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

1 概要

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．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．

2 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．

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 [2–7]．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 [8] やDNSCat2 [9] 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ヶ月間などに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 [5]．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．

2.1 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

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．

2.2 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．

3 脅威モデル

3.1 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 [5]．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．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．

3.1.1 DNS Exfiltration

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．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，“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 を用いている [10]．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 1 で示す通りである．

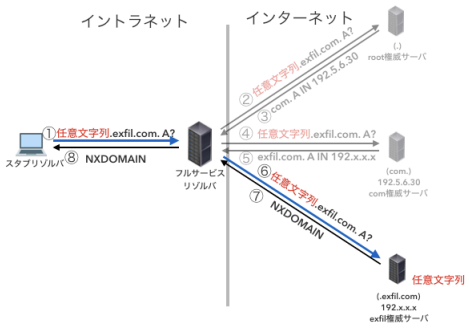

図 1: ドメイン名のラベル部に任意の文字列が注入された DNS クエリが，インターネット上の権威サーバ (“exfil.com”) に転送される様子．

3.1.2 DNS Infiltration

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．

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･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．

$$www.exfil.com. \quad IN \quad TXT \quad uname\ -ap \qquad (1)$$

TXT レコードタイプを用いて DNS Infiltration することを考える．(1) 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 2 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．

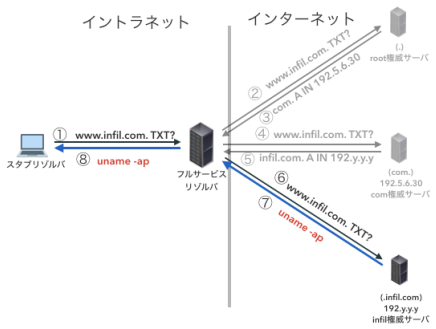

図 2: TXT レコードに登録された情報について，DNS クエリで問い合わせることで権威サーバから命令情報を取得している様子

3.2 DNS トンネリングへの既存対策

DNS トンネリングに対して，森下ら [11] 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．

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 の適用
3. DNS ファイヤーウォールの導入

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．

3.2.1 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 [12]．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．

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，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．実際のトンネリング実装における DNS Infiltration 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 [13] 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．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，問い合わせに対して，表 1 のようなステータス情報を応答する．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 1: 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

ドメイン名に含まれる文字列の出現頻度

Born ら [2] 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．

3.2.2 既存検知手法に対する脅威モデル

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 [2–7]．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 [8] のに対し，APT 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 [14, 15]．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 [16–18]．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．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．

4　関連研究

4.1　P2P に基づく名前解決システム

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 [19] 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 [20]．システムにおける名前解決の仕組みは，図 3 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．クライアントからの問い合わせ

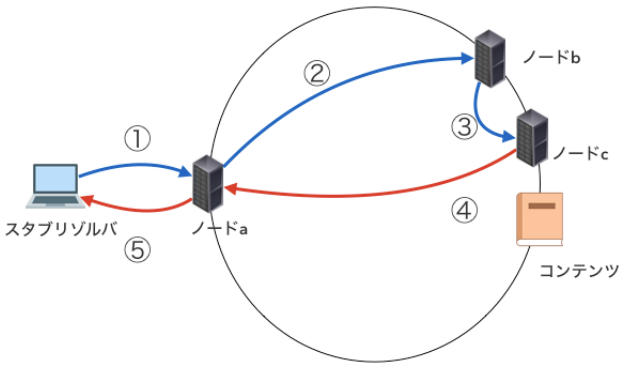

図 3: DDNS における名前解決プロセス

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 [21]．

Yiting ら [22] 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 4 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．

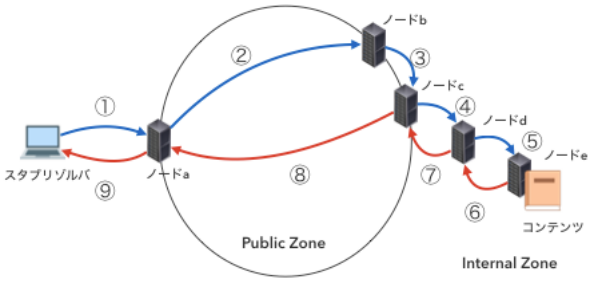

図 4: HDNS における名前解決プロセス

GNU [23] 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 5 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．

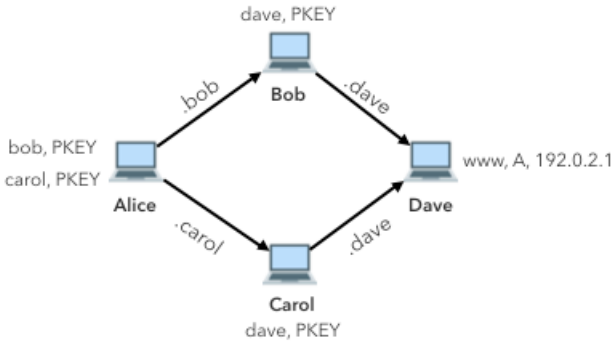

図 5: GNS における名前解決プロセス

4.2　課題

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．

5　提案システム

5.1　システムアーキテクチャ

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 6 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．

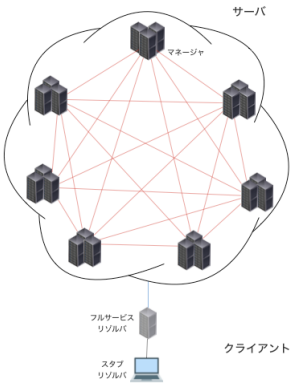

図 6: DNS-TD におけるクライアントサーバアーキテクチャ

5.2　サービスノード

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 5.1 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 7 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．

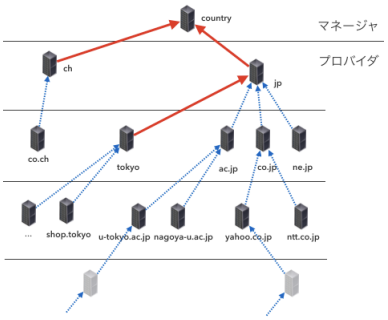

図 7: マネージャとプロバイダの関係

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．

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，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．

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 [24]．以上のことを踏まえ，DNS-TD では，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．

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，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．

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．

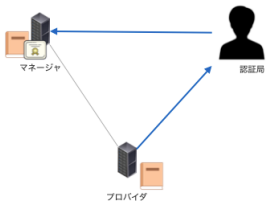

図 8: レコード情報操作におけるプロセスの概略図

5.3 識別子

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 5.2 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．

5.3.1 ハッシュアルゴリズム

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 2 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 2: 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 1 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．

Algorithm 1: 識別子の導出方法

```
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
```

5.3.2 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 3 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．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 3: 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
…	…	…

6 評価

6.1 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 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 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．

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 9 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．

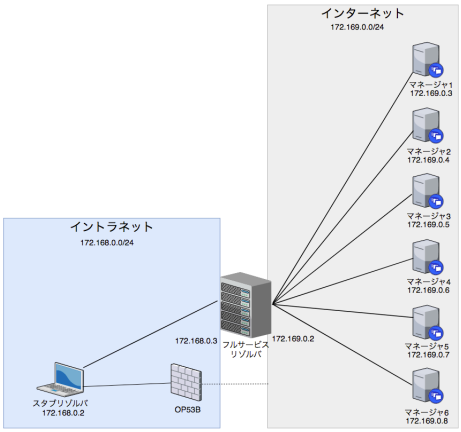

図 9: Docker 環境内におけるネットワークトポロジー

6.2 DNS トンネリング

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．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．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．

6.3 特性評価

6.3.1 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．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 [25] 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．

$$
\begin{aligned}
&\text{(n Label's Traffic)} \\
&= (R + (Label\ Length) + r + Rtype + O) \times n \\
&= (1 + (Label\ Length) + 1 + 1 + 68) \times n \\
&= (71 + (Label\ Length)) \times n
\end{aligned}
$$

表 4: 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	1
r	Root を表す “.”	1
O	Qname 以外	68
Rtype	レコードタイプ	1(A) 2(NS)

図 10 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．

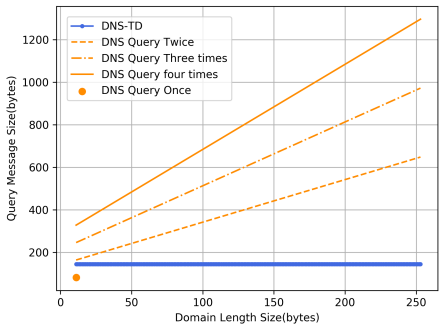

図 10: DNS-TD と DNS における名前解決に使用されるクエリパケットサイズの比較

図 10 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．

6.3.2 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 11 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 5 の通りである．

表 5: 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 11 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 12 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．

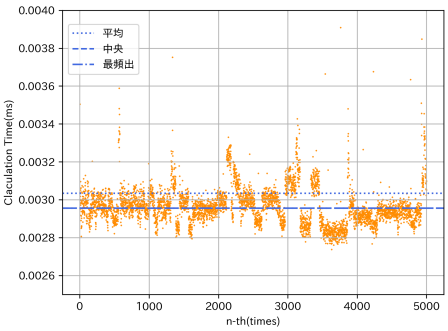

図 11: コンテンツ ID とドメイン ID 導出にかかる計算時間のオーバーヘッド

6.3.3 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 12 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 13 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．

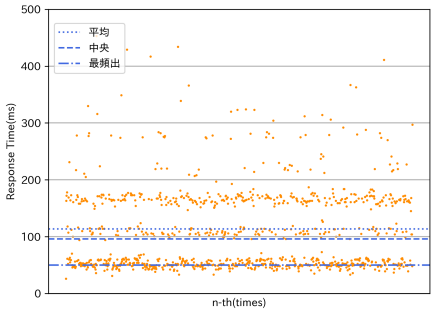

図 12: Root 権威サーバにおける RTT の分布

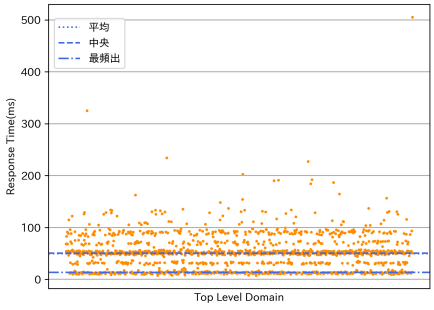

図 13: TLD 権威サーバにおける RTT の分布

7 結論

本研究では，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 を利用した秘匿通信，DNS

トンネリングに対する既存の検知に基づく対策アプローチにはペイロードやトラフィックを調整することで迂回される課題があった．DNS トンネリング，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クエリの透過性に起因する課題であることについて，動作の仕組みを分析することによって明らかにした．クエリ透過性を抑止するため，既存システムにおける権威サーバのレコード情報を保持しクライアントに応答する機能とレコード情報を編集する機能を分離させるという新しい名前解決の仕組みを提案した．提案したアプローチは，クライアントからの名前解決問い合わせをマネージャという代表サーバが任意に設置できるノードに代わって応答することで，任意ノードへデータが転送されるクエリの透過性を抑止することに寄与する．また，クエリが透過しない名前解決システムを分析し，遅延とネットワーク構築にかかるトラフィック量が増加する課題があることを明らかにした．この課題を踏まえ，提案システムでは，マネージャの探索方法にハッシュテーブルを採用することで遅延に対処し，またマネージャの数を固定することによってネットワーク構築にかかるトラフィックが発生させないアーキテクチャを採用することでトラフィックの課題に対処させた．

クエリ透過性の評価にあたり，提案システムの各サービスノードを実装し，Docker による仮想環境上で名前解決が行える実験環境を用意した．上記の環境で，擬似的に作成した長さと文字列がランダムな Qname をもつ DNS クエリパケットをクライアントから問い合わせるというシミュレーションを実施し，シミュレーションテストの結果に基づき，提案システム上ではクライアントから任意のサーバにクエリが転送されないことを示した．提案システム独自の名前解決における識別子導出処理に関するオーバーヘッドの可能性ついて評価を行った．評価では，A をレコードタイプとしながらランダムな文字列で構成される長さの異なる 5000 個の Qname のペアを用意し，このペアにおける識別子導出にかかる時間を計測した．計測の結果，中央値が約 0.003 ミリ秒と無視できる程度であることを実験に基づき示した．また，ルート権威サーバと既存の TLD 権威サーバの RTT の調査に基づき，提案システムにおけるクエリに対する応答速度の推定を行った．提案システムにおける名前解決の応答速度について，ルート権威サーバの RTT である約 50-100 ミリ秒に加え，委譲数 n($n \in \mathbb{N}$) に対して n-1 個の権威サーバの RTT の分だけ高速化することを理論に基づき示した．加えて，既存システムにおける長さと委譲数の異なる 253 個の Qname を用意し，この Qname をおける名前解決時のトラフィック量について，既存システムとの比較に基づき提案システムの方が名前解決にかかるトラフィック量が少ないことを実験に基づき示した．トンネリング抑止の機能だけでなくシステム特性の評価の結果から，提案システムの有用性を示した．

今後の課題は，実ネットワークを想定したトラフィックシミュレーションに基づく提案システムの負荷耐性の分析により，マネージャノードに求められる計算リソースおよび数を推定することである．

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] 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.

[3] 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.

[4] 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.

[5] 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.

[6] 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.

[7] 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.

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

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

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

[11] 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).

[12] 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.

[13] 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.

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

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

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

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

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

[19] C. Russ, M. Athicha, and R. T. Morris, "Serving dns using a peer-to-peer lookup service," in *Peer-to-Peer Systems*, pp. 155–165, Springer Berlin Heidelberg, 2002.

[20] 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.

[21] 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.

[22] 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.

[23] 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.

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

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