Vol.2012-MBL-64 No.17
Vol.2012-ITS-51 No.17
2012/11/16

DNSトラヒックによるPassive OS Fingerprinting手法の提案

松中 隆志[1,a)]　山田 明[2]　窪田 歩[1]

概要：本稿では，DNS トラヒックの解析から OS fingerprinting を実現する手法を提案する．本手法では OS ごとにみられる DNS クエリに関する特徴を利用して OS fingerprinting を行い，ネットワーク内に存在する端末数を OS ごとに推定する．当該特徴は，各 OS にはそれぞれその OS のみしかクエリを送信しないドメインがある，各 OS は固有の時間間隔で定期的に当該ドメインに関するクエリを送信する，の 2 つである．本手法はまた DNS クエリの送信時間間隔がずれる確率も特徴として端末数の推定の際に加味することで，推定結果の精度を向上させる．本稿では，端末から送信される DNS トラヒックの解析を行い，OS fingerprinting に利用できる DNS クエリにおける特徴を抽出したので報告する．さらに，抽出した特徴を用いた端末数推定結果を検証すべく，社内イントラネットワーク内の DNS トラヒックを用いて実際に OS fingerprinting を行った結果について報告する．検証により，提案手法による推定結果は DHCP フレームを用いた fingerprinting の結果に近い推定が可能であることが確認された．

キーワード：OS Fingerprinting，トラヒック解析

Passive OS Fingerprinting by DNS Traffic Analysis

Abstract: In this paper, we propose a new passive OS fingerprinting method which requires analyzing only DNS traffic. The method utilizes characteristics on DNS queries that each OS sends DNS queries related to specific domains, and each OS sends these queries with specific patterns of time interval between them. The method can estimate the number of devices with each OS from the number of queries by utilizing the characteristics of the time interval patterns. The method considers the likelihood of irregular events that some queries are sent less than regular time intervals, and some other queries are sent more than regular time intervals. We analyze DNS traffic sent by each OS and extract the characteristics for OS fingerprinting. Then, we examine our estimation method by using DNS traffic in our intra-network. According to our examination, some results of our estimation method are close to the results of DHCP fingerprinting.

Keywords: OS Fingerprinting, Traffic Analysis

1. はじめに

近年，スマートフォンの普及などによりデータトラヒック量が爆発的に上昇している．増加傾向にあるトラヒック需要に的確に対応するために，ネットワーク管理者は自身の管理ネットワークの利用者がどのようなサービス・アプリケーションをどれくらい使っているか，デバイス，OS ごとの利用率はどれくらいか，トラヒック需要のピークはいつか，トラヒック需要の多い場所・スポットはどこかを把握する必要がある．ネットワーク管理において OS の利用率を把握することは重要である．Ericsson の報告 [1] では，搭載されている OS によって発生するトラヒックの傾向が異なる．また，搭載 OS が異なればインストールされるアプリも異なるため，それに付随して発生トラヒックの傾向も異なる．しかしながら，昨今のアクセスネットワークは多様化しており，さらにトラヒックの off-load のためのセルラ網–無線 LAN への移動の頻発，スマートフォン，モバイルルータを介したテザリングの普及などにより，管

[1] (株)KDDI 研究所
KDDI R&D Laboratories, Inc., Ohara 2–1–15, Fujimino, Saitama, 356–0003, Japan
[2] KDDI(株)
KDDI CORPORATION
[a)] ta-matsunaka@kddilabs.jp

理ネットワークの状況把握は容易ではなくなっている．

管理ネットワークの状況を把握するために，これまで多くの研究がなされてきた．[11]，[12] では，ネットワーク内のトラヒックの解析によってネットワーク利用者のアクティビティを把握し，利用者のタイプを分類している．[2]，[3] では，ネットワークを流れるトラヒックから当該トラヒックの発生元端末に搭載されている OS を判定 (OS fingerprinting) している．[2] では，TCP/IP ヘッダにみられる OS ごとの特徴から OS fingerprinting を行っている．[3] では，DHCP(Dynamic Host Configuration Protocol) のリクエストフレーム内の特定のフィールドの情報を利用して OS fingerprinting を行っている．さらに [4]，[6] では，搭載 OS を検出したい端末に対して能動的にプローブパケットを送信し，その返答パケットを観察することで OS fingerprinting を実施している．[10] は，上記の受動的なトラヒック監視と能動的な監視を両方活用するハイブリッドな手法を用いている．しかしながら [3] を除くこれらの手法は，昨今の大規模かつ複雑化されたネットワークへの適用を考えると，解析に利用するトラヒックデータが膨大となるためストレージコスト，解析コストにおいて現実的ではない．また，解析に必要なデータを収集するための装置を各アクセスネットワークに設置する必要がある．[3] においては，DHCP フレームのみを利用するため適用可能だと考えられるが，利用者のアクティビティなどネットワークの利用状況に係る付加的な情報が得られない．さらに，[4]，[6] では，プローブパケットを全端末に送信する必要があるため，送信を行うシステム構築のための手間，および監視目的のために能動的に利用者の端末へパケットを送信することへの利用者の抵抗感などにより，大規模かつ複雑化されたネットワークへの適用は困難である．受動的な監視による手法 [2]，[3] では，NAT(Network Address Transform) 環境，テザリングなど複数の端末で同一の IP アドレスを共有するような状況化において正確に状況を把握できない．ネットワークの監視・管理に係る負荷を軽減すべく，[7]，[8]，[9] ではより少ない監視ノードでネットワークの状況把握を可能にしようとする手法が提案されている．これらの手法では，センサネットワークや仮想ネットワークのように動的にネットワークの構成が変化する状況下でも効果的に監視・管理を行うことができる．しかしながら，これらの手法ではそれぞれの手法に対応した監視ノードをネットワーク全域に配置する必要があり，既配備の装置の置換，改修が必須となる．

以上のようなネットワーク監視・管理に係る困難性を鑑み，著者らは DNS(Domain Name System) トラヒックの監視によってネットワークの利用状況を把握することに着目した．DNS トラヒックの監視によって，ネットワーク管理者は様々な情報を得ることが期待できる．DNS トラヒックの監視は，ネットワーク内の DNS サーバ付近でトラヒックを監視するのみで実現できるため，監視設備・機能の追加が少なくすみ，大規模かつ複雑化されたネットワークにおいても実現可能である．例えば DNS クエリ内のドメイン名から利用者が利用するアプリケーション・サービスに関する情報を得ることができる．また，DNS クエリの発生量からアプリケーション・サービスのトレンド，発生トラヒック量の傾向を把握できる．さらに NAT 環境など IP アドレスが共有される状況においても，DNS トラヒックは各端末から個別に送信されるため，当該環境においても状況の把握が可能である．

本稿では，DNS トラヒックの受動的な監視による OS fingerprinting 手法を提案する．本手法は，DNS クエリにみられる OS ごとの特徴を用いる．各 OS にはそれぞれ，他の OS が DNS クエリを送信しない固有のドメインがあり，当該ドメインに対する DNS クエリは一定の時間間隔で送信される特徴がみられる．本手法ではそのような特徴を用いて，ある期間内に取得された特定のドメインに対する DNS クエリの数から当該期間内にネットワークに在圏した搭載 OS ごとの端末数を推定する．DNS クエリの送信周期においては，特徴として抽出された送信周期から外れて DNS クエリが送信される事象もみられる．本手法では，そのような送信周期から外れる事象も加味して端末数を推定する．本手法の実現性を検証すべく，本稿ではまず搭載 OS ごとに送信される DNS トラヒックを解析し，上記のような特徴を抽出した．さらに本稿では，抽出した特徴を用いて社内イントラネットワーク内の DNS トラヒックから端末数を推定した．検証結果から，本手法によって DHCP フレームによる OS fingerprinting での推定台数に近い値を得られることを確認した．

本稿の構成を説明する．2 節では，本稿の内容に関連する研究・技術について述べる．3 節では，提案手法の前提条件などについて述べる．4 節では，DNS トラヒックの解析結果と抽出された DNS トラヒックの特徴について述べる．さらに抽出された特徴を利用して OS ごとの端末数を推定する手法について述べる．5 節では，提案手法を用いて社内イントラネットワーク内に存在する端末数を推定した結果について述べる．6 節で本稿の内容をまとめ，今後の課題について言及する．

2. 関連研究

2.1 OS fingerprinting

OS fingerprinting の手法は受動的な手法と能動的な手法に大別される．

Zalewski[2] は受動的にトラヒックの TCP/IP ヘッダを監視して，当該ヘッダ内の TTL(Time To Live)，MSS(Maximum Segment Size) フィールドにみられる特徴をもとに OS fingerprinting を行う手法を考案した．HTTP ヘッダにおいては，User–Agent フィールドに利用者の Web

ブラウザの情報とともに OS の情報が記載されていることがあり，その情報を用いて OS fingerprinting を行うことができる．Shah[5] は User–Agent フィールドなど HTTP レスポンスヘッダ内の情報を用いて Web サーバのソフトウェアを判別する手法を考案した．しかしながらこれらの手法は，トラヒックデータの取得のために管理ネットワークの外部へのゲートウェイや全アクセスネットワークなど多くの観測点にトラヒックを監視するノードを設置する必要がある．さらに，各装置で取得された膨大なトラヒックデータから必要な情報を抽出して OS fingerprinting を実施する必要がある．そのため，昨今の大規模かつ複雑化されたネットワークへの適用は難しい．さらに [2] においては，NAT やテザリングなど，TCP/IP ヘッダが介在する装置，端末によって書き換えられる場合においては正しく OS fingerprinting を実施できない．Kollmann[3] は DHCP フレーム内の情報を OS fingerprinting に利用する手法を考案した．当該手法では，DHCP フレーム内の情報にみられる特徴の他に，DHCP フレームの再送時間間隔にみられる特徴を利用して OS fingerprinting を行う．しかしながら，DHCP フレーム内には利用者が利用しているアプリケーション・サービスに係る情報は含まれていないため，このようなネットワークの利用状況に係る情報を取得することができない．

能動的な OS fingerprinting の手法も多く提案されている．NMAP[6] はネットワークのスキャニングを行うツールである．NMAP には remote OS fingerprinting 機能も備えており，プローブパケットをターゲットとなる端末に送信してその返答パケットを監視することで OS fingerprinting を実施する．Gagnon[10] は受動的なトラヒックの監視による手法と能動的なプローブパケットの送信による手法の両方を適用したハイブリッドな手法を提案した．しかしながらこれらの能動的な手法においては，ターゲットの端末との間にファイアウォールや NAT 装置，テザリング端末などが介在している場合に，プローブパケットをターゲットの端末に送信できないため OS fingerprinting を実施できない．NAT やテザリング環境における OS fingerprinting 手法もいくつか提案されている．Beverly[13] は NAT 配下のホストを検出する受動的な手法を提案した．Beverly は NAT 配下のホストの TCP/IP ヘッダの特徴を検出するために Naïve Bayes による分類を行っている．Schulz[4] はテザリング環境下における能動的な手法を提案した．Schulz は ICMP(Internet Control Message Protocol) のエラーパケットをターゲット端末の TCP セッションに挿入し，その返答の振る舞いにおける特徴から OS fingerprinting を実施している．しかしながら，これらの手法においても監視ノードの設置コストの問題，プローブパケットの送信システムの実現における困難性の問題がある．

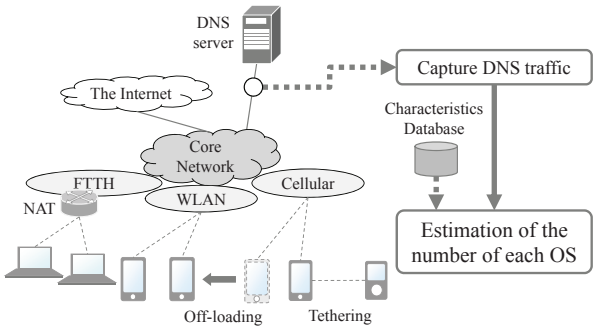


図 1: 想定ネットワーク環境
Fig. 1 Assumption of the network environment

2.2 トラヒック解析

ネットワーク内のトラヒックを解析することで，利用者のサービス・アプリケーションの利用状況を把握する手法がいくつか提案されている．Xu ら [11] はインターネットの backbone のトラヒックを解析し，トラヒックの送信元・送信先アドレスを server/services，heavy hitter hosts，scans/exploits に分類した．しかしながら，Xu らの手法においても監視ノードの設置コストの問題がある．Zhang ら [12] は無線 LAN における MAC(Medium Access Contorl) 層のトラヒックを解析し，伝送レート，フレームの送信間隔などの特徴から，Web ブラウジング，オンラインゲーム，動画ストリーミングなど利用者のアクティビティを推定する手法を提案した．Zhang らの手法においては，対象が無線 LAN のトラヒックに特定されている．また膨大なトラヒックデータを収集，解析する必要があり，ストレージコスト，解析にかかるコストの面で大規模なネットワークには適さない．

3. 提案手法

図 1 に提案手法で想定するネットワーク環境を示す．ネットワークはいくつかのアクセスネットワーク (セルラ，無線 LAN など) とコアネットワークで構成されており，利用者の端末はアクセスネットワークを介して The Internet に接続することができる．コアネットワークには DNS サーバ (群) が設置されており，端末はどのアクセスネットワークを利用していても当該 DNS サーバへ DNS クエリを送信する．また，端末は NAT やテザリングのように他の端末，装置を介してアクセスネットワークに接続される場合も想定される．そのため IP アドレスは複数の端末で共有されることがある．また，端末に割り当てられる IP アドレスは割り当てられたアドレスの有効期限が切れる，利用するアクセスネットワークが変わるなどで頻繁に変わる．例えばトラヒックの off-load のために，セルラから無線 LAN にアクセスネットワークを変更する場合などが該当する．

本稿で提案する OS fingerprinting 手法の流れを以下に示す．

(1) 試験環境下で各 OS 搭載端末から送信される DNS トラヒックを収集する．
(2) 収集された DNS トラヒックから，各 OS 特有の DNS クエリ対象ドメイン，当該ドメインに関する DNS クエリの送信周期といった特徴を抽出する．
(3) 抽出された特徴を OS ごとにデータベースなどに格納する．
(4) 実際のネットワーク環境において DNS トラヒックをキャプチャし，収集された DNS トラヒックからネットワーク内に存在する端末数を OS ごとに推定する．

次の節以降では OS から送信される DNS トラヒックの解析結果，および解析により抽出された特徴を用いて Android OS を搭載した端末の数を推定した結果について述べる．4 節では，DNS トラヒックの解析により抽出された特徴の例として，Android OS の DNS クエリの送信における特徴について述べる．解析によって，Android OS は DNS クエリにおいて以下の特徴を有することが確認された．(A) Android OS は他の OS がクエリを送信しないようなドメイン (`android.clients.google.com`) を有する．(B) Android OS は当該ドメインに関する DNS クエリを送信する際に，最初に `AAAA` レコードのクエリを送信し，次に `A` レコードのクエリを送信する．その後レスポンス内に記載された IP アドレスの仲の 1 つに対して `PTR` レコードのクエリを送信する．(C) Android OS は当該ドメインへの DNS クエリを，特定の送信周期ごとに送信する．特徴 (A)，(B) については 4.1 節，(C) については 4.2 節で詳しく述べる．特徴 (A)，(B) は該当する DNS クエリ単体で OS を判定できる特徴である．特徴 (C) は，DNS トラヒックの収集期間が当該クエリの送信周期よりも短い場合においても，当該期間内に存在する OS 搭載端末の数を推定するのに有益な特徴である．送信周期の特徴を利用した端末数の推定方法については 4.3 節で詳しく述べる．

提案方式による端末数の推定の有効性を，実際に社内のイントラネットワークに流れる DNS トラヒックを用いて推定した結果については 5 節で述べる．

4. DNS トラヒックの解析結果

OS fingerprinting に利用する DNS トラヒックの特徴を抽出するために，OS から送信される DNS トラヒックを収集し解析を行った．各 OS の DNS トラヒックを収集するために 4 台のスマートフォン (内 2 台は Android OS 端末) を用いた．スマートフォンは無線 LAN を介して The Internet に接続し，アプリケーションのバックグラウンドでの同期，GPS(Global Positioning System) 機能を有効にして，操作を行わない状態での DNS トラヒックを収集した．DNS トラヒックは DNS サーバ上でキャプチャした．

表 1: Android OS における固有ドメイン名
Table 1 Specific domain names for the Android OS

domain names	`android.clients.google.com`
	`mtalk.google.com`

4.1 OS 特有の DNS クエリ

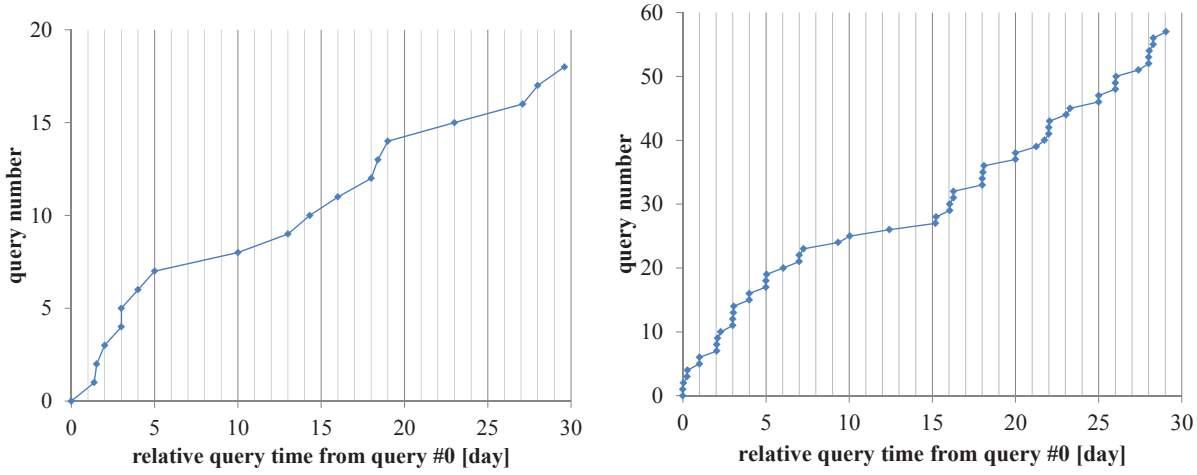
まず収集された DNS トラヒックから，OS ごとに他の OS が DNS クエリを送信しないようなドメイン名を抽出した．**表 1** に Android OS において抽出された Android OS 固有のドメイン名を例示する．さらに，Android OS が `android.clients.google.com` に DNS クエリを送信する際，以下のような動作をすることが確認された．まず，Android OS は当該ドメインに関する `AAAA` レコードのクエリを送信する．次に Android OS は当該ドメインに関する `A` レコードのクエリを送信する．そして DNS サーバからの回答が届いてから平均して 200 ミリ秒後に，回答として送られてきた `android.clients.google.com` の IP アドレスに対して `PTR` レコードのクエリを送信する．この `PTR` レコードクエリを送信する動作は，ほかの `google.com` のサブドメインに対しても観測された．

4.2 DNS クエリの送信周期

次に OS 固有のドメインに対する DNS クエリの送信間隔の周期性を解析する．**図 2** は，Android OS における `android.clients.google.com` へのクエリ 20 日間分における送信時刻を示す．図 2 では最初に観測された当該ドメインへの DNS クエリの送信時刻を 0 としている．端末 1 と 2 はいずれも Android OS 2.3 を搭載しているが異なるベンダによって製造された端末である．図 2 を見ると，DNS クエリは毎日同じ時刻に送信される傾向があることが確認できるが，その一方で 1 日の間で 1 回もクエリを送信しない日や 1 日の間で複数回クエリを送信する日が存在することも確認できる．端末 2 に関しては，クエリを送信したおよそ 3 秒後に再びクエリを送信し，そのさらに 1 時間後 (3,600 秒後) にもう一度クエリを送信するような現象が確認された．

図 3 に，`android.clients.google.com` に対するクエリの 60 日間分を用いて，送信間隔を集計した結果を度数分布で示す．**図 3a** に示すとおり，端末 1 においては 21.4%のクエリがおよそ 86,400 秒 (84,600〜88,200 秒) 後に送信されていた．また 42.9%のクエリが 1 日以上 (88,200 秒〜) 間隔を空けて送信されていた．上記の結果より，端末 1 においては，1 日ごとにクエリを送信する傾向があるが，1 日以上送信間隔を空けてクエリを送信する場合が多々ある傾向にあると推測される．

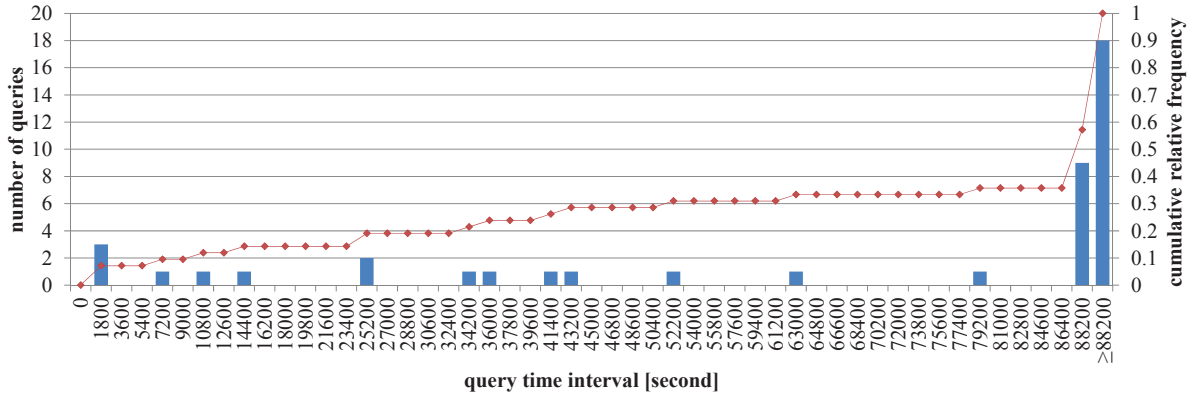
一方，端末 2 においては，**図 3b** に示すとおり，およそ 86,400 秒 (84,600〜88,200 秒) の間隔で送信されているク



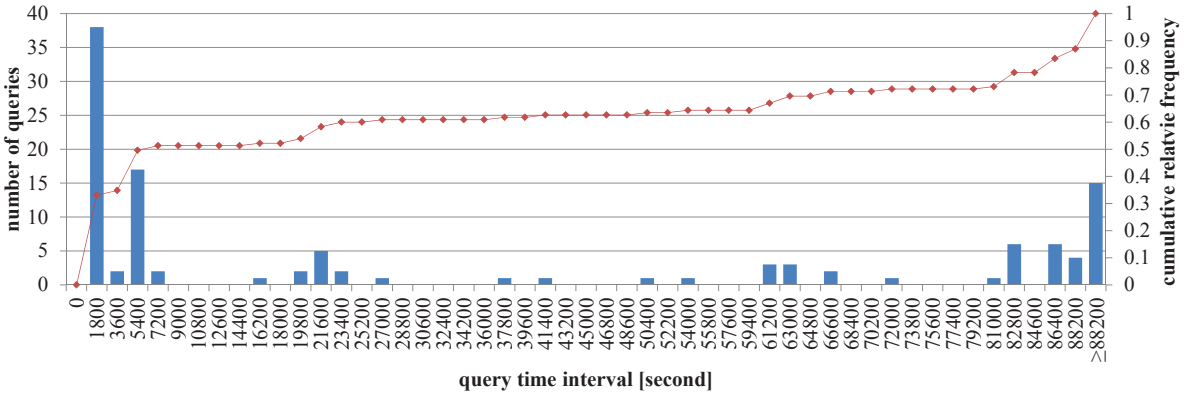
(a) 端末 1 Device 1　　(b) 端末 2 Device 2

図 2: DNS クエリの送信時刻 (*android.clients.google.com*)

Fig. 2 DNS query evolving time (*android.clinets.google.com*)



(a) 端末 1 Device 1



(b) 端末 2 Device 2

図 3: DNS クエリ送信間隔の度数分布 (*android.clients.google.com*)

Fig. 3 Distribution of query time interval (*android.clients.google.com*)

エリは 8.7%，1 日以上 (88,200 秒〜) の間隔で送信されているクエリは 13.0%であった．端末 2 では，33.0%のクエリが 1,800 秒以下 (そのうちのほとんどのクエリは 3 秒以下) の間隔で送信され，16.5%のクエリが 3,600 秒〜5,400 秒の間の間隔で送信されていた．また端末 2 では，6.1%のクエリが 82,800 秒 (81,000〜82,800 秒) 程度の間隔で送信されていた．上記の結果と図 2 の結果から，端末 2 においてはクエリが送信された約 3 秒後に再びクエリを送信し，その 3,600 秒から 5,400 秒後にもう一度クエリを送信した後，さらにその 82,800 秒後 (初めのクエリからおよそ 1 日

後) にクエリを送信する傾向があると推測される．

以上の解析結果から抽出された Android OS の DNS クエリに関する特徴をまとめると以下となる．

- `android.clients.google.com` など他の OS が DNS クエリを送信しない固有のドメインを持つ．
- `android.clients.google.com` の `A` レコードの DNS クエリを送信した 200 ミリ秒ほど後に，先のクエリに対するレスポンスで返答された IP アドレスについて `PTR` レコードのクエリを送信する．
- `android.clients.google.com` の DNS クエリを毎日ほぼ同時刻に送信する．しかし，当該ドメインのクエリを 1 日以上送信しないことがある (端末 1 では 42.9%，端末 2 では 13.0%)．
- `android.clients.google.com` の DNS クエリを上記の時刻とは違う時刻でも送信する．その送信パターンは端末によって異なる．
  - 例えば，端末 2 においては 1 日に 3 回当該ドメインのクエリを送信する．クエリを送信した約 3 秒後に再度送信し，そのおよそ 5,400 秒後に送信し，さらにそのおよそ 82,800 秒後 (最初のクエリから 1 日後) に送信する．

4.3 OS 搭載端末数の推定

前節で抽出された送信間隔の特徴から，端末数を推定するためには (A) DNS クエリ送信時間には周期性がある，(B) DNS クエリの送信間隔は想定された周期よりも短い時間となる場合がある，(C) DNS クエリの送信間隔は想定された周期よりも長い時間となる場合がある，の 3 つの特徴を考慮する必要がある．さらに，推定に用いる DNS トラヒックデータが，DNS クエリの送信周期より短い期間で収集されたものである場合を考慮する必要がある．これは，DNS トラヒックデータが収集されている間アクティブであったにもかかわらず，対象となる OS 固有のクエリを送信しない場合が存在することを意味している．この節では，はじめに上記のように DNS トラヒックデータの収集期間が，推定に用いる DNS クエリの送信周期よりも短い場合においても，端末数の推定を可能にする方法について述べる．次に DNS クエリの送信間隔が，特徴として想定していた送信周期からずれる場合 (上記の特徴 (B) および (C)) を考慮して端末数を推定する方法について述べる．以下，後述の説明のため**図 4** を例として用いる．また後述の説明で使用する記法を**表 2** にまとめる．

はじめに特徴 (A) を用いて DNS トラヒックデータの収集期間 T_q が DNS クエリの送信周期 T_d よりも短い場合 ($T_q < T_d$) に端末数 $N(T_q)$ を推定する方法について述べる．このとき，ドメイン d の DNS クエリが収集期間 T_q 内に送信される確率 $p_d(T_q)$ は $p_d(T_q) = T_q/T_d$ を満たす．よって，収集期間 T_q 内に送信されたドメイン d の DNS クエリ数 $Q_d(T_q)$ は $Q_d(T_q) = N(T_q) \cdot p_d(T_q)$ を満たす．よって端末数 $N(T_q)$ は以下の式を満たす．

$$N(T_q) = \frac{Q_d(T_q)}{p_d(T_q)} = \frac{T_d}{T_q} Q_d(T_q) \quad (1)$$

図 4 の例の場合，収集期間 T_q 内に観測されたクエリ数は $N_d(T_q) = 3$(クエリ $q_{1,0}$，$q_{1,1}$，$q_{1,2}$)．さらに収集期間 T_q を $T_q = T_d/2$ とすると，端末数 $N(T_q)$ は $N(T_q) = 3/(1/2) = 6$ と推定される．

次に特徴 (B) を考慮した場合の推定方法について述べる．特徴 (B) によると，DNS クエリは必ずしも送信周期 T_d ごとに送信されず，T_d よりも短い送信間隔で送信されることがある．そのため，収集期間 T_q の長さによっては，同一の端末から送信されたクエリが複数観測されることがあり得る (図 4 におけるクエリ $q_{1,1}$，$q_{1,2}$)．そこで，端末数を正確に推定するためには，同一の端末から送信されたクエリ分を観測されたクエリ数 $Q_d(T_q)$ から除く必要がある．

表 2: 記法
Table 2 Notation

T_d	ドメイン d の送信周期
T_q	DNS トラヒックデータの収集期間
$Q_d(T_q)$	収集期間 T_q の間に送信されたドメイン d の DNS クエリの数
$N_d^1(T_q)$	収集期間 T_q に，ドメイン d の DNS クエリを 1 回だけ送信する端末の数
$N_d^2(T_q)$	収集期間 T_q に，ドメイン d の DNS クエリを複数送信する端末の数
$\mu_d^2(T_q)$	ドメイン d の DNS クエリを収集期間 T_q 内に複数送信する場合の送信回数の期待値
$p_d(T_q)$	ドメイン d の DNS クエリを収集期間 T_q 内に送信する確率
$p_d^2(T_q)$	ドメイン d の DNS クエリを収集期間 T_q 内に複数送信する確率
$p_d^0(T_d)$	ドメイン d の DNS クエリを送信周期 T_d 内に送信しない確率

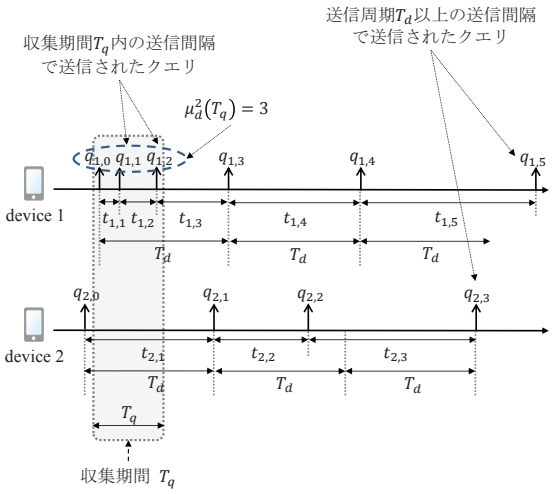


図 4: 端末数推定における状況例
Fig. 4 An example of the estimation situation

収集期間 T_q にクエリを複数送信する端末の数を $N_d^2(T_q)$ とすると，$N_d^2(T_q)$ は，クエリを複数送信する確率 $p_d^2(T_q)$ を用いて $N_d^2(T_q) = p_d^2(T_q) \cdot p_d(T_q) \cdot N(T_q)$ と表される．今，収集期間 T_q 内に同一の端末からクエリが複数送信される場合に同一端末から送信されるクエリの数の期待値を $\mu_d^2(T_q)$ とすると，同一の端末から送信されたクエリの数は $(\mu_d^2(T_q)-1)\cdot N_d^2(T_q) = (\mu_d^2(T_q)-1)\cdot p_d^2(T_q)\cdot p_d(T_q)\cdot N(T_q)$ と表される．よって，式 (1) は以下のように変形される．

$$N(T_q) = p_d(T_q) \cdot(Q_d(T_q) - (\mu_d^2(T_q) - 1) \cdot p_d^2(T_q) \cdot p_d(T_q) \cdot N(T_q))$$
$$N(T_q) = \frac{Q_d(T_q)}{(T_q/T_d)(1 + (\mu_d^2(T_q) - 1) \cdot p_d^2(T_q))} \quad (2)$$

図 4 の例の場合，端末 1 から送信されたクエリの内，送信間隔が T_q より短いものは 2 つ ($q_{1,1}$，$q_{1,2}$) である．よって，端末 1 のクエリの送信パターンから送信クエリが複数送信される確率を算出すると，$p_d^2(T_q) = 2/6 = 1/3$ となる．また，クエリが複数送信される場合のクエリ数の期待値は $\mu_d^2(T_q) = 3$ となる．以上から，端末数 $N(T_q)$ は $N(T_q) = 3/((1/2)(1 + (3 - 1) \cdot 1/3)) = 18/5 = 3.6$ と推定される．

最後に特徴 (C) を考慮した場合の推定方法について述べる．特徴 (C) によると，DNS クエリは送信周期 T_d よりも長い送信間隔で送信されることがある．式 (1) および式 (1) における DNS クエリを収集期間 T_q 内に送信する確率 $p_d(T_q)$ は，クエリが必ず送信周期 T_d 内に送信される場合において $p_d(T_q) = T_q/T_d$ と表される．しかし，特徴 (C) によりクエリは必ずしも送信周期 T_d 内に送信されないため，この $p_d(T_d)$ において特徴 (C) の事象を考慮する必要がある．ある端末が送信したクエリが収集期間 T_q 内に観測されるのは，端末が送信周期 T_d 内にクエリを送信し，かつ送信した時間と収集期間 T_q が一致する場合である．よって，特徴 (C) がみられる確率を $p_d^0 T_d$ とすると，クエリが収集期間 T_q 内に送信される確率 $p_d(T_q)$ は $p_d(T_q) = (T_q/T_d) \cdot (1 - p_d^0(T_d))$ となる．よって，式 (2) から端末数 $N(T_q)$ は以下の式を満たす．

$$N(T_q) = \frac{Q_d(T_q)}{p_d(T_q)(1 + (\mu_d^2(T_q) - 1) \cdot p_d^2(T_q))} = \frac{Q_d(T_q)}{(T_q/T_d) \cdot (1 - p_d^0(T_d))(1 + (\mu_d^2(T_q) - 1) \cdot p_d^2(T_q))} \quad (3)$$

図 4 の例の場合，端末 1 から送信されたクエリの内，送信間隔が T_d より長いものは 1 つ ($q_{1,5}$) である．よって，端末 1 のクエリの送信パターンから送信クエリが送信周期 T_d 内に送信されない確率を算出すると，$p_d^0(T_q) = 1/6$ となる．以上から，端末数 $N(T_q)$ は $N(T_q) = 3/((1/2) \cdot (1 - 1/6) \cdot (1 + (3 - 1) \cdot 1/3)) = 108/25 = 4.32$ と推定される．

表 3: 推定式における各変数の値
Table 3 Parameters for the estimate equation

Device 1				
T_d	T_q	$p_d^2(T_q)$	$p_d^0(T_d)$	$\mu_d^2(T_q)$
86400	86400	0.357	0.429	2.00
	43200	0.262		2.00
	21600	0.143		2.00
	10800	0.095		2.00
	5400	0.071		2.00
Device 2				
T_d	T_q	$p_d^2(T_q)$	$p_d^0(T_d)$	$\mu_d^2(T_q)$
86400	86400	0.783	0.130	2.86
	43200	0.626		2.34
	21600	0.539		2.05
	10800	0.513		2.05
	5400	0.348		2.00

5. 実験

提案手法の有効性を検証すべく，社内のイントラネットワークにおける DNS トラヒックを用いて実際に Android OS の端末数を推定した．また DNS トラヒックと同時にイントラネットワーク内の DHCP トラヒックを収集し，DHCP による fingerprinting を行い，提案方式による推定結果との比較を行った．検証に利用した推定式 (3) の各変数は 4.2 節での端末 1，2 それぞれにおける解析結果から導出した．各変数の値は**表 3** にまとめる．

図 5 に表 3 の各変数を用いて端末数を推定した結果を示す．図 5 の推定結果は 60 日分の DNS トラヒックデータから，各々の収集期間 T_q において観測された `android.clients.google.com` ドメインへの DNS クエリの数の平均を利用して算出した．図 5 の誤差範囲は，それぞれの収集期間 T_q における DNS クエリ数のばらつきによる端末数の推定値の標準誤差の範囲を示す．また図 5 の点線は，DHCP による fingerprinting 手法を用いて端末数を推定した結果の 60 日間の平均値を示す．図 5 より，端末 2 の DNS クエリの解析結果から導かれた各変数を推定に用いた場合の推定結果は，DHCP による OS fingerprinting の結果と近い値をとっている．これより，社内イントラネットワークに存在する Android OS は，端末 2 が示した DNS クエリにおける特徴に近い特性を有していると推測される．端末 1 については，社内イントラネットワークに存在した Android OS よりも，送信間隔が周期より長くなる傾向が強く ($p_d^0(T_q)$ の値が大きい)，また周期内に送信される DNS クエリの数も小さくなる傾向がある ($\mu_d^2(T_q)$，$p_d^2(T_q)$ の値が小さい) ため，推定された端末数が DHCP OS fingerprinting の結果よりも大きくなったと推測される．このことから，推定につかう変数の選定はネットワーク内に存在する端末の特性から大きく乖離しないように行

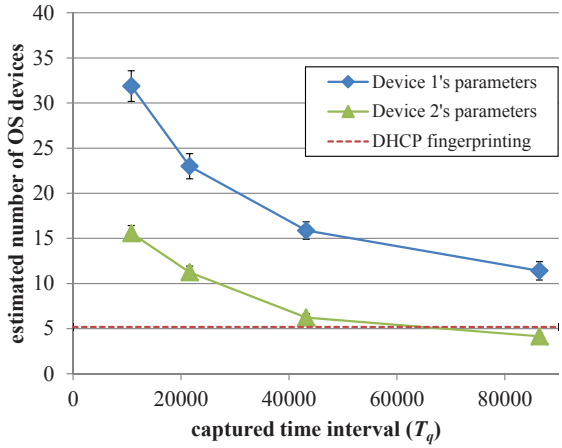


図 5: 端末数の推定結果
Fig. 5 Estimation results

う必要があることがいえる．また，同時に DNS トラヒックによる OS fingerprinting では，OS の判別のみならず端末の判別にも利用できる可能性があることを示している．

6. おわりに

本稿では DNS トラヒックを用いた OS fingerprinting 手法を提案した．提案手法はネットワークから収集した DNS トラヒックから，その収集期間内に存在した OS の端末数を推定するものである．

本稿では，まず提案手法の実現性を検証するために，OS から送信される DNS トラヒックを解析し特徴を抽出した．解析により，各 OS は他の OS が DNS クエリを送信しないような OS 固有のドメイン名を持ち，その固有ドメインの DNS クエリの送信間隔に周期性を持つ，といった特徴を有することが明かとなった．また同時に，送信周期より外れて DNS クエリを送信する場合があることも明らかとなった．

本稿では，次に上記解析によって抽出された DNS クエリにおける OS ごとの特徴を利用して端末数を推定する方法を提案した．提案手法は，DNS クエリの送信間隔における周期性に加えて，DNS クエリが周期よりも短い時間で送信される場合，周期よりも長い時間で送信される場合も考慮して端末数を推定する．提案手法では，これらの周期性から外れる事象が起こる確率，周期内に送信される DNS クエリの数を用いて，収集した DNS トラヒックの中に存在する固有のドメインの DNS クエリの数から端末数の推定を行う．

最後に，本稿では提案手法による推定の有効性を検証するために，社内イントラネットワーク内の DNS トラヒックを用いて Android OS 端末の数の推定を行った結果を示した．検証では，DHCP による fingerprinting を行った結果と照合した．検証結果によると，端末 2 の解析結果から抽出したパラメータ (周期性から外れる確率，周期内に送信される DNS クエリ数) を用いた推定において，提案手法と DHCP による fingerprinting とで推定された端末数が近い値となった．

今後の課題として，推定を正確に行うためのパラメータを抽出する方法の検討，Android OS 以外の OS での提案方法の検証があげられる．

謝辞 本研究は，独立行政法人情報通信研究機構の委託研究「ドライブ・バイ・ダウンロード攻撃対策フレームワークに関する研究開発」の一環として行われた．ここに深謝する．また，本研究にあたり元 (株)KDDI 研究所の山下 公章氏に多大なるご貢献をいただいたことを感謝する．

参考文献

[1] Ericsson: “Traffic and Market Report”, Available at http://www.ericsson.com/res/docs/2012/traffic_and_market_report_june_2012.pdf, Jun., 2012.
[2] M. Zalewski, “p0f v3”, Available at http://lcamtuf.coredump.cx/p0f3/.
[3] E. Kollmann, “Chatter on the Wire: A look at DHCP traffic”, Available at http://myweb.cableone.net/xnih/download/Chatter-DHCP.pdf, 2007.
[4] S. Schulz, A. Sadeghi, M. Zhdanova, H. A. Mustafa, W. Xu and V. Varadharajan, “Tetherway: A Framework for Tethering Camouflage”, Proc. ACM Wireless Network Security (WISEC 2012), pp. 149–160, 2012.
[5] S. Shah, “HTTP Fingerprinting and Advanced Assessment Techniques”, Blackhat 2003 USA, Available at http://www.blackhat.com/presentations/bh-usa-03/bh-us-03-shah/bhus-03-shah.ppt, 2003.
[6] G. F. Lyon., “Remote OS Detection via TCP/IP stack fingerprinting”, Available at http://nmap.org/book/osdetect.html, 2011.
[7] C. Popi, O. Festor, “A Scheme for Dynamic Monitoring and Logging of Topology Information in Wireless Mesh Networks”, Proc. IEEE Network Operations and Management Symposium (NOMS 2008), pp. 759–762, 2008.
[8] D. Tuncer, M. Charalambides, G. Pavlou and N. Wang, “DACoRM: A Coordinated, Decentralized and Adaptive Network Resource Management Scheme”, Proc. IEEE Network Operations and Management Symposium (NOMS 2011), pp. 417–425, 2011.
[9] R. G. Clegg, S. Clayman, G. Pavlou, L. Mamatas and A. Galis, “On the selection of management/monitoring nodes in highly dynamic networks”, IEEE Trans. on Computers, vol. 99, pp. 1–15, Mar., 2012.
[10] F. Gagnon and B. Esfandiari, “A Hybrid Approach to Operating System Discovery Based on Diagnosis Theory”, Proc. IEEE Network Operations and Management Symposium (NOMS 2012), pp. 860–865, 2012.
[11] K. Xui, Z. Zhang and S. Bhattacharyya, “Profiling Internet Backbone Traffic: Behavior Models and Applications”, Proc. ACM SIGCOMM 2005, pp. 169–180, 2005.
[12] F. Zhang, W. He, X. Liu and P. G. Bridges, “Inferring Users’ Online Activities Through Traffic Analysis”, Proc. ACM conference on Wireless Network Security (WISEC 2011), pp. 59–70, 2011.
[13] R. Beverly, “A Robust Classifier of Passive TCP/IP Fingerprinting”, Proc. Workshop Passive and Active Network Measurement (PAM2004), pp. 158–167, 2004.