opcode (4)	AA (1)	TC (1)	RD (1)	RA (1)	Z (1)	AD (1)	CD (1)	Rcode (4)

表 7 代表的な Rcode 一覧

値	名前	意味
0	NoError	正常
1	FormErr	フォーマットエラー
2	ServFail	サーバエラー
3	NXDomain	存在しないドメイン
4	NotImp	未実装
5	Refused	問い合わせ拒否

ドメイン名に含まれる文字列の出現頻度

Born ら [10] は，ドメイン名に使用されている文字列の分布について，流布しているトンネリング実装と正規の DNS 通信について調査した．その結果，正規のドメイン名が英語における文字列の出現分布と相関があるのに対して，トンネリング実装によって生成されるドメイン名における文字列の出現頻度では相関がみられず，文字列の出現頻度はランダムとなる傾向にある．

2.3.2 既存検知手法に対する脅威モデル

本項では，既存検知アプローチを迂回する脅威モデルを示しながら，既存対策手法の課題を説明する．第 2.3.1 項で示すように，DNS トンネリング手法に基づいてデータを転送するとき，正規の DNS クエリパケットと比べて，そのパケットは不自然に長くランダムな Qname になる性質や頻繁なトラフィックが発生するという性質が現れる傾向にある．そのような性質を特徴量として分析することによって，これまでに多数の検知手法が提案されてきた [10–15]．その際，実際の攻撃シーンに使用されている DNS トンネリング通信を入手することが困難であるため，多くの研究では公開されているトンネリング実装を擬似 DNS トンネリング通信と仮定することによって，それぞれの提案手法の評価を行なっている．しかし，実際の攻撃シーンに使用される DNS トンネリング通信とそれらトンネリング実装によって発生する通信は，目的の違いに起因して通信の性質が異なるため，既存の研究はトンネリング実装依存もしくはその実装に特化した検知手法であると言える．流布しているトンネリング実装は，メッセージやファイル転送の通信を DNS に代替させることによって目的のデータを通信させることだけが目的である [16] のに対し，APT[*7]や悪意目的実行者は，悪意通信の解析を回避することが制約条件として上乗せされていることが想定される．このため，データ転送の通信を DNS に代替させるだけでなく，それら通信が IDS・IPS や Firewall といったセキュリティシステムに検知されないように調整していることが考えられる．

DNS トンネリング通信における検知システムの迂回には，以下に示す 2 つのアプローチが考えられる．

- 暗号化による解析自体の無効化
- スループット低下よる正規通信への埋没化

暗号化による解析自体の無効化

現在 DNS では，クライアントとサーバ間の通信におけるプライバシーが盛んに議論されており，スタブリゾルバとフルサービスリゾルバ間の通信を TLS も

[*7]APT(Advanced Persisted Threat): 高度な専門知識と膨大なリソースに基づき，抵抗する防御側の技術に適応させる高度に持続的な脅威 [42].

しくは HTTPS を用いて暗号化する仕組みが標準化されるに至っている [43, 44]．通信の暗号化に関わらず DNS トンネリングは，権威サーバを用意しそのドメインを宛先とすることによって変わらず動作する．事実，DoH(DNS over HTTPS) を用いたトンネリングは実装が公開され，その機能性が確認されている [32–34]．このような DNS トンネリングを暗号化手法によって，スタブリゾルバからのクエリパケットに基づいて分析することは難しく，流出した後の権威サーバとの通信パケットのみで分析する以外に手段がなくなる．また，組織において OP53B が適用されていなかった場合，攻撃者はオープンリゾルバを介した暗号化 DNS トンネリングすることが想定され，この場合検知使用できる特徴量がペイロードが暗号化されるためトラフィックのみとなるため，DNS トンネリングを検知することは極めて困難になる．

スループット低下による正規通信への埋没化

はじめに述べたように，DNS トンネリングと正規通信との違いは，Qname の特徴やトラフィック頻度となって現れる傾向にある．しかし，データサイズは一回あたりの転送の削減によって，トラフィック頻度はパケット間のインターバルを長期化することによって，容易に調整することができる．このようにして，トンネリング通信を正規通信に埋没させる場合，トンネリング通信を検知することは難しく，検知には誤検知の課題が浮上する．

以上のように，検知に基づくアプローチには限界があることを踏まえ，DNS トンネリングの本質的な解決には，DNS の名前解決の仕組みを改善することが必要である．

3. 関連研究

本章では，次世代 DNS として提案されている名前解決システムを紹介し，DNS トンネリング抑止への課題について明らかにする．

3.1 P2P ネットワーク

本節では，P2P ネットワークに基づいた DNS について説明する．クライアントとサーバを排除し，ノード間をピアで接続する P2P ネットワークには耐障害性とスケーラビリティの特性がある．この優位性を名前解決に応用した手法は，過去に複数提案されている．Cox ら [45] は，既存システムの管理における非効率なロードバランシングと低い障害耐性の課題を解決することを目的に P2P ネットワークに基づいたシステムを提案している．提案されたシステム DDNS(Distributed domain name system) は，既存システムの再帰問い合わせの仕組みに代わる，Chord アルゴリズムを使った DHash に基づいてサーバ探索することで目的を実現させる [46]．システムにおける名前解決の仕組みは，図 5 で示すように，ハッシュ関数によって算出された ID に基づいてノードとコンテンツをリング状に射影し，コンテンツに近いノードがコンテンツ情報を管理・応答することによって機能する．クラ

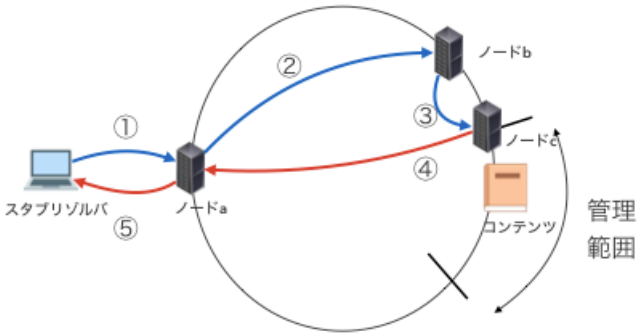


図 5 DDNS における名前解決プロセス

イアントからの問い合わせを受け取ったノード a は，クエリされたコンテンツ ID を導出し，経路情報に明記された ID のリストの内で最も近しいノード b にクエ

リを転送する．コンテンツの ID に最も近いノード c まで再帰的に問い合わせられると，最終的にコンテンツを保持するノード c がコンテンツ情報をノード a に応答することで名前が解決される．Chord リング上では，コンテンツ ID と経路情報に基づいて宛先ノードが決まるため，特定ノードにデータを意図的に転送することは難しく，DNS トンネリングの発生を抑止することできる．しかし，Chord アルゴリズムではコンテンツを保持するノードの探索が非効率であることが指摘されている．線型探索の課題を改善するために，“finger table”に基づいていくつかのノードをスキップ手法が提案されているが，この場合の探索でも $O(\log N)$ のクエリを必要とするため，パフォーマンスの課題がある [47]．

Yiting ら [48] は，P2P ネットワークにおける遅延の課題を解消する HDNS(Hybrid DNS) を提案している．提案システムは，Public Zone と呼ばれる Chord アルゴリズムに基づいた P2P ネットワークと，Internal Zone と呼ばれる従来の階層型 DNS ツリー構造を組み合わせで構成されている．図 6 で示すように，クエリは DDNS 同様の要領でコンテンツに近いノードに転送した後，階層構造に基づきクエリを転送する．HDNS では，パフォーマンスこそ改善されるが，コンテンツを保持するノードがドメイン名の階層構造に従うため，DNS トンネリンを抑止することはできない．

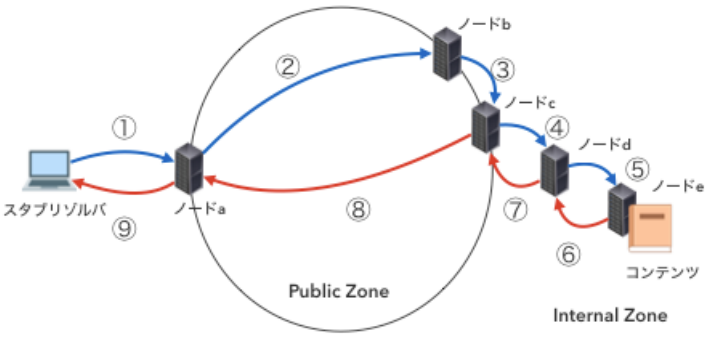


図 6　HDNS における名前解決プロセス

GNU [49] は，DNS におけるプライバシーの課題を解消することを目的に，ピュアな P2P で接続する GNS を提案している．GNS では，全てのユーザが自身の名前空間を保持・管理し (例: .bob)，また他のユーザにサブドメインを委譲できる (例:

.sample.bob) 設計になっている．図 7 は，Alice リゾルバが “www.bob.dave.gun” と “www.carol.dave.gnu” という名前を解決するために辿るパスを示している．このように，ユーザ間はピュアな P2P で接続され，更に通信には暗号化を前提としているという特徴がある．GNS では，擬似的な TLD として “.gnu” が使われるなど，従来の名前空間とは全く異なる．GNS では，ユーザ同士が暗号化した通信を用いて直接やりとりため，既存システムにおける DNS トンネリングの仕組みでデータを転送することを許す設計になっている．

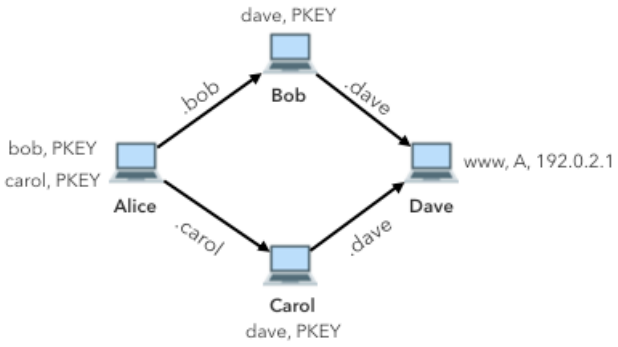


図 7 GNS における名前解決プロセス

3.2 課題

上記で説明してきたように，同一のハッシュ関数から導出される ID を持つコンテンツとノードを P2P ネットワークに構築する手法だけが，DNS トンネリングを抑止することができる．しかし，この手法に基づいた名前解決では，既存システムと比べて大きな遅延が発生するという課題がある．その他に提案されてきたシステムでは，DNS トンネリングの手法は依然として利用できてしまう．次章では，DNS トンネリング抑止機能だけでなく，高速な名前解決の両方を備える提案システムについて説明する．

4. 提案システム

本章では，DNS トンネリングの発生抑止を目的に設計した名前解決システム DNS-TD(DNS for Tunneling Deterrence) を説明する．

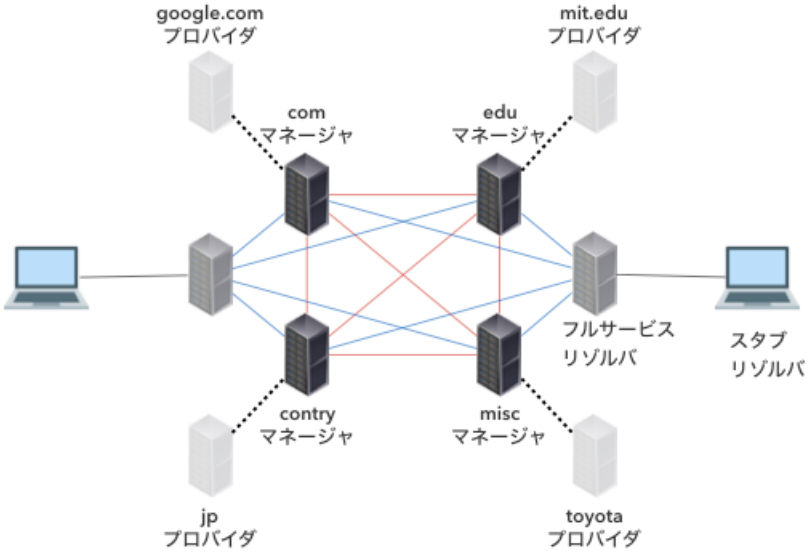


図 8　提案システムの概略図

4.1 概要

現在の DNS の名前解決の仕組みにおいて，名前空間が委譲の仕組みに基づきドメインごとにゾーンで分割されているため，名前解決クエリは目的ドメインの権威サーバまで転送される必要がある．また，リソースレコードはドメイン名に関係ない任意の情報を関連づけることができる設計になっている．この 2 つの特性に起因して，DNS トンネリングは機能する．すなわち，再帰問い合わせに基づく名前解決の仕組みとドメイン名に関連づけるレコード情報に高い自由度を排除することによって，DNS トンネリングの発生抑止を実現することができる．一方で，名前解決システムとしての機能を維持するために，以下に示す 2 つの性質を満たす必要がある．

名前解決

ドメイン名に IP アドレスなどの情報を関連づけることができ，それを解決することができる

スケーラビリティ

ドメイン名の増加および関連づけられるレコード情報の増加に対応することができる

以上から，期待される名前解決システムの要件は，上記 2 つの性質を満たしながら先の特性を排除することである．そこで，提案システム DNS-TD では，識別子の割り当て不足が無視できる程度に大規模の名前空間と範囲に基づくゾーン分割によって名前解決とスケーラビリティを実現し，再帰問い合わせの特性を排除させ，認証システムによってリソースレコードの自由度を下げることで上記の要件を満たす．以降では，その要件を満たすための手法および仕組みを概観する．

不足が無視できる程度に大規模の名前空間

DNS-TD では，ドメイン名とレコード情報の組に対して，識別子を付与することによって名前解決を行う．その際に使用される識別子には，ドメイン名とレコードタイプをメッセージとする写像関数を用いて算出されたダイジェストを利用する．識別子が固有のものでない場合，クエリされた識別子に複数の値が応答され望ましい名前解決に支障を来す．このため，識別子には 84bytes という不足が無視できる程度に大規模の名前空間を持つハッシュ関数を使用している．識別子は，ドメイン名とそのレコードタイプの文字列和をメッセージとするハッシュ関数から算出されるダイジェストである．例えば，ドメイン名が “www.example.com” でレコードタイプが “A” のペアを考える．この場合，メッセージが “www.example.comA”，識別子がこのメッセージをハッシュ関数に与えたダイジェスト “例:86ff...485” となる．

範囲に基づくゾーン分割

識別子の名前空間について，ソートされた空間の特定範囲に基づいてゾーンが分割される．提案システムにおけるサーバは，このようにして分割されたゾーンを

それぞれ担当することによって，分散的な管理システムとして協調することで名前解決機能を実現する．提案システムにおけるサーバ機能は，一部を除いた[*8]gTLDによって集約される．すなわち，SLD 以降の権威サーバにサーバ機能はなく，ドメイン名とレコードタイプの作成と更新の機能のみを担う．既存の SLD 以降の権威サーバは，gTLD サーバにドメイン名の階層構造の序列を維持した状態で連結し，コンテンツ情報の操作を通じて gTLD にサーバ機能を委任する．このように既存システムの権威サーバの機能を，サーバ機能とコンテンツ作成などの操作機能に分類することによって，クライアントからの権威サーバへの透過性を防ぎ，DNS Exfiltration を抑止する．

認証システム

DNS-TD では，認証の仕組みを導入することでレコード情報の真正性を確保する設計をとっている．既存システムでは，ドメイン名に関連づける情報はゾーンファイルにて定義されるが，ゾーンファイルを編集する主体が権威サーバであるため任意の情報を含めることができる設計になっている．提案システムでは，先に述べるようにコンテンツの管理機能と編集機能とを分離させる．コンテンツの編集機能は，サーバに階層的に接続されるノード，プロバイダが担う．プロバイダを起点として行われるコンテンツへの操作は，認証機関を介在した後でサーバで実行される設計になっている．この認証プロセスでは，依頼者 (プロバイダ) 情報およびレコード情報とその関連先となるドメイン名について真正性について検証される．例えば，アドレスなどの情報であれば接続性が検証され，その他の情報であればレコード情報をドメイン名に関連づける正当性などが検証される．この認証プロセスをパスし，証明書が発行されたコンテンツのみが，サーバによって管理される．この認証プロセスによって，不審な情報がドメイン名に関連づけられることを未然に対処する．

以降では，上記 3 つのアプローチについて詳細に説明する．また，DNS-TD で使う用語を表 8 でまとめて示す．

[*8]ccTLD とブランド TLD は SLD として扱い，それぞれ “cc” と “brand” という TLD に接続される．

表 8 DNS-TD における用語

表記	意味または機能
コンテンツ	・識別子に関連づけられたレコード情報の実体
コンテンツ ID	・識別子
ドメイン ID	・識別子 (コンテンツ ID が重複した際に使用)
レコード情報	・リソースレコードの具体的な値 (例 IP アドレス)
リソースレコードタイプ	・オブジェクトに関連づけるリソースレコードの型 (例 A, AAAA, MX)
オブジェクト	・問い合わせる対象 (ドメイン名もしくは IP アドレス)
スタブリゾルバ	・名前解決クライアント
フルサービスリゾルバ	・スタブリゾルバからのクエリハンドリング ・識別子の作成
マネージャ	・フルサービスリゾルバからのクエリハンドリング ・ゾーンの管理 ・コンテンツの保持
プロバイダ	・コンテンツの作成・更新・削除操作

4.2 システムアーキテクチャ

本節では，DNS-TD のシステムアーキテクチャについて説明する．現在，DNS はインターネットの根幹に位置づく技術であり，ほぼ全てのクライアントノードは既存システムが提供するアーキテクチャおよびプロトコルに依存している背景がある．このため，システムのアーキテクチャの再構成において，エッジノードに対して変更が加えられるのは，導入負荷が高くなることが予想される．現在の DNS による名前解決は，フルサービスリゾルバを介在させながら，スタブリゾルバをクライアント，権威サーバをサーバとするクライアントサーバアーキテクチャで構成されている．DNS-TD では，導入フェーズで予想されるクライアントに対する名前解決処理システムの負荷を軽減することを目的として，従来同様のクライアントサーバアーキテクチャを踏襲する．サーバ群は，図 9 で示すように，相互で接続されたフルメッシュなネットワークで構築される．

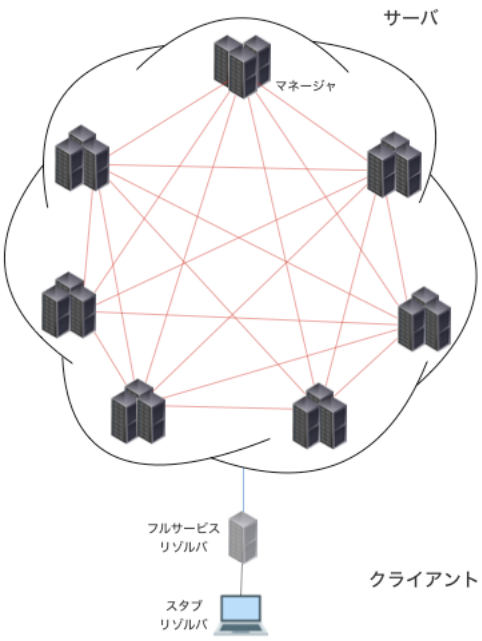


図 9 DNS-TD におけるクライアントサーバアーキテクチャ

4.3 サービスノード

本節では，DNS-TD における各サービスノードの機能と他のサービスとの関わりを説明する．DNS-TD の名前解決ネットワークにおいて，クライアントがスタブリゾルバ，サーバの機能はマネージャが担当する．

スタブリゾルバ

スタブリゾルバは，既存システムと変わらない．これは第 4.2 節で述べるように，名前解決の仕組みの変化に伴ってクライアントに接続障害が発生する可能性がある．既存の DNS に依存したクライアントの存在を踏まえて，接続性に影響を与えないためにスタブリゾルバは現行の方法で目的のリソース情報を解決できる設計になっている．すなわち，スタブリゾルバは，IP アドレスをはじめとしたオブジェクトに関連づけられたレコード情報を問い合わせ，目的サービスを提供するサーバのリソースにアクセスするクライアントノードである．既存システム同様，スタブリゾルバのクエリはフルサービスリゾルバに転送され，キャッシュにヒットした場合には即座にレコード情報の応答結果を取得する．ヒットしなかった場合には，フルサービスリゾルバがスタブリゾルバに変わって，サーバにクエリを転送し，応答結果をスタブリゾルバに返す．

マネージャ

マネージャは，2 つの機能を担うサービスノードである．それは，クライアントからの問い合わせに応答する機能と他のマネージャに操作リクエストを転送する機能である．マネージャは，既存システムにおける権威サーバから分離した機能の一部であり，その残りの機能はプロバイダが担当している．はじめに，マネージャとドメインおよびプロバイダの関係について説明する．

DNS-TD では，既存のドメインの階層構造は引き継がれ，マネージャとプロバイダそれぞれが独自のドメインを持っている．プロバイダは，マネージャと親子関係にあるノードであり，マネージャが上位ドメイン，プロバイダが下位ドメインという構成である．マネージャは，既存システムにおける TLD に相当するドメインを保持する．TLD には国や地域に割り当てられる ccTLD と分野別の gTLD の

2 つに大別することができる．DNS-TD では，コンテンツはその ID に基づき管理する主体が決定する．このため，ccTLD がマネージャである場合，ナショナリズムや政治などに起因した操作判断が，名前解決システムの全体の運用に支障を来す事態が発生する可能性がある．このことを回避するために，DNS-TD の設計ではマネージャが保有できる TLD を gTLD に限定している．ccTLD は，“country” をドメインに持つマネージャにサーバ機能を委譲し，プロバイダとしてレコード情報の操作を行うことで現在の TLD のドメインレベルを保つ．これは，図 10 で示すように，ドメインが “jp.country” となるのではなく，サーバ機能を “country” をドメインに持つマネージャに委ねるということである．他方で現在，gTLD にはコミュニティ以外に “google” をはじめとした企業 TLD がある．先の国や地域に基づくシナリオであったように，民間企業の判断でインターネット全体に影響が波及するような接続性の断絶は起きうる．そこで，企業やブランドを表す TLD は，“brand” というドメインにもつマネージャにサーバ機能を委譲し，プロバイダとして存在を継続させる．その他のクラスとして分類することが困難な “foo” といった TLD については，“misc” というドメインを持つマネージャにサーバ機能を委譲させる．このように，DNS-TD では，ドメインの名前空間を継続しながら，サーバとしての機能を再定義する．

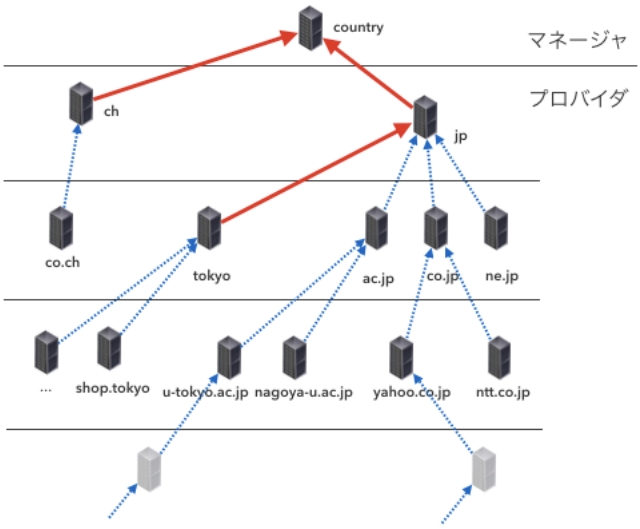


図 10　マネージャとプロバイダの関係

以上のことを踏まえて，マネージャの機能について説明する．1つ目は，ドメイン名とそれに関連づけられたレコード情報を保持し，フルサービスリゾルバからの問い合わせに応答する機能である．マネージャは，レコード情報を管理するためにデータベースを用いる．マネージャが保持するコンテンツは，そのドメイン名とレコードタイプによって決まる．必ずしも自身のドメインを含むコンテンツを保持するわけではない．例えば，“www.example.com”の A レコードについて考える．この組のコンテンツ ID が “47d8...cb6”であるとする．他方で，“com” マネージャのゾーンは，“a000...000”から “bzzz...zzz”を担当しているとする．また，“org”マネージャが “4000...000”から “5zzz...zzz”を担当しているとする．この時，“www.example.com”は com という TLD をもつが “com”マネージャではなく，“org”マネージャが保持する．このようにして，コンテンツの管理は，ドメインに基づいて管理されるのではなくコンテンツ ID の値とハッシュ値の範囲に基づいて決まる．マネージャによるクエリの処理は，アルゴリズム 1 である．

表 9　マネージャが使用する関数と保持する情報

表記	意味
parser()	クエリパケットをデータ構造に分解する関数
db_accesser()	データベースにクエリする関数
benigh_responce()	正常応答用のペイロードを作成する関数
error_responce()	不在応答用のペイロードを作成する関数
pack()	パケットの DNS のデータ構造にパックする関数
sendto()	クライアントに結果を応答する関数
record_value	レコード情報

アルゴリズム **1.** マネージャにおける名前解決問い合わせ処理

```
handler(query_data):
    content_id, qtype ← parser(query_data)
    record_value ← db_accesser(content_id)
    if value :
        payload ← benigh_response(content_id, qtype, ttl, record_value)
    else:
        payload ← error_response(content_id, qtype)
    payload ← payload.pack()
    sendto(payload, client_address)
```

2つ目は，プロバイダからコンテンツに対するの操作リクエストを受け付け，コンテンツ ID を算出し担当のマネージャに操作リクエストを転送する機能である．フルサービスリゾルバから問い合わせが発生した際，はじめにアルゴリズム **??** で示すようにクエリパケットからコンテンツ ID を取得する．次に，レコード情報を取得するために，コンテンツ ID をキーとしてデータベースから対応するコンテンツを探索する．コンテンツの存在の有無に従い，存在した場合にはレコード情報が応答され，そうでなかった場合には不在として応答される．

アルゴリズム 2. マネージャにおけるコンテンツ操作問い合わせ処理

プロバイダからのコンテンツ操作リクエストハンドリング

handler(*request_data*)**:**

```
data, provider_addr ← parser(request_data)
content_id, domain_id ← calculate_id(data.object, data.rtype)
manager_addr ← find_manager(start,end,content_id)
sendto(data,manager_addr)
```

コンテンツ ID とドメイン ID の算出

calculate_id(*qname, rtype*)**:**

```
content_id ← hash.sha3_224(qname + rtype)
domain_id ← hash.sha3_224(qname)[: 28]
return content_id, domain_id
```

コンテンツ ID が含まれるゾーンを保持するマネージャアドレスの解決

find_manager(*start, end, content_id*)**:**

```
for i, j in map_start, map_end :
    if i ≤ content_id ≤ j :
        p ← map_start.index(i)
        manager_addr ← map.addr[p]
        return manager_addr
```

最後に，マネージャにおけるコンテンツの管理について説明する．コンテンツの実態は，以下に示す 5 つの要素が含まれた情報の集合である．

- ドメイン名
- レコードタイプ
- TTL(Time To Live)
- レコード情報
- 証明書

保持するコンテンツ情報は，上記で示すように，長さに変化のある固定数の文字列で表現される要素が単純に列挙された構造である．このデータには，クエリ情報に基づいてデータリソースにアクセスできることが DNS-TD におけるデータモデルの要件になる．名前解決におけるクエリ情報は，ドメイン名とそれに関連づけるデータのタイプ情報であることから，この 2 つの情報に基づいて生成される識別子をキーとして，コンテンツをバリューとする KVS モデルが最も単純であると考えられる [50]．以上のことを踏まえ，DNS-TD では，図 12 に示すように，クエリ情報から算出される識別子をキー，コンテンツ情報をカンマ区切りで表現した文字列で表現したデータをバリューとするモデルを採用する．

図 11　コンテンツのデータフォーマット

フルサービスリゾルバ

フルサービスリゾルバは，サーバからの応答をキャッシュするを持つサービスノードである．また，コンテンツ ID およびドメイン ID を算出し，コンテンツを保持するマネージャに問い合わせる機能を担う．全てのフルサービスリゾルバは，マネージャとそのマネージャのゾーンに関する対応表のファイルを保持している．この対応表は，ICANN から提供される “Root.hints” ファイルのようにウェブ上で公開され，入手することができる．フルサービスリゾルバは，アルゴリズム 3

で示すように，スタブリゾルバからのクエリに含まれるドメイン名とレコードタイプに基づきコンテンツ ID とドメイン ID を導き出す．コンテンツを保持するマネージャは，コンテンツ ID が含まれるゾーンを探索することで一意に決定される．名前解決には，“コンテンツ ID. ドメイン ID”のようにドット区切りで ID を組み合わせたものを識別子としてマネージャに問い合わせる．レコード情報もしくは不在情報に関する応答パケットをマネージャから受け取ると，フルサービスリゾルバは既存システム同様に応答情報をキャッシュした後，スタブリゾルバに応答する．

表 10　フルサービスリゾルバが使用する関数と保持する情報

表記	意味
query_manager()	マネージャに問い合わせる関数
response_client()	結果をクライアントに応答する関数
hash.sha3_224()	54bytes の SHA3 ハッシュ関数
start	ゾーンにおける範囲の開始アドレス
end	ゾーンにおける範囲の終了アドレス
client_address	クライアントの IP アドレスとポートのタプル
answer.rcode	マネージャにおける応答コード
answer.rdata	レコード情報
map_start	ゾーンにおける範囲の開始アドレスのリスト
map_end	ゾーンにおける範囲の終了アドレスのリスト

プロバイダ

プロバイダは，既存システムの権威サーバの機能のうち，レコード情報を操作する機能を担当するノードである．すなわち，既存システムの SLD 以降のドメイン情報に関して，作成・更新および消去といったレコード情報の操作を担当する．メインの階層構造に上位のドメインを保持するマネージャが，プロバイダが保持するドメインのサーバ機能を担当する．プロバイダは，認証局にてコンテンツ情報の真正性を評価されたのちに，プロバイダの上位に位置づくマネージャがそのコンテンツを担当するマネージャに依頼することでレコード情報を操作する．プ

アルゴリズム 3. フルサービスリゾルバにおける問い合わせ転送処理

handler(*query_data, rtype*):

```
content_id, domain_id ← calculate_id(query_data, rtype)
manager_addr ← find_manager(start, end, content_id)
answer ← query_manager(manager_addr, content_id, domain_id)
response_client(client_address, qname, answer.rcode, answer.rdata)
```

ロバイダが認証局に転送する情報には以下の 4 つである．

- ドメイン名
- レコードタイプ
- TTL(Time To Live)
- レコード情報

認証局

認証局は，レコード情報の真正性を検証する信頼された第 3 者機関である．DNS を用いた既存の名前解決システムでは，ドメイン名に任意の情報を関連づけることができることに起因して，DNS トンネリングとして利用される課題があった．この課題に対して DNS-TD では，ドメイン名に関連づけるレコード情報について，第 3 者機関からの認証を介在させることによって，不審な情報がドメイン名に関連づけられることを抑止する．認証局を用いた認証プロセスでは，プロバイダからのドメイン名へのレコード情報を関連づけるリクエストをきっかけとする．認証局に転送されるリクエストパケットに関して，認証局は内容と依頼元の情報に基づいたデータの真正性を検証する．この検証フェーズで認証されたコンテンツは，リクエストしたプロバイダの上位に位置づくマネージャに証明書を付与して転送される．認証されなかった場合には，リクエストは破棄され，その破棄された結果がリクエストしたプロバイダに応答される．

例えば，ドメイン名が “www.example.com” で，このドメイン名に “uname -ax” という文字列を TXT レコードに関連づけることを考える．プロバイダは，認証

局を宛先としてコンテンツの真正性に関する検証評価を依頼する．依頼には，ドメインにレコード情報を関連づける目的情報を併せて要求する．認証局は，関連づけたい内容と目的を評価する．この場合，“uname -ax”は Linux コマンドであり不審なデータとして評価され，リクエストは破棄される．関連づける情報が，IP アドレスであった場合には，接続性が評価された後，証明書を付与したコンテンツをマネージャに転送する．これが認証における一連のプロセスであり，これによって不審なデータがドメインに関連づけられることを抑止する．

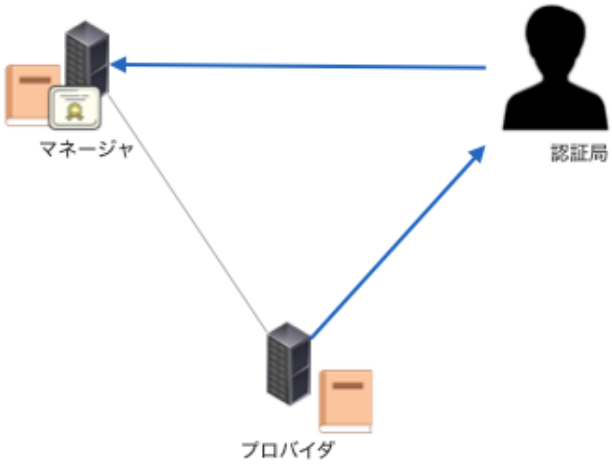


図 12　レコード情報操作におけるプロセスの概略図

4.4 識別子

本節では，コンテンツに付与する識別子について説明する．DNS の名前解決システムでは，委譲の仕組みに基づいてドメインごとにゾーンを保持する設計になっているため，名前解決にあたりクエリ情報はそのドメイン名をゾーンとする権威サーバまで転送される．この仕組みでは，ドメインを作成することで，そのドメインをゾーンにもつ権威サーバに対して任意の情報を DNS クエリに含めることでデータを転送することができる，DNS トンネリングとして機能する潜在的な特性がある．転送する際に，正規のクエリ時に使用される長さや文字列に調整することで，正規クエリとトンネリングクエリを判別することは曖昧でき，このようにしてセキュリティシステムを迂回される脅威に発展する．この課題は，名前空間を保持する機能とその空間上のドメイン情報を提供する機能が共在することに起因する．そこで，DNS-TD では，名前空間の保持機能と提供する機能を分離することで課題を解決する．これは，ドメインの名前空間は維持したまま，コンテンツに識別子を付与し，この識別子の名前空間をフラットにすることで実現される．DNS-TD では，ドメイン名とレコードタイプという組みを単位として，全ての組みに画一的な名前空間上の値を識別子として付与する．このように，DNS-TD では，全てのドメイン名とレコードタイプの組みに識別子を付与するため，その識別子の名前空間は数の不足が無視できる程度に大きくなくてはならない．また，第 4.3 節で述べるように，既存の名前解決の仕組みに依存したものは多く，プロトコルのフォーマットに変化を加えないことが望ましい．DNS のクエリパケットにおいてデータを含められる Question セクションの Qname の最大長は，253bytes である．また，Qname はドメイン名を想定した設計になっているため，ドメイン名の制約にある最大 63bytes とするラベル長の制約を満たす必要がある．上記の制約を満たしながら，ドメイン名とレコードタイプから生成される識別子には，54bytes の名前空間を持つハッシュ関数によって生成されるメッセージダイジェストを用いる．また，ダイジェストの衝突には，ダイジェストを増やし名前空間を拡張させることによって対処する．以降では，識別子に用いられるハッシュアルゴリズムとシステムの分散処理を目的としたゾーン分割法について説明する．

4.4.1 ハッシュアルゴリズム

本項では，コンテンツを識別するために使われる識別子の算出に利用するハッシュアルゴリズムについて説明する．トンネリング抑止を目的とする提案システムにおけるハッシュアルゴリズムの要件は，以下の通りである．

- 不足を無視できる程度に大きい名前空間を持つ
- DNS プロトコルフォーマットに準拠する

不足を無視できる程度に大きい名前空間を持つ

全てのドメイン名とレコードタイプ情報を組みに付与される識別子は，数の不足が無視できる程度に大きい名前空間を必要とする．IPv6 は全てのホストのインターフェースに一意に識別子を付与するのに，不足を無視できる名前空間として 128bits(32bytes) を採用している．提案システムにおいて，候補とするハッシュアルゴリズムを表 11 で示す．これらアルゴリズムの名前空間は，IPv6 で使用されていた 32bytes と同等かそれ以上の名前空間であり，どのアルゴリズムを使用しても数の不足は無視できると考えられる．

表 11 ハッシュアルゴリズムの一覧

アルゴリズム	名前空間 (bytes)
MD5	32
SHA1	40
SHA2	56, 64, 96, 128
SHA3	56, 64, 96, 128

DNS プロトコルフォーマットに準拠する

DNS-TD における識別子を用いた名前解決では，既存の DNS パケットのフォーマットを利用する．識別子を格納する領域は，Question セクションの Qname である．Qname では，ドメイン名における命名規則に準拠するため，最大のラベル長が 63bytes，最大のドメイン長が 253bytes となる．1 つのラベルにメッセージダイジェストを含める場合，そのラベル長の制約を満たすのは MD5・SHA1・

SHA2(56bytes)・SHA3(56bytes) の 4 つである．上記で挙げる 4 つのアルゴリズムを利用することでトンネリング抑止のための提案システムの要件は満たされる．他方で，フルサービスリゾルバとマネージャ間におけるトラフィックは，プライバシーの観点から第 3 者から覗かれるべきではない．第 3 者からのパケットの中身を抑止するためには，パケット全体を TLS 暗号などに基づいて暗号化する方法が考えられる．ハッシュ関数として原像計算困難性 (弱衝突耐性) の性質を持つアルゴリズムを選択することで，クエリの機密性の効果を副次的に期待される．このことを踏まえて，現在標準化されているハッシュ関数の中から，最新の SHA3 を提案システムのハッシュ関数として採用する．以上から，DNS-TD では，56bytes(224bits) の名前空間をもつ SHA3 をハッシュアルゴリズムを用いる．

有限空間に写像するハッシュ関数には，写像したダイジェストが他のコンテンツによって算出されたダイジェストと衝突する可能性がある．以降では，ダイジェストのコリジョンを回避するために，名前空間を拡張させるドメイン ID について説明する．DNS-TD におけるコンテンツ ID は，ドメイン名とレコードタイプの文字列和をメッセージとするハッシュ関数のダイジェストである．例えば，ドメイン名が “www.example.com” で A のレコードタイプの組み合わせを考える．アルゴリズム 4 に従い，コンテンツ ID のメッセージになるのは “www.example.comA” である．そして，このメッセージをハッシュ関数にかけて算出された値 “47d87...4cb6” がコンテンツ ID となる．ドメイン ID は，ドメイン名をメッセージとするハッシュ関数から算出されるダイジェストの前半 28bytes である．すなわち，メッセージが “www.example.com” で，“86ff20...bf026” がドメイン ID となる．以上から，最終的なマネージャに問い合わせられる識別子は，それぞれの ID をドット区切りで連結された “(コンテンツ ID).(ドメイン ID)” となる．

4.4.2 ゾーン分割

本項では，ゾーンの分割方法とそのゾーンとマネージャの対応表について説明する．DNS-TD におけるゾーンは，ソートされたコンテンツ ID の名前空間の連続した範囲に従って分割される．表 12 で示すように，“com” ドメインを管理するマネージャは “000...00” から “2xx...xx” の連続した範囲の名前空間を管理する．

アルゴリズム 4. コンテンツ ID とドメイン ID の導出方法

calculate_id(*qname, rtype*):

```
content_id ← hash.sha3_224(qname + rtype)
domain_id ← hash.sha3_224(qname)[: 28]
identity ← content_id + "." + domain_id
return identity
```

コンテンツ ID がこの範囲下に含まれる場合には，“com”に問い合わせることによってレコード情報を管理することができる．対応表は，ICANN から提供される “Root.hints” ファイルのようにウェブ上で公開され，常時入手可能な状態が維持される．全てのフルサービスリゾルバと認証局は，この対応表に基づきコンテンツ ID を導出することで一意に管理マネージャを特定することができる．

表 12 マネージャ情報とそのマネージャが管理するゾーンに関する対応表

ゾーン	マネージャ アドレス	ドメイン
(000…00, 2zz…zz)	192.35.51.30	com
...	...	...
(500…00, 6zz…zz)	192.5.6.30	net
...	...	...
(b00…00, czz…zz)	199.249.112.1	org
...	...	...
(n00…00, mzz…zz)	199.254.31.1	info
...	...	...
(y00…00, zzz…zz)	194.0.0.53	arpa

5. 評価

本章では，提案システムのDNSトンネリング抑止効果について評価を行った結果を示す．提案システムがトンネリング抑止に有効であることを，実装したプロトタイプ上で擬似トンネリング通信を発生させるシミュレーションに基づいて明らかにする．また，提案システムのフルサービスリゾルバからサーバまで1ホップで問い合わせられる仕組みが，トラフィック量の削減と高速な名前解決の特性を備えていることを示す．

5.1 プロトタイプ実装とシミュレーション環境

提案システムにおけるトンネリング抑止機能の評価にあたり，システムのプロトタイプをPython3を用いて実装した．実装では，DNSパケットの作成およびパース(分解)のためにdnslib[*9]ライブラリを使用した．また，マネージャにおけるコンテンツ管理には，Redis[*10]データベースを使用した．フルサービスリゾルバとマネージャのサービスを実装し，Dockerに基づいた仮想環境上でコンテナをサービスノードとして動作させることによる擬似的な名前解決基盤を構築した．スタブリゾルバから，名前解決のクエリとしてdigコマンドを使用した．表13に，使用したソフトウェアとそのバージョンを示す．

表13 使用したライブラリと環境

ソフトウェア	バージョン
Python	3.7.5
Docker	19.03.5
dnslib	0.9.10
Redis	3.3.11
dig	9.10.6

[*9]dnslib: https://bitbucket.org/paulc/dnslib/src/default/
[*10]Redis: https://redis.io/

次に，Docker 環境上に構成したネットワーク構成について説明する．ネットワークは，一般的な組織内からインターネットに設置されている外部リソースにアクセスするために，インターネット上に設置されたサーバ (マネージャ) にアクセスするというシナリオに基づいて設計した．また，組織におけるインターネット利用は，インターネットに設置されたオープンリゾルバや直接サーバに接続することを防止するために OP53B が設定されているという想定である．そのため，図 13 で示すように，組織内部のスタブリゾルバは同内部のフルサービスリゾルバを経由して，インターネット上のマネージャに問い合わせられることで名前が解決される．

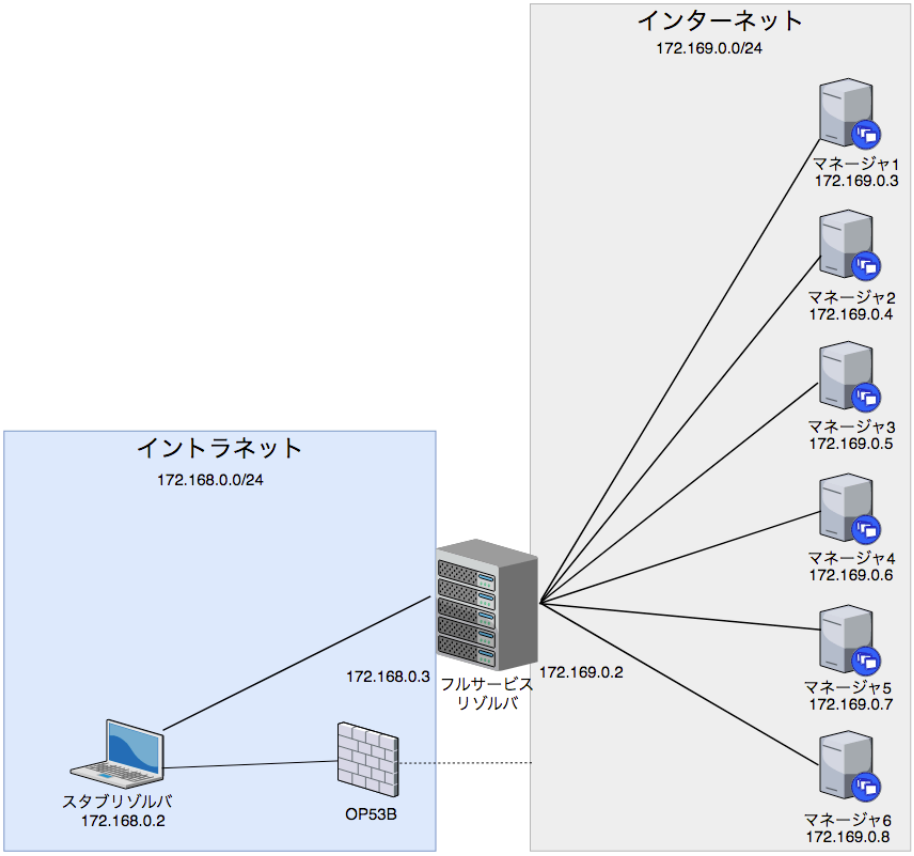
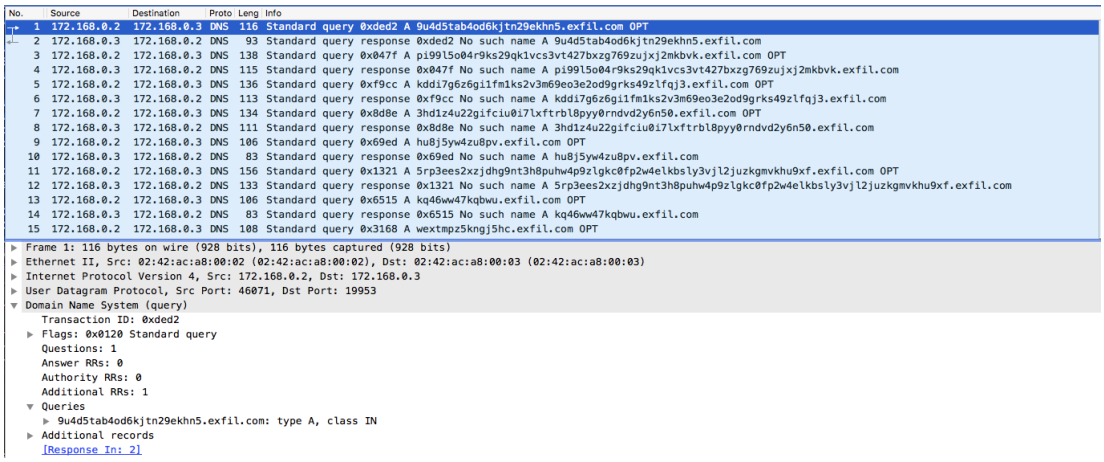


図 13　Docker 環境内におけるネットワークトポロジー

5.2 DNS トンネリング

本節では，提案システムの DNS Exilfiltration 抑止機能について，擬似 DNS トンネリングの通信を提案システム上で発生させるシミュレーションに基づいて評価した結果を示す．シナリオは，DNS Exiltration の手法に基づいて，スタブリゾルバから “exfil.com” を宛先に DNS クエリが発せられることを想定する．DNS トンネリングの問い合わせに用いる擬似ドメイン名には，1 文字から 63 文字の長さでランダムな文字列で生成されたラベルを 5000 個を用意し，ドメインが “exil.com” となるようにそのラベルをホスト名として組み合わせたドメイン名を作成した．DNS Exfiltration では，問い合わせるリソースレコードのタイプの種類は関係ないため，全て A レコードを設定した．実験では，先の 5000 個のドメイン名を問い合わせるスクリプトを用意し，スタブリゾルバからクエリさせた．図 14 で示すように，クエリは既存システム同様，はじめに組織内部のフルサービスリゾルバに転送される．フルサービスリゾルバは，問い合わせられたドメイン名とレコードタイプからドメイン ID とコンテンツ ID を算出し，コンテンツを保持するマネージャのアドレスをコンテンツ ID に基づいて決定する．図 15 で示すように，Question セクションの Qname には，識別子である “コンテンツ ID. ドメイン ID” が含まれている．

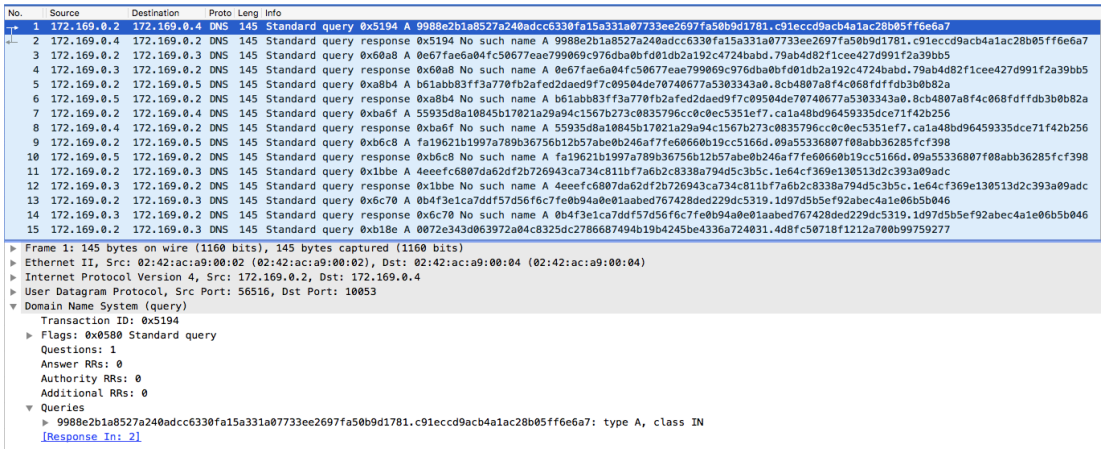
このような仕組みによって，スタブリゾルバからのクエリは，コンテンツを操作するプロバイダを介在せずに名前解決を行える．このメカニズムによって，任意のサーバをデータ転送先とする DNS Exfiltration の発生を抑止する．



```
No.  Source          Destination     Proto Leng Info
  1  172.168.0.2     172.168.0.3     DNS   116  Standard query 0xded2 A 9u4d5tab4od6kjtn29ekhn5.exfil.com OPT
  2  172.168.0.3     172.168.0.2     DNS    93  Standard query response 0xded2 No such name A 9u4d5tab4od6kjtn29ekhn5.exfil.com
  3  172.168.0.2     172.168.0.3     DNS   138  Standard query 0x047f A pi99l5o04r9ks29qk1vcs3vt427bxzg769zujxj2mkbvk.exfil.com OPT
  4  172.168.0.3     172.168.0.2     DNS   115  Standard query response 0x047f No such name A pi99l5o04r9ks29qk1vcs3vt427bxzg769zujxj2mkbvk.exfil.com
  5  172.168.0.2     172.168.0.3     DNS   136  Standard query 0xf9cc A kddi7g6z6gi1fm1ks2v3m69eo3e2od9grks49zlfqj3.exfil.com OPT
  6  172.168.0.3     172.168.0.2     DNS   113  Standard query response 0xf9cc No such name A kddi7g6z6gi1fm1ks2v3m69eo3e2od9grks49zlfqj3.exfil.com
  7  172.168.0.2     172.168.0.3     DNS   134  Standard query 0x8d8e A 3hd1z4u22gifciu0i7lxftrbl8pyy0rndvd2y6n50.exfil.com OPT
  8  172.168.0.3     172.168.0.2     DNS   111  Standard query response 0x8d8e No such name A 3hd1z4u22gifciu0i7lxftrbl8pyy0rndvd2y6n50.exfil.com
  9  172.168.0.2     172.168.0.3     DNS   106  Standard query 0x69ed A hu8j5yw4zu8pv.exfil.com OPT
 10  172.168.0.3     172.168.0.2     DNS    83  Standard query response 0x69ed No such name A hu8j5yw4zu8pv.exfil.com
 11  172.168.0.2     172.168.0.3     DNS   156  Standard query 0x1321 A 5rp3ees2xzjdhg9nt3h8puhw4p9zlgkc0fp2w4elkbsly3vjl2juzkgmvkhu9xf.exfil.com OPT
 12  172.168.0.3     172.168.0.2     DNS   133  Standard query response 0x1321 No such name A 5rp3ees2xzjdhg9nt3h8puhw4p9zlgkc0fp2w4elkbsly3vjl2juzkgmvkhu9xf.exfil.com
 13  172.168.0.2     172.168.0.3     DNS   106  Standard query 0x6515 A kq46ww47kqbwu.exfil.com OPT
 14  172.168.0.3     172.168.0.2     DNS    83  Standard query response 0x6515 No such name A kq46ww47kqbwu.exfil.com
 15  172.168.0.2     172.168.0.3     DNS   108  Standard query 0x3168 A wextmpz5kngj5hc.exfil.com OPT

▶ Frame 1: 116 bytes on wire (928 bits), 116 bytes captured (928 bits)
▶ Ethernet II, Src: 02:42:ac:a8:00:02 (02:42:ac:a8:00:02), Dst: 02:42:ac:a8:00:03 (02:42:ac:a8:00:03)
▶ Internet Protocol Version 4, Src: 172.168.0.2, Dst: 172.168.0.3
▶ User Datagram Protocol, Src Port: 46071, Dst Port: 19953
▼ Domain Name System (query)
    Transaction ID: 0xded2
  ▶ Flags: 0x0120 Standard query
    Questions: 1
    Answer RRs: 0
    Authority RRs: 0
    Additional RRs: 1
  ▼ Queries
    ▶ 9u4d5tab4od6kjtn29ekhn5.exfil.com: type A, class IN
  ▶ Additional records
    [Response In: 2]
```

図 14　スタブリゾルバからフルサービスリゾルバにおける通信



```
No.  Source          Destination     Proto Leng Info
  1  172.169.0.2     172.169.0.4     DNS   145  Standard query 0x5194 A 9988e2b1a8527a240adcc6330fa15a331a07733ee2697fa50b9d1781.c91eccd9acb4a1ac28b05ff6e6a7
  2  172.169.0.4     172.169.0.2     DNS   145  Standard query response 0x5194 No such name A 9988e2b1a8527a240adcc6330fa15a331a07733ee2697fa50b9d1781.c91eccd9acb4a1ac28b05ff6e6a7
  3  172.169.0.2     172.169.0.3     DNS   145  Standard query 0x60a8 A 0e67fae6a04fc50677eae799069c976dba0bfd01db2a192c4724babd.79ab4d82f1cee427d991f2a39bb5
  4  172.169.0.3     172.169.0.2     DNS   145  Standard query response 0x60a8 No such name A 0e67fae6a04fc50677eae799069c976dba0bfd01db2a192c4724babd.79ab4d82f1cee427d991f2a39bb5
  5  172.169.0.2     172.169.0.5     DNS   145  Standard query 0xa8b4 A b61abb83ff3a770fb2afed2daed9f7c09504de70740677a5303343a0.8cb4807a8f4c068fdffdb3b0b82a
  6  172.169.0.5     172.169.0.2     DNS   145  Standard query response 0xa8b4 No such name A b61abb83ff3a770fb2afed2daed9f7c09504de70740677a5303343a0.8cb4807a8f4c068fdffdb3b0b82a
  7  172.169.0.2     172.169.0.4     DNS   145  Standard query 0xba6f A 55935d8a10845b17021a29a94c1567b273c0835796cc0c0ec5351ef7.ca1a48bd96459335dce71f42b256
  8  172.169.0.4     172.169.0.2     DNS   145  Standard query response 0xba6f No such name A 55935d8a10845b17021a29a94c1567b273c0835796cc0c0ec5351ef7.ca1a48bd96459335dce71f42b256
  9  172.169.0.2     172.169.0.5     DNS   145  Standard query 0xb6c8 A fa19621b1997a789b36756b12b57abe0b246af7fe60660b19cc5166d.09a55336807f08abb36285fcf398
 10  172.169.0.5     172.169.0.2     DNS   145  Standard query response 0xb6c8 No such name A fa19621b1997a789b36756b12b57abe0b246af7fe60660b19cc5166d.09a55336807f08abb36285fcf398
 11  172.169.0.2     172.169.0.3     DNS   145  Standard query 0x1bbe A 4eeefc6807da62df2b726943ca734c811bf7a6b2c8338a794d5c3b5c.1e64cf369e130513d2c393a09adc
 12  172.169.0.3     172.169.0.2     DNS   145  Standard query response 0x1bbe No such name A 4eeefc6807da62df2b726943ca734c811bf7a6b2c8338a794d5c3b5c.1e64cf369e130513d2c393a09adc
 13  172.169.0.2     172.169.0.3     DNS   145  Standard query 0x6c70 A 0b4f3e1ca7ddf57d56f6c7fe0b94a0e01aabed767428ded229dc5319.1d97d5b5ef92abec4a1e06b5b046
 14  172.169.0.3     172.169.0.2     DNS   145  Standard query response 0x6c70 No such name A 0b4f3e1ca7ddf57d56f6c7fe0b94a0e01aabed767428ded229dc5319.1d97d5b5ef92abec4a1e06b5b046
 15  172.169.0.2     172.169.0.3     DNS   145  Standard query 0xb18e A 0072e343d063972a04c8325dc2786687494b19b4245be4336a724031.4d8fc50718f1212a700b99759277

▶ Frame 1: 145 bytes on wire (1160 bits), 145 bytes captured (1160 bits)
▶ Ethernet II, Src: 02:42:ac:a9:00:02 (02:42:ac:a9:00:02), Dst: 02:42:ac:a9:00:04 (02:42:ac:a9:00:04)
▶ Internet Protocol Version 4, Src: 172.169.0.2, Dst: 172.169.0.4
▶ User Datagram Protocol, Src Port: 56516, Dst Port: 10053
▼ Domain Name System (query)
    Transaction ID: 0x5194
  ▶ Flags: 0x0580 Standard query
    Questions: 1
    Answer RRs: 0
    Authority RRs: 0
    Additional RRs: 0
  ▼ Queries
    ▶ 9988e2b1a8527a240adcc6330fa15a331a07733ee2697fa50b9d1781.c91eccd9acb4a1ac28b05ff6e6a7: type A, class IN
    [Response In: 2]
```

図 15　フルサービスリゾルバからマネージャにおける通信

5.3 特性評価

本節では，既存システムと比べて提案システムがトラフィック量が少なくなる点と名前解決の高速が期待される点について説明する．提案システムは，既存システムと比較するとき，表 14 に示すような特性が現れる．

表 14 DNS と DNS-TD の特性比較

	DNS	**DNS-TD**
Qname サイズ (bytes)	変長 (最大 253)	固定長 (85) (コンテンツ ID(56).(1) ドメイン ID(28))
問い合わせ回数	委譲された回数	1
RTT	全ての権威サーバとの RTT 総和	マネージャとの RTT のみ
ゾーン管理	ファイル	データベース
オーバーヘッド	再帰問い合わせ	・コンテンツ ID の計算 ・ドメイン ID の計算 ・マネージャ探索処理

5.3.1 トラフィック量

本項では，名前解決に伴って発生するトラフィック量を比較評価した結果を示す．表 14 で示すように，提案システムではサーバへの問い合わせは一回で済む．既存システムでは，コンテンツを保持するサーバまで再帰的に問い合わせることを踏まえると，提案手法の方がトラフィック数は少なくなる．一方で，既存システムは任意のドメイン名が使用されるのに対して，提案システムでは常に固定長の 85bytes のドメイン名が使用される．

トラフィック量の評価においては，クエリパケットのみに焦点を当てた．既存システムにおける再帰問い合わせでは，ルートから TLD，SLD と権威サーバの

アドレスが Authoirity セクションに含まれて応答されるが，問い合わせられるドメイン名ごとで委譲されている数が異なる．また，権威サーバのアドレスとして含めることができるアドレスは一つでないため，応答パケットのサイズにはドメイン毎にランダムである特性がある．このように応答パケットのサイズはドメイン依存であるため推定することが困難である．以上から，トラフィック量の推定には，クエリパケットのみを焦点に当てた．また，既存システムと提案システムのスタブリゾルバからフルサービスリゾルバまでの通信は両者とも共通であるため，評価するトラフィックはフルサービスリゾルバとサーバ (権威サーバ，マネージャ) 間の通信を評価した．評価では，長さの異なる 253 種類のドメイン名をスタブリゾルバからクエリし，フルサービスリゾルバから権威サーバまでのクエリパケットのサイズを対象とした．既存システムでは，権威サーバへの問い合わせる方法には，2 つの種類がある．1 つ目は，通常の全ての権威サーバに同じ FQDN で問い合わせる方法である．この場合，権威サーバを宛先とするパケットは，常に同じパケットサイズとなる．2 つ目は，宛先となる権威サーバにはその次の権威サーバのドメイン名のみを問い合わせる Qname Minimisation [51] と呼ばれる手法である．Qname Minimisation は，権威サーバに問い合わせる Question セクションのドメイン名が最小限に留められる．例えば，“www.example.com” について考える．フルサービスリゾルバにおいて Qname Minimisation の設定が有効になっている場合，ルート権威サーバには “com” の NS レコード情報が問い合わせられる．同様にして，“com” 権威サーバには，“example.com” の NS レコード情報が問い合わせられるという具合である．

はじめに，ルートの A レコードタイプに関するクエリパケットのサイズを収集した．次に，全ての TLD のドメイン名を収集し，TLD の A レコードタイプに関するクエリパケットのサイズを収集した．また，全てのドメインの長さパターンにおけるパケットサイズのデータを収集した．Qname Minimisation を使用しない場合，名前解決に伴うクエリの総トラフィックサイズは，以下の計算式で求めることができる．

表 15 パケット構成する要素とそのサイズ

表記	意味	サイズ (bytes)
R	ラベルの長さを示す領域	1
r	Root を表す “.” をストアする領域	1
O	Qname 以外のクエリパケットサイズ	68
Rtype	レコードタイプ	1(A) 2(NS)

$$(n\ Label's\ Traffic) = (R + (Label\ Length) + r + Rtype + O) \times n \quad (5)$$

$$= (1 + (Label\ Length) + 1 + 1 + 68) \times n \quad (6)$$

$$= (71 + (Label\ Length)) \times n \quad (7)$$

図 16 は，DNS と DNS-TD における名前解決で発生するクエリパケットの総トラフィック量の比較結果である．x 軸がスタブリゾルバから問い合わせられたドメイン名の長さで，y 軸が総トラフィック量である．

図 16 から，Qname Minimisation を使用しない場合では，提案システムのような固定長の方が再帰的に問い合わせるよりもトラフィック量は抑えられることが確認できる．

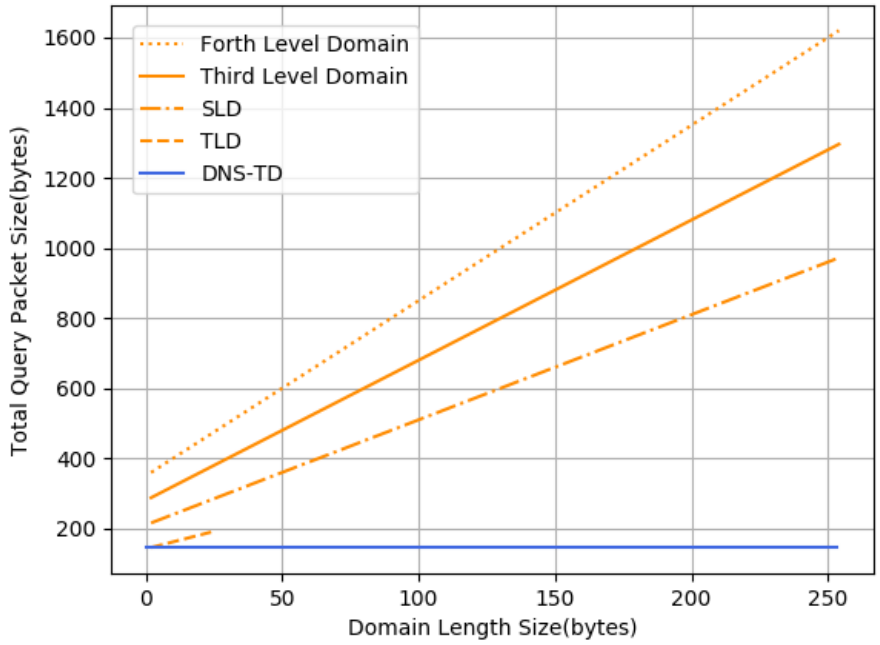


図 16　DNS-TD と DNS における名前解決に使用されるクエリパケットサイズの比較

5.3.2 オーバーヘッド

提案システムの名前解決メカニズムでは，名前解決問い合わせの都度，コンテンツ ID とドメイン ID を導出する必要がある．ハッシュ関数に基づいているこの 2 つの識別子を導出する処理は，名前解決処理におけるオーバーヘッドになることが予想される．本項では，識別子の導出処理に伴う時間的なオーバーヘッドについて，検証実験に基づいて評価した結果を示す．

評価では，はじめに “exfil.com” をドメインとするランダムに作成した 5000 個のホスト名とリソースレコードのタイプの組を用意した．その組からコンテンツ ID とドメイン ID を導出するのにかかった時間の計測した．この操作を 4 回繰り返し，組ごとのダイジェスト導出にかかった時間の平均をとったのが，表 17 である．処理時間を計測には，Python3 における精度評価に用いられる time ライブラリの perf_counter メソッドを用いた．検証環境は，表 16 の通りである．

表 16 識別子算出のパフォーマンステスト環境

要素	環境
OS	MacOS(10.14.6)
CPU	1.6GHz Intel Core i5
メモリ	8GB 1600GHz DDR3

図 17 で示す検証の結果から，識別子の組を導出するのにかかる時間は約 0.003 ミリ秒の分布する．名前解決にかかる時間は，ルート権威サーバを例にとると図 18 で示すように，0 から 1000 ミリ秒と振り幅はあるものの明らかに識別子にかかる時間は無視できる程度に小さいことが確認できる．以上から，提案システムにおける識別子導出にかかる時間的オーバーヘッドは無視できるものと捉えられる．

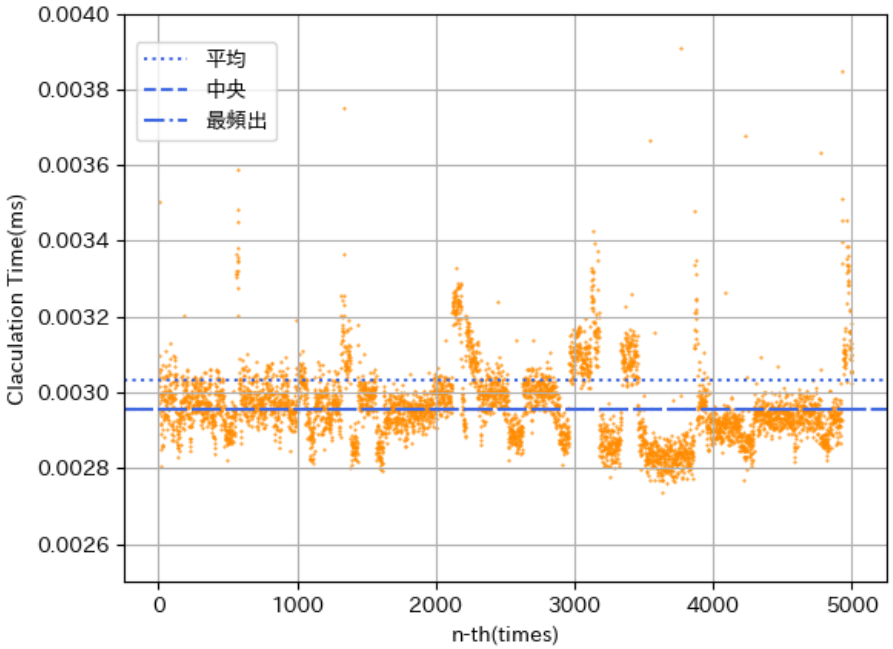


図 17　コンテンツ ID とドメイン ID 導出にかかる計算時間のオーバーヘッド

5.3.3 名前解決速度

提案システムでは，コンテンツを保持するサーバはコンテンツ ID から一意に定まる．一方，既存システムでは，ルートから階層的にコンテンツを保持するサーバを探索した後に定まるため，提案システムの方が高速に名前解決できることが期待される．この特性を踏まえて，本項では，既存システムとの秘匿に基づいて名前解決速度について評価する．

名前解決速度の評価には，フルサービスリゾルバによる問い合わせに対する権威サーバからの応答までの時間に基づいて評価する．提案システムにおいて，マネージャサービスは既存システムにおける TLD が担当する．図 18 で示すように，ルート権威サーバまでの RTT は 50 ミリ秒周辺が最も多く，平均すると 100 ミリ秒に収束する．他方で，図 19 で示すように，TLD 権威サーバまでの RTT は 13 ミリ秒周辺が最も多く，平均は 50 ミリ秒に収束することがわかる．既存システムでは，問い合わせるドメイン名のゾーン構成に従い，最終的な権威サーバまでの RTT が加算される．他方で，提案システムでは，TLD を想定するマネージャにフルサービスリゾルバから 1 ホップの問い合わせで名前解決が実現される．以上のことから，少なくても SLD 以降の権威サーバにかかる RTT 分高速に名前解決できることがわかる．

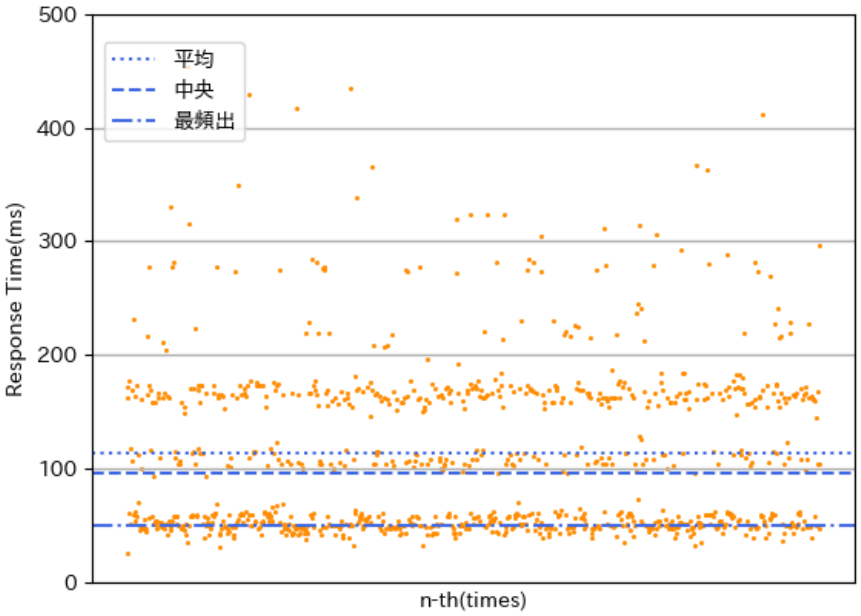


図 18　Root 権威サーバにおける RTT の分布

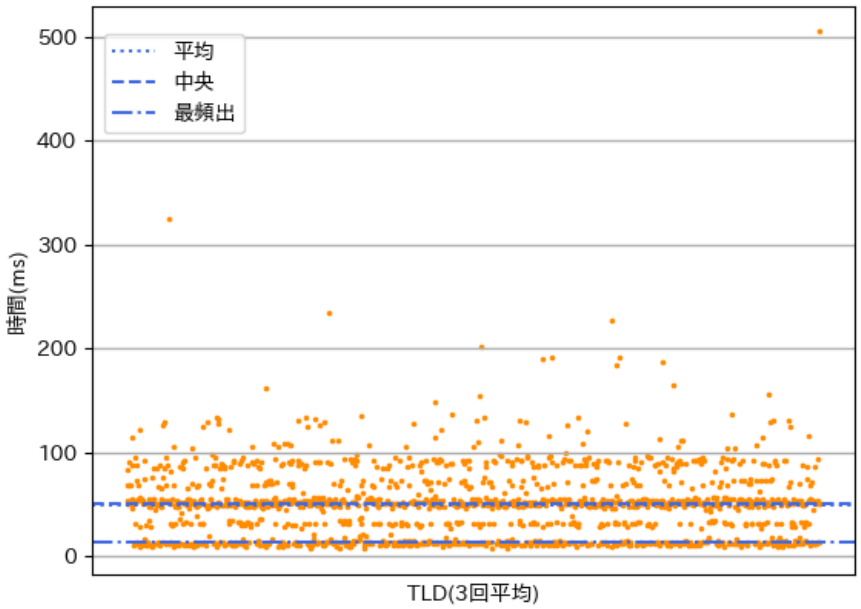


図 19　TLD 権威サーバにおける RTT の分布

6. 考察

本章では，提案システムにおける課題と有用性について考察する．

6.1 提案システムをバイパスするトンネリング手法

提案システムにはマネージャ以外のサービスノードにサーバ機能がないため，マネージャに脅威が及ばない限りトンネリングの機能を抑止することができる．しかし，何らかの手法によってマネージャが攻撃される場合，マネージャを通じてトンネリング通信される潜在的な可能性はある．そのような攻撃手法には，細工した DNS クエリを用いることが考えられる．クエリパケットを通じてマネージャの権限を取得する攻撃には，クエリを分解する機能にエスケープ処理を施す実装するにすることによって対処できると考えられる．

6.2 マイグレーション

基盤システムへの修正において，マイグレーションの実現性は重要である．既存システムから提案システムに移行させるプロセスでは，はじめに識別子の名前空間に関してどのマネージャがどの範囲をゾーンとするのかを決める．マネージャが保持するドメインの候補となるのは，一部のドメインを除いた gTLD である．2020 年 1 月時点で，1514 個のドメインが gTLD として登録されている．マネージャが保持するドメイン情報に基づき，そのドメインに関連したドメインを保持するプロバイダが連結されるため，マネージャのドメインはドメイン全体を分類し易い一般的なドメインであることが望ましい．そのため，一部の既存ドメインはプロバイダが保持するドメインに位置づけ，関連するマネージャドメインに連結させる処理を施していく．

次に，データベースを保持するマネージャを動作させ，既存システムにおける権威サーバによって管理されていたドメイン名とレコーとタイプの組みの情報について算出される識別子に基づき，その識別子をゾーンとするマネージャにコンテンツ情報をフィードしていく．マネージャにコンテンツ情報が貯められる状態

になると，識別子を Qname に含めることで希望の名前解決サービスを提供する基盤が整う．上記のマネージャへのコンテンツ情報のフィード処理と並行しながら，フルサービスリゾルバに識別子を導出し Qname に識別子を含める機能を Bind や Unbound をはじめとするフルサービスリゾルバのソフトウェアに実装する．これらの手続きを実現することによって，既存システムから提案システムの名前解決基盤に移行させることができる．

6.3 ハッシュ関数危殆化の影響とシステムの継続性

計算リソースの向上や解析アルゴリズムの効率化などによって，使用するハッシュアルゴリズムが危殆化する可能性がある．提案システムでは，ハッシュ関数の原像計算困難性に基づき，フルサービスリゾルバとサーバ間におけるクエリの機密性を確保している．ハッシュ関数が危殆化した場合，クエリパケット内の識別子から元のメッセージに復元されるというプライバシー侵害の脅威に発展する．このように提案システムの目的であるトンネリング抑止の機能に対しては影響はないものの，プライバシーに影響が及ぶ課題がある．フルサービスリゾルバとマネージャ間通信におけるトラフィックを暗号化させる仕組みが今後の課題である．

6.4 範囲に基づくゾーン分割に起因する脅威

提案システムでは，マネージャというサービスノードへの信頼に基づいて，ドメイン名とレコードタイプのペアに識別子を付与し，複数のマネージャが識別子が帰属する名前空間の範囲を協調的に管理することでコンテンツが管理される．この設計には，マネージャに障害が発生した際には極めて広範囲に影響が及ぶ潜在的な脅威がある．それには，ユーザとの単一接続点となるマネージャへの DoS 攻撃[*11]のアプローチが攻撃手法として考えられる．DoS 攻撃に対して，既存システムではラウンドロビンによる負荷分散の手法が考えられるが，提案システムで

[*11] DoS(Denial of Service) 攻撃: サービス不全攻撃．サーバに大量の問い合わせることによって，サーバの計算リソースを意図的に消費されサービス提供を不全にさせる攻撃手法．

は検討されていない．提案システムにおけるマネージャの負荷分散機能は，今後の課題である．

7. 結論

本論文では，DNS トンネリングを抑止する名前解決システムを提案した．DNS トンネリングが動作する仕組みを分析し，修正する対象として，スタブリゾルバから権威サーバまでクエリパケットが転送される仕組みに着目した．提案システム DNS-TD は，権威サーバのコンテンツを保持およびクライアントからの問い合わせに応答する機能とコンテンツを編集する機能を 2 つサービスノードに分割させる設計をとった．これによって，スタブリゾルバと任意の権威サーバ間でデータがやりとりされることが抑制され，DNS トンネリング通信発生の抑止に寄与する．評価では，実装した提案システムのプロトタイプ上での擬似トンネリング通信を発生させるシミュレーションテストに基づいて，トンネリング通信の抑止に対して本提案システムが有用であることを示した．また，特性評価の結果から，提案システムの名前解決におけるクエリ転送回数がスタブリゾルバからのサーバまで 2 回である点は，他のどの名前解決システムよりも少ない．すなわち，提案システムが高速な名前解決とトラフィック量を削減できるという特性を持っていることを明らかにした．

提案システムは，ドメインに依らないフラットな名前空間の範囲に基づきゾーンを分割する設計上，1 つのサーバが保持するゾーンには複数のドメインが含まれる．そのため，既存システムにおける権威サーバの障害では，その権威サーバが管理するドメインのみに影響が及ぶのに対し，提案システムではインターネット全体に影響が及ぶ潜在的な脅威がある．今後の課題は，この脅威の対処として負荷分散の仕組みを検討することである．

謝辞

ご指導ご鞭撻賜りありがとうございました.

参考文献

[1] M. Att&ck, "Custom command and control protocol." https://attack.mitre.org/techniques/T1094/. (accessed at 2020-1-28).

[2] KrebsonSecurity, "Deconstructing the 2014 sally beauty breach." https://krebsonsecurity.com/2015/05/deconstructing-the-2014-sally-beauty-breach/, May 2015. (accessed at 2020-1-28).

[3] IronNet, "Chirp of the poisonfrog." https://ironnet.com/blog/chirp-of-the-poisonfrog/, February 2019. (accessed at 2020-1-28).

[4] N. Hoffman, "Bernhardpos." https://securitykitten.github.io/2015/07/14/bernhardpos.html, July 2015. (accessed at 2020-1-28).

[5] FireEye, "Multigrain – point of sale attackers make an unhealthy addition to the pantry." https://www.fieeye.com/blog/threat-research/2016/04/multigrain_pointo.html, April 2016. (accessed at 2019-11-30).

[6] P. alto Networks, "New wekby attacks use dns requests as command and control mechanism." https://unit42.paloaltonetworks.com/unit42-new-wekby-attacks-use-dns-requests-as-command-and-control-mechanism/, May 2016. (accessed at 2019-11-30).

[7] Kaspersky, "Use of dns tunneling for c&c communications." https://securelist.com/use-of-dns-tunneling-for-cc-communications/78203/, April 2017. (accessed at 2019-11-30).

[8] C. Talos, "Spoofed sec emails distribute evolved dnsmessenger." https://blog.talosintelligence.com/2017/10/dnsmessenger-sec-campaign.html, October 2017. (accessed at 2019-11-30).

[9] Cylance, "Threat spotlight: Inside udpos malware." https://threatvector.cylance.com/en_us/home/threat-spotlight-inside-udpos-malware.html, Feburary 2018. (accessed at 2019-11-30).

[10] K. Born and D. Gustafson, "Ngviz: Detecting dns tunnels through n-gram visualization and quantitative analysis," in *Proceedings of the Sixth Annual Workshop on Cyber Security and Information Intelligence Research*, CSIIRW '10, (New York, NY, USA), p. 4, Association for Computing Machinery, 2010.

[11] C. Qi, X. Chen, C. Xu, J. Shi, and P. Liu, "A bigram based real time dns tunnel detection approach," *Procedia Computer Science*, vol. 17, pp. 852 – 860, 2013. First International Conference on Information Technology and Quantitative Management.

[12] J. Liu, S. Li, Y. Zhang, J. Xiao, P. Chang, and C. Peng, "Detecting dns tunnel through binary-classification based on behavior features," in *2017 IEEE Trustcom/BigDataSE/ICESS*, pp. 339–346, Aug 2017.

[13] A. Nadler, A. Aminov, and A. Shabtai, "Detection of malicious and low throughput data exfiltration over the dns protocol," in *Computers and Security*, vol. 80, pp. 36 – 53, 2019.

[14] J. Steadman and S. Scott-Hayward, "Dnsxd: Detecting data exfiltration over dns," in *2018 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, pp. 1–6, November 2018.

[15] J. Ahmed, H. H. Gharakheili, Q. Raza, C. Russell, and V. Sivaraman, "Monitoring enterprise dns queries for detecting data exfiltration from internal hosts," *IEEE Transactions on Network and Service Management*, pp. 1–1, 2019.

[16] E. Ekman, "iodine." http://code.kryo.se/iodine/. (accessed at 2020-1-28).

[17] R. Bowes, "dnscat2." https://github.com/iagox86/dnscat2. (accessed at 2020-1-28).

[18] P. Mockapetris, "Domain names - concepts and facilities." https://www.rfc-editor.org/info/std13, November 1987. RFC1034.

[19] P. Mockapetris, "Domain names - implementation and specification." https://www.rfc-editor.org/info/std13, November 1987. RFC1035.

[20] J. Klensin, "Internationalized domain names for applications (idna): Definitions and document framework." https://www.rfc-editor.org/info/rfc5890, August 2010. RFC5890.

[21] O. Pearson, "Bugtraq mailing list archives dns tunnel - through bastion hosts." https://seclists.org/bugtraq/1998/Apr/79. (accessed at 2019-1-4).

[22] M. V. Horenbeeck, "Dns tunneling." http://web.archive.org/web/20060709044338/www.daemon.be/maarten/dnstunnel.html. (accessed at 2020-1-28).

[23] D. Kaminsky, "Ozymandns." https://dankaminsky.com/2004/07/29/51/. (accessed at 2020-1-28).

[24] D. Kaminsky, "Reverse dns tunneling staged loading shellcode." https://www.blackhat.com/presentations/bh-usa-08/Miller/BH_US_08_Ty_Miller_Reverse_DNS_Tunneling_Shellcode.pdf. (accessed at 2020-1-28).

[25] A. Revelli and N. Leidecker, "Heyoka." http://heyoka.sourceforge.net/. (accessed at 2020-1-28).

[26] Tim, "Tcp-over-dns." http://analogbit.com/software/tcp-over-dns/. (accessed at 2020-1-28).

[27] R. Bowes, "dnscat." https://wiki.skullsecurity.org/Dnscat. (accessed at 2020-1-28).

[28] M. Dornseif, “Denise.” https://github.com/mdornseif/DeNiSe. (accessed at 2020-1-28).

[29] SensePost, “Dns-shell.” https://github.com/sensepost/DNS-Shell. (accessed at 2020-1-28).

[30] Sturt, “Dnsbotnet.” https://github.com/magisterquis/dnsbotnet. (accessed at 2020-1-28).

[31] F. Ceratto, “Dnscapy.” https://github.com/FedericoCeratto/dnscapy. (accessed at 2020-1-28).

[32] iceman, “dohtunnel.” https://github.com/jansect/dohtunnel/blob/master/doh/doh.go. (accessed at 2020-1-28).

[33] SensePost, “godoh.” https://github.com/sensepost/goDoH. (accessed at 2020-1-28).

[34] SpiderLabs, “Dohc2.” https://github.com/SpiderLabs/DoHC2. (accessed at 2020-1-28).

[35] MagicTunnel, “magictunnelandroid.” https://github.com/MagicTunnelM/agicTunnelAndroid. (accessed at 2020-1-28).

[36] “dns2tcp.” https://github.com/alex-sector/dns2tcp. (accessed at 2020-1-28).

[37] L. Nussbaum, “Tuns.” https://github.com/lnussbaum/tuns. (accessed at 2020-1-28).

[38] S. Josefsson, “The base16, base32, and base64 data encodings.” https://www.rfc-editor.org/info/rfc4648, October 2006. RFC4648.

[39] 森下泰宏 and 尾崎勝義, “Dns 運用の「見抜く」を探る～インシデント事例の紹介と必要な要素・項目～ランチのおともに dns.”

https://jprs.jp/tech/material/iw2016-lunch-L3-01.pdf. (accessed at 2020-1-28).

[40] V. Paxson, M. Christodorescu, M. Javed, J. Rao, R. Sailer, D. L. Schales, M. Stoecklin, K. Thomas, W. Venema, and N. Weaver, "Practical comprehensive bounds on surreptitious communication over DNS," in *Presented as part of the 22nd USENIX Security Symposium (USENIX Security 13)*, (Washington, D.C.), pp. 17–32, USENIX, 2013.

[41] D. Herrmann, C. Banse, and H. Federrath, "Behavior-based tracking: Exploiting characteristic patterns in dns traffic," *Comput. Secur.*, vol. 39, p. 17–33, November 2013.

[42] NIST, "Managing information security risk: Organization, mission, and information system view." https://www.nist.gov/publications/managing-information-security-risk-organization-mission-and-information-system-view, March 2011. Special Publication (NIST SP) - 800-39.

[43] T. Reddy and P. Patil, "Dns over datagram transport layer security (dtls)." https://www.rfc-editor.org/info/rfc8094, February 2017. RFC8094.

[44] P. Hoffman and P. McManus, "Dns queries over https (doh)." https://www.rfc-editor.org/info/rfc8484, October 2018. RFC8484.

[45] C. Russ, M. Athicha, and R. T. Morris, "Serving dns using a peer-to-peer lookup service," in *Peer-to-Peer Systems* (P. Druschel, K. Frans, and A. Rowstron, eds.), (Berlin, Heidelberg), pp. 155–165, Springer Berlin Heidelberg, 2002.

[46] F. Dabek, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica, "Wide-area cooperative storage with cfs," in *SOSP01: 18th Symposium on Operating System Principles*, (New York, United States), Association for Computing Machinery, October 2001.

[47] P. Danielis, V. Altmann, J. Skodzik, T. Wegner, A. Koerner, and D. Timmermann, "P-donas: A p2p-based domain name system in access networks," *ACM Transactions on Internet Technology (TOIT)*, vol. 15, pp. 1–21, September 2015.

[48] Y. Song and K. Koyanagi, "Study on a hybrid p2p based dns," in *2011 IEEE International Conference on Computer Science and Automation Engineering*, vol. 4, pp. 152–155, June 2011.

[49] W. Matthias, S. Martin, and G. Christian, "A censorship-resistant, privacy-enhancing and fully decentralized name system," in *Cryptology and Network Security* (D. Gritzalis, A. Kiayias, and I. Askoxylakis, eds.), (Cham), pp. 127–142, Springer International Publishing, 2014.

[50] A. Davoudian, L. Chen, and M. Liu, "A survey on nosql stores," *ACM Comput. Surv.*, vol. 51, pp. 1–43, April 2018.

[51] S. Bortzmeyer, "Dns query name minimisation to improve privacy." https://www.rfc-editor.org/info/rfc7816, March 2016. RFC7816.

付録

A. レコードタイプの使用分布

表 17 レコードタイプの使用分布

タイプ	パケット数	割合
A	236,210,050	54.778
AAAA	149,322,427	34.629
PTR	43,060,608	9.986
SRV	1,497,622	0.347
MX	474,827	0.110
ANY	281,023	0.0657
SOA	226,975	0.053
TXT	115,300	0.027
NS	12,028	0.003
TKEY	4518	0.001
NAPTR	4281	0.001
SPF	512	0.000
CNAME	196	0.000
AXFR	2	0.000
NULL	2	0.000
合計	431,210,371	100.000

B. 発表リスト (口頭発表)

1. 高須賀 昌烈, 妙中 雄三, 門林 雄基, “非実在ドメインに対するネガティブキャッシュの拡張と再帰問い合わせハッシュ化の提案,” 電子情報通信学会情報ネットワーク研究会, 2019-10-ICTSSL-IN, 2019 年 10 月.