目 次

第 1 章　序論

近年ネットワークの高度化とコンピュータの一般家庭への普及に伴い，インターネットの利用者数は年々増加し平成 28 年における我が国の利用者数は 1 億人を超えている [1]．しかしインターネットにはさまざまな脅威が潜んでおり，情報漏えいやデータの改ざんやマルウェアの被害に対策を講じておくことが望まれている．

マルウェアに感染したコンピュータは，利用者の意図しない不正な通信を行うことが知られている．そこで，利用者が簡単にパケットの送受信状況を観察することができれば，意図しない通信を感知し早めの対処をすることができる．また，過去の通信状況を遡って知ることができれば，マルウェア感染当時の状況を知ることへと繋がり，被害状況の把握もできる．

トラフィックを可視化する研究は盛んに行われている．例えば，独立行政法人情報通信研究機構 (NICT) が推進する nicter (Network Incient analysis Center for Tactical Emergency Response) と呼ばれる研究プロジェクトでは，世界中から日本に飛来する攻撃を可視化するシステムを開発している．ただし，nicter は大規模なシステムに対して設計されており，個人のコンピュータの監視には向いていない．個人がパケットを観察するツールには TCPEye や Wireshark [2] が存在するが，これらを用いて通信状況を把握するにはプロトコルや IP アドレス等の専門的な知識を要する場合が多く，万人がトラフィックの状況を直感的に理解するのは難しい．

そこで本研究では，リアルタイムの通信状況から不正通信を検知し，その結果をパケット情報とともに可視化することで直感的に理解しやすいシステムの構築を行う．

以下，2 章では通信の仕組みと不正通信の検知方法について述べる．3 章ではトラフィックの可視化に関する研究について述べる．4 章では本研究で提案するシステムの構成と，システムの評価について述べる．そして最後に 5 章では本研究のまとめと今後の課題について述べる．

第2章　通信と検知手法

マルウェアに感染したコンピュータの多くは，インターネットを介した通信を不正に行う．不正通信を観測するには，インターネット通信の仕組みを知ることが不可欠である．そこで，本章ではインターネット通信の仕組みと，通信の分析に必要な情報の取得方法について述べる．

2.1　プロトコルの役割

コンピュータネットワークには TCP や IP を始め複数のプロトコルが必要であり，TCP/IP はこれらを総称したプログラム群のことを指す．かつて現在では， TCP/IP はコンピュータネットワークにおいて最も利用されているプロトコル群である．TCP/IP のプロトコルは，役割ごとに階層化することができる．この階層モデルを図 2.1 に示す．

アプリケーション層
DNS, URI, HTML, HTTP, TLS/SSL, SMTP, POP, IMAP, MIME, TELNET, SSH, FTP, SNMP, MIB, SIP, RTP, LDAP
トランスポート層
TCP, UDP, UDP-Lite, SCTP, DCCP
インターネット層
ARP, IP, ICMP
ネットワークインタフェース層
（ハードウェア）

図 2.1　TCP/IP の階層モデル

TCP/IP による通信では，送信側はアプリケーション層で作成したメッセージを分割し，下位層に順番に渡す．このとき，各層では上位層から渡されたデータにヘッダを付加する．こうして送信データにヘッダを重ねていったものがパケットとなる．最終的に送信するパケットの構造の一例を図 2.2 に示す．

受信側は送信側からパケットを受け取り，下位層から上位層へとパケットを渡す．このとき

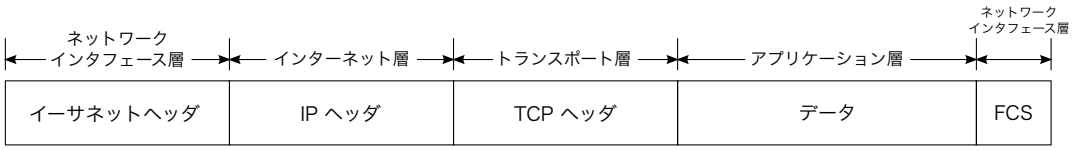

図 2.2　パケットの構造の一例

各層でパケットのヘッダを解析することで，どのようなプロトコルが使われているかがわかり，適当な上位層へデータを渡すことができる．

以下で，TCP/IP 階層モデルの各層の詳細を述べる．

ネットワークインタフェース層

ネットワークインタフェース層は，データリンクを利用して通信を行うためのインタフェースとなる階層である．ネットワークインタフェース層で扱うパケットのことを特にフレームという．データリンクの中で現在最も普及しているのが **Ethernet** であり，Ethernet で伝達されるフレームはすべて Ethernet フレームとして運ばれる．Ethernet の中でもいくつか異なる仕様が存在するが，そのフォーマットは 2 種類に大別される．Ethernet のフレームフォーマットを図 2.3 に示す．

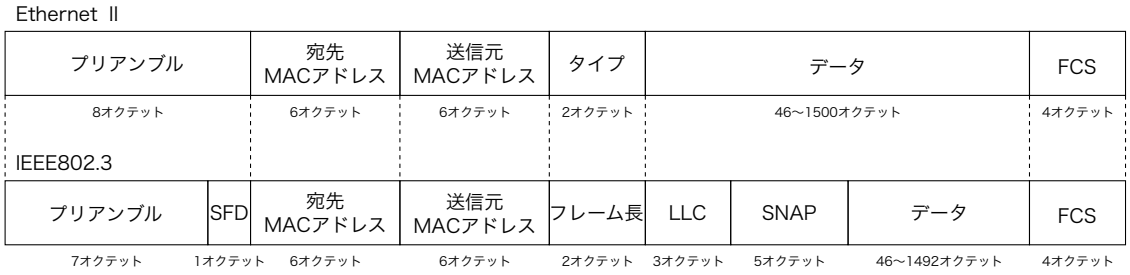

図 2.3　Ethernet フレームフォーマット

図 2.3 からわかる通り，送信元 MAC アドレスに続く 2 オクテットの部分が異なっている．IEEE802.3 Ethernet フレームフォーマットの方はフレーム長であり，後に続くデータ部分の長さを示しているため最大値は 1500 となる．そこで上の Ethernet フレームフォーマットのタイプ番号には 1500 よりも大きい値が用いられており，この部分を調べることでフレームフォーマットを識別することができる．また，タイプ番号は Ethernet の上位層のプロトコルを示しており，こちらのフレームフォーマットの方が現在主流のため，この部分の数値を調べることで，以降のデータ部内のヘッダを識別することもできる．

インターネット層

インターネット層は，宛先までデータを届ける役割を持つ階層である．インターネット層では，**IP** (Internet Protocol) が支配的に用いられいる．IP はパケットが宛先に正しく届いたかを保証せず，このように保証のないパケットのことを特にデータグラムという．現在用いられている IP には バージョン 4 (IPv4) と バージョン 6 (IPv6) があるが，ここでは特に主要に用いられている IPv4 について述べる．

図 2.4　IPv4 ヘッダフォーマット

図 2.4 は IPv4 のヘッダフォーマットを示した図である．この図に示す通り，IP ヘッダ内に送信元・宛先 IP アドレスが示されており，このおかげでインターネットを介してパケットをやり取りすることができる．そのため，インターネットに接続されるすべてのホストやルータは，必ず IP の機能を備えていなければならない．

トランスポート層

トランスポート層の最も重要な役割は，アプリケーション間の通信を実現することである．これは，コンピュータ内部では複数のアプリケーションが同時に動作しており，どのプログラム同士が通信しているのかを識別する必要があるためである．これには，**ポート番号**という識別子が使われる．

トランスポート層では代表的な プロトコルが 2 つ存在する．

TCP (Transmission Control Protocol) は，コネクション型で信頼性を持つプロトコルである．もし通信経路の途中でデータの損失や入れ替わりが発生しても，TCP によって解決することができる．ただし，信頼性を高める代わりに制御のパケットをやり取りする必要があり，一定間隔で決められた量のデータを転送するような通信にはあまり向いていない．

UDP (User Datagram Protocol) は，コネクションレス型で信頼性のないプロトコルである．TCP とは違い，送信したデータが宛先に届いているかの確認をしないため，データの損失や入れ替わりの確認はアプリケーション側で行う必要がある．しかし，確認しない代わりに効率よく通信を行うことができ，パケット数が少ない通信や，音声通信等の一定間隔の通信に向いたプロトコルと言える．

アプリケーション層

アプリケーション層では，アプリケーション内で行われるような処理を行う．アプリケーション層で用いられるプロトコルは，ブラウザとサーバー間の通信に使われる HTTP や電子メールの送受信に用いられる SMTP 等，特定のアプリケーションに特化したプロトコルが多い．

2.2　不正通信の検知手法

不正通信を検知する手法は，大きく**ブラックリスト方式（シグネチャ型検知）**，**アノマリ型検知**，**ホワイトリスト方式**の 3 つに別けることができる [3]．

ブラックリスト方式

ブラックリスト方式（シグネチャ型検知）は，信頼できる専門家によって既知の攻撃パターン（シグネチャ）をリスト化し，通信時にリストとのマッチングで一致した場合に警告を出す（あるいは通信を遮断する）方式である．過去の攻撃に合致したパターンのみを検知するため，正常な状態を異常と見なすこと（フォールスポジティブ）が少ないという特徴がある．しかし，未知の攻撃に対してはパターンを適用することができず，異常を見逃してしまうこと（フォールスネガティブ）が起こってしまうことがある．

ホワイトリスト方式

ホワイトリスト方式は，信頼できる通信のみをリスト化し，リストと一致しない場合に警告や遮断を行う方式である．ブラックリスト方式とは対照的に，フォールスネガティブは少なく，攻撃被害への耐性は高い．一方で，フォールスネガティブに伴う通信の遮断によってサービス停止を招いてしまうリスクも高まることから，安全性と運用上のコストはトレードオフの関係にあると言える．

アノマリ型検知

アノマリ型検知は，通常の状態のプロファイルを設定しておき，これに違反した場合に異常と見なす検知の方法である．シグネチャ型と比較して異常の定義が広いため，フォールスポジティブやフォールスネガティブが比較的多くなってしまうが，プロファイルのしきい値を調整することで誤検知を減らすことができる．また，この方法では過去に観測されていない未知の攻撃に対しても対応できるというメリットがある．

どの方法にもメリットとデメリットがそれぞれ存在するため，互いのデメリットを補うために複数の方法を組み合わせて実装することが多い．

第3章　トラフィックの分析と可視化

本章では，ユーザのコンピュータ上でトラフィックを分析する手法と，現在研究開発が進んでいるトラフィックの可視化に関する研究について述べる．

3.1　ツールを用いたトラフィックの分析手法

ユーザがコンピュータ上のトラフィックを分析する際には，専ら既存のパケットキャプチャツールが利用される．数あるパケットキャプチャツールの中から，ここでは一例として Wireshark と TCPEye について説明する．

Wireshark

Wireshark は，ネットワーク分析において最も普及しているツールである．Wireshark は豊富や機能を持っている [2]．その一部を以下に示す．

- プロトコルの何百もの詳細な検査が，常に追加される
- リアルタイムパケットキャプチャとオフライン分析
- マルチプラットフォーム：Windows，Linux，macOS，Solaris，FreeBSD，NetBSD など
- 豊富な VoIP 分析
- Coloring rules
- XML、PostScript®，CSV，またはプレーンテキストに出力可能

Wireshark を起動し，ネットワークアダプタを指定してキャプチャを開始すると，図 3.1 のようにリアルタイムパケットが表示される．パケットリストは独自の Coloring rules によって色別けされる．パケットリストからパケットを選択すると，さらに詳細なパケット情報が表示される．このように，知りたいパケットを見つけて詳細情報を確認することで，通信状況の分析を行う．

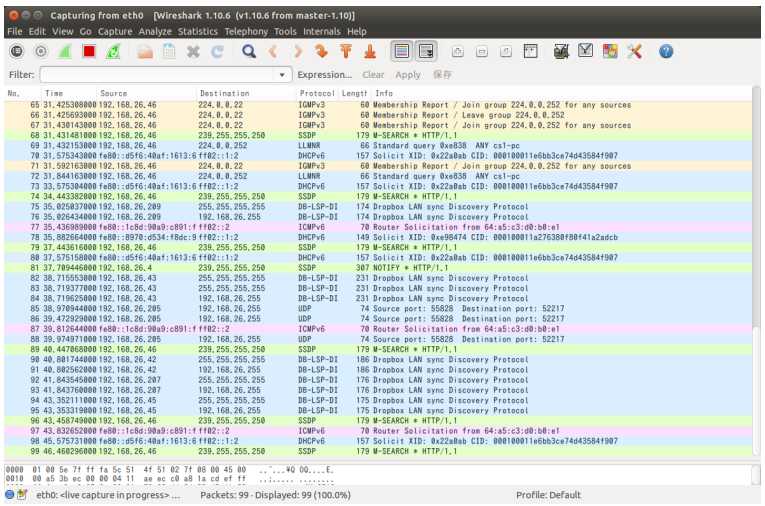

図 3.1　Wireshark を用いたパケット分析

TCPEye

TCPEye は TCP/UDP を対象とした通信モニタリングツールである．図 3.2 のように，リアルタイムで通信を行っているプロセスとその通信状況を一覧表示することでネットワー

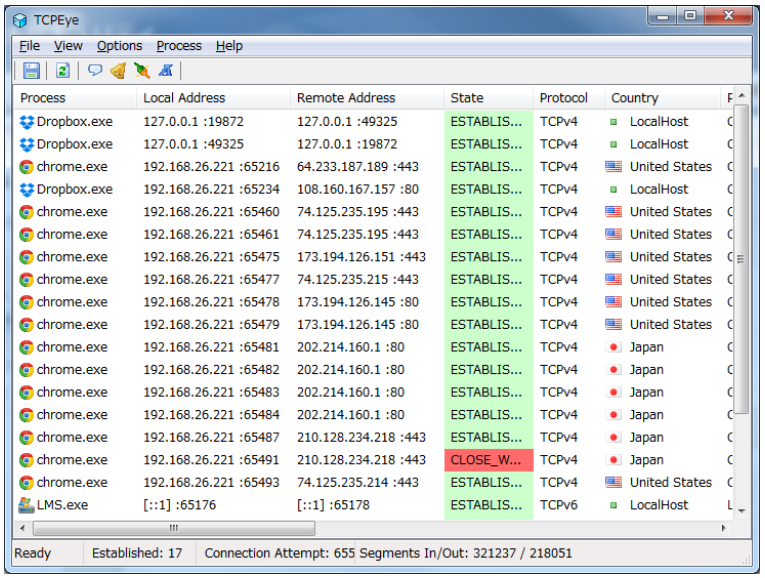

図 3.2　TCPEye を用いたパケット分析

クの状況を行うことができる．IP アドレス，ポート，プロトコルの識別といった一般的なパケットキャプチャツールが持つ機能に加え，IP アドレスから送信元の場所を割り出したり，外部サービスと連携したウィルスチェック等，有用な機能を備えている．

しかし，2 つのツールともに，アドレスやプロトコルなどの専門的な用語が表示されており，

これらの用語の意味を知らない人にとってはどこが異常かを判別するのは難しい．Wireshark や TCPEye に限らず殆どのパケットキャプチャツールについても同様のことが言えるが，万人が一見して不正通信を見分けるのは困難である．この解決として，全ての人が一目で不正通信を見分けるには通信の可視化等を施して，直感的に異常を識別できるような機能が求められる．

3.2　侵入検知システム (IDS)

3.1 節で述べたツールを用いた手法は，異常の有無をユーザが能動的に調べるための手法であった．これに対して，不正通信を自動的に発見する手段として現在最も利用されているのが，**侵入検知システム** (Intrusion Detection System, **IDS**) を用いた方法である．IDS はネットワークへの不正な通信の兆候を検知し，管理者に知らせる機能を持ったソフトウェアもしくはハードウェアのことである．IDS は監視対象によってネットワーク型 IDS (NIDS) とホスト型 IDS (HIDS) に大別され，前者はコンピュータネットワークの通信内容を検査し，後者は対象となるサーバに組み込むことで異常が発生していないかを監視する．NIDS と HIDS それぞれの特徴を比較したものを表 3.1 に示す．

表 3.1　NIDS と HIDS の比較

	NIDS	HIDS
監視対象	ネットワーク	サーバ
設置	**セグメントごと**	コンピュータごと
マルウェアや攻撃者からの検知	**されにくくすることが可能**	比較的されやすい
暗号化通信	検知不可	**復号によって検知可**
コンピュータ内部の異常	検知不可	**検知可**

NIDS は一般的にミラーポートと呼ばれる監視専用のポートを持ち，ミラーポートをネットワークセグメントに接続することで監視を行う．このミラーポートには IP アドレスを割り当てないようにでき，この機能によって外部からの特定や攻撃を受けにくくするメリットがある．しかし，監視した通信内容が暗号化されている場合は復号する術を持たないため，たとえ不正な通信であっても検知することができない．HIDS は各サーバごとに設置する必要があり，ネットワークセグメントごとに設置すればよい NIDS に比べ管理コストがかかる．しかし，サーバ内部を監視するため，暗号化された通信も受信後に復号することで検知が可能である．また，ログの改ざん等のサーバ内部の異常も検知することができる．

しかし，いずれの場合も異常を検知するには事前に IDS を設置し，常に運用しなければならず，コストがかかってしまうという問題がある．そのため，個人がコンピュータを監視するような場合にはあまり適しているとは言えない．

3.3　nicter

独立行政法人情報処理研究機構 (NICT) が進めている研究に **nicter** (Network Incient analysis Center for Tactical Emergency Response) がある．nicter は「インターネット上で時々刻々と発生しているセキュリティインシデントへの迅速な対応」を目的としており，インターネット上で生起する多種多様な事象の収集および分析を実施している．また，nicter で行われている研究の一つがトラフィックの可視化である．

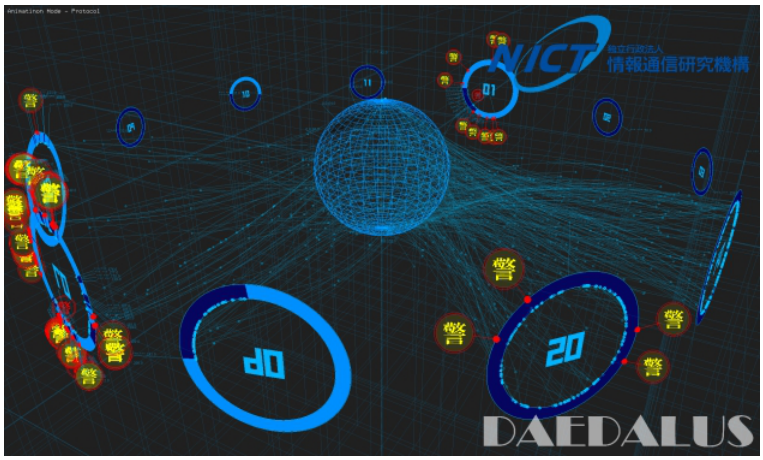

図 3.3　DAEDALUS によるアラート可視化 [5]

図 3.3 は **DAEDALUS** (Direct Alert Environment for Darknet And Livenet Unified Security) と呼ばれる，nicter で構築した大規模なダークネット観測網を活用した対サイバー攻撃アラートシステム [5] である．ダークネットとは到達可能な IP アドレスのうち特定のホストコンピュータが割り当てられていないアドレス空間，すなわち未使用の IP アドレスのことである．DAEDALUS では，ダークネット上を流れるパケットの多くが不正な目的に起因することを利用し，不正パケットを割り出して可視化している．図 3.3 からわかる通り，地球に見立てたオブジェクトに向けてパケットが流れている状況を一目で理解することができる．

もう一つ nicter の研究の一つである **Atlas** について述べる．Atlas は日本に対してリアルタイムに行われている攻撃を可視化するシステムである．図 3.4 を見ると，世界地図上の各国から日本に向けてデータが飛んでくる様子が見て取れる．Atlas はトラフィックを単に

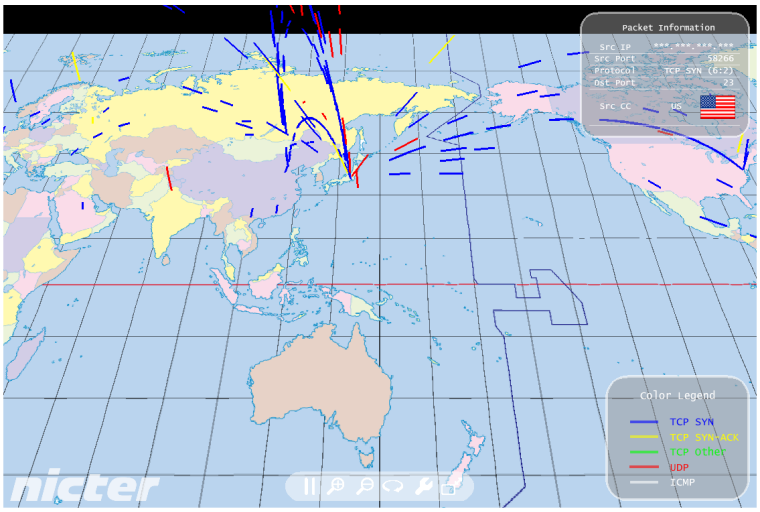

図 3.4　Atlas によるサイバー攻撃の可視化

可視化するだけでなく，高さでポート番号，色でプロトコルを示すことで，わかりやすさの中に多くの情報を内包している．専門知識のない人でも見ただけで状況を把握でき，専門知識がある人ならさらに詳しい分析を行えるようになっている．

上で挙げた nicter の 2 つの研究からわかる通り，トラフィックの可視化は通信状況を把握する上で有用な手法であるといえる．

第4章　提案システムの実装と評価

本研究では次の 3 つの条件を満たすトラフィック可視化システムを提案する．

- 個人の PC に接続されたネットワークを対象とする．
- リアルタイムに流れるパケットを分析し，異常を検知できる．
- 一目で通信（被害）状況がわかる．

4.1　実装環境

本研究で使用した環境について表 4.1 に示す．

表 4.1　実装環境

OS	ubuntu 14.04 LTS
メモリ	32 GB
CPU	Intel® Core™ i7-3770
GPU	GeForce GTX 960/PCIe/SSE2
Web ブラウザ	Google Chrome version 53.0.2785.143

4.2　システム詳細

本研究で提案するシステムは，次の 3 つの機能により構成される．

1. パケット情報取得
2. 異常パケット判定
3. トラフィック可視化

パケット情報取得については 4.2.1 節で，異常パケット判定については 4.2.2 節で，トラフィック可視化については 4.2.3 節でそれぞれ詳細を述べる．

4.2.1　パケット情報取得

送受信されるパケットから情報を取得する．求めるパケット情報は **取得日時，送信元・宛先 IP アドレス，プロトコル，送信元・宛先ポート番号**（TCP/UDP のみ），**パケット長**（Ethernet ヘッダから FCS までの大きさ）とする．なお，本研究で対象とするパケットは，IPv4 パケットに限定する．

パケットを取得する際にはパケット解析 API である **pcap** を利用した．pcap にはパケットキャプチャのための関数があり，提案システムに対して関数を用いた処理の流れを図 4.1 に示す．

pcap_findalldevs 関数はコンピュータ上のデバイスをリストアップし，配列に格納する関数である．取得したリストからデバイスを 1 つ選び，デバイスを開くための関数である pcap_open_live 関数に渡す．pcap_open_live 関数では，プロミスキャスモード（パケットを無差別に拾う状態）にするかどうかを選択することができる．提案システムではすべてのパケットを対象にキャプチャを行いたいため，プロミスキャスモードで動作させる．デバイスを開いた後，pcap_loop 関数で実際にキャプチャを行う．

パケットのキャプチャが成功したら，パケットデータを Ethernet ヘッダ構造体にキャストし，Ethernet ヘッダのタイプ番号を調べる．タイプ番号が IPv4 を示していれば続行し，それ以外の場合は再び pcap_loop 関数に戻る．IPv4 であると判明したパケットについて，節の冒頭で示したパケット情報を取得する．

4.2.2　異常パケット判定

不正通信の検知方法については 2.2 節で述べたが，本研究では**ホワイトリスト**を用いた異常パケット判定を採用する．

本研究では過去 3 日間に行った通信を正常な通信と仮定し，通信時のパケット取得日時と相手の IP アドレスをデータベースに記録する．4.2.1 節で示したパケット情報をリアルタイムで取得するとき，リアルタイムパケットの IP アドレスとホワイトリストを照合し，一致する IP アドレスがあれば正常，なければ異常とする．この方法は 3 章で述べた IDS と比較して検知精度は劣るものの，軽量に動作することや構成コストが小さいことから，個人の PC におけるリアルタイムでの異常パケット判定に適している．

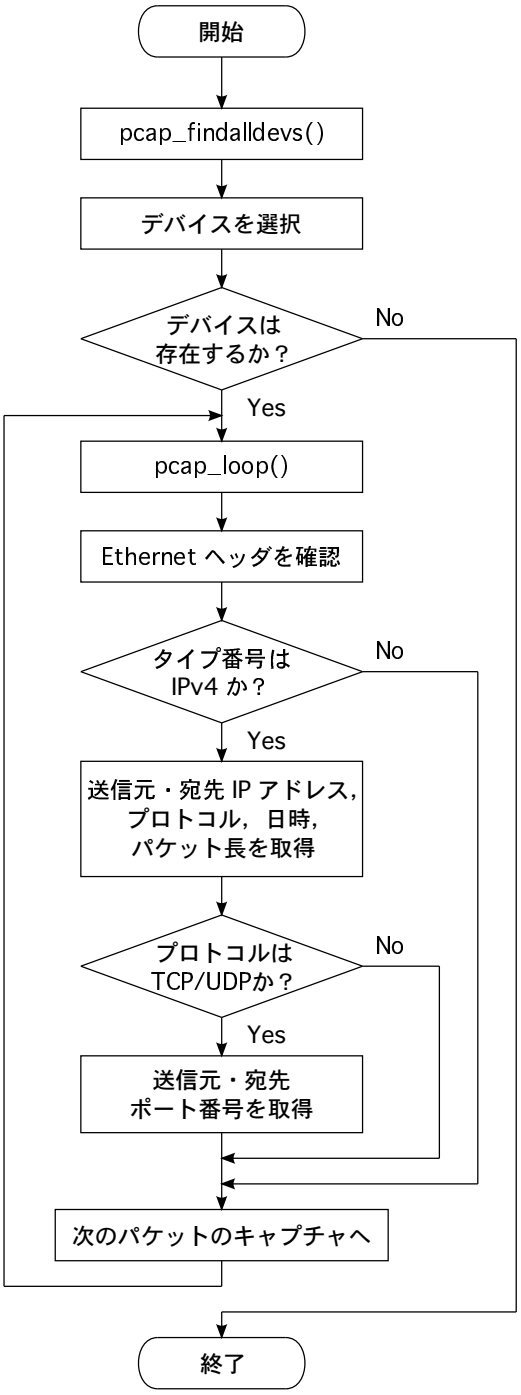

図 4.1　pcap を利用したパケット情報の処理

4.2.3　トラフィック可視化

4.2.1 節で記録したパケット情報をウェブブラウザで読み込み，可視化処理を行う．本節では可視化に用いる技術と可視化方法について述べる．

canvas 要素

ウェブ上での動画表現には従来より Flash や Java Applet といったプラグインによる方法が採られてきたが， HTML5 から，**canvas 要素**を用いた表現を行うことができるようになった．canvas 要素は HTML の要素の一つであり，最新版のウェブブラウザでは標準でサポートしている．canvas 要素の編集は JavaScript によって行う．また，HTML の他の要素と連携した処理を実装したり，PHP 等の他のウェブ系のスクリプト言語と組み合わせることもでき，幅広い表現を実現することができる．

WebGL

トラフィックの可視化には 3D グラフィクス API である WebGL を用いる．WebGL はウェブブラウザ上で 3DCG を表示させるための標準仕様であり，現在の主要ブラウザの最新版なら利用することができる．同じく 3DCG API である OpenGL を基としており，現在盛んに利用されている．

WebGL は HTML5 の canvas 要素と JavaScript を用いて表示させるため，軽快に動作し，Web 系の他のスクリプト言語とも相性が良い．そのため，ローカルのファイルやデータベースとの連携も比較的容易に行うことができ，コンピュータ上に蓄積させたデータの可視化に向いた言語と言える．

本システムではプロセス間通信によって 4.2.1 節で取得したパケット情報を JSON 形式で受け取る．読み込んだファイルに対して 4.2.2 節の判定の後，パケット情報を WebGL を用いて可視化する．

可視化グラフィック

提案システム可視化した情報の見た目のグラフィックついて説明する．canvas 要素を用いた表示領域上に，座標（緯度・経度）を設定した半透明の地球儀オブジェクトを表示する．地球儀オブジェクトの中心には自分の PC に見立てたオブジェクトを表示する．パケットの IP アドレスから座標を割り出し，自分の PC と通信相手の場所を直線で結び，その上をパ

ケットに見立てた球体オブジェクトを流すことでトラフィックを表現する．球体オブジェクトの色は，異常パケットが見つかった箇所はピンク，それ以外は緑で表示する．

座標の特定には GeoIP を用いる．GeoIP は MaxMind 社が提供する，IP アドレスから地理情報を得る仕組みである．提案システムでは，4.2.1 節で示したパケットキャプチャの際， GeoIP 関数にパケット情報の送信元 IP アドレスおよび宛先 IP アドレスを与えることで地理情報（緯度・経度，国コード）を取得する．可視化時に，得られた地理情報を地球儀の座標と対応させる．

また 3.1 節で示したツールと同様，タブを切り替えることでパケット情報を文字ベースでも表示する．

4.3　実行結果

提案システムを用いて，リアルタイムパケットの可視化を行う．結果を図 4.2 に示す．

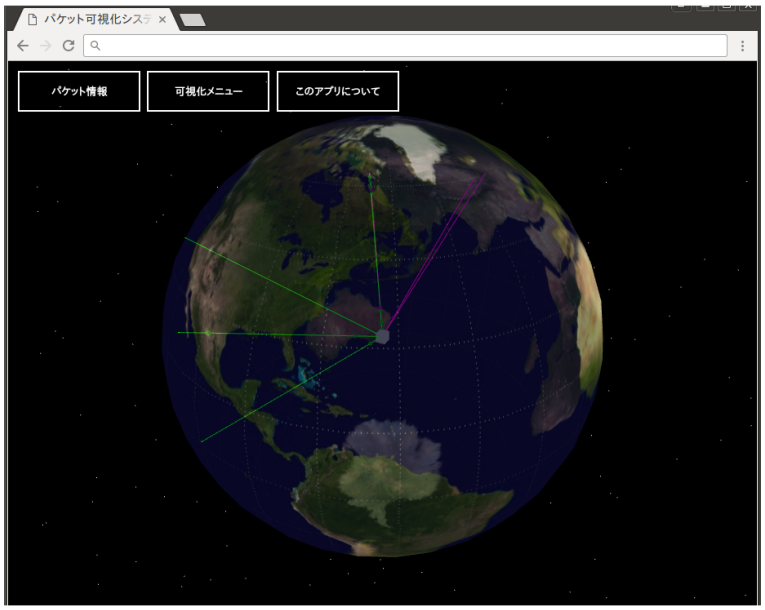

図 4.2　実行結果

図 4.2 は自身の PC が 5 つの送受信先と通信を行っている状況を示す．4 つの通信はホワイトリストに含まれる通信であるが，1 つの通信は異常トラフィックと判断されピンクで表示されている．また左上のタブは図 4.3 のような文字ベースのパケット情報を表示するために用いる．

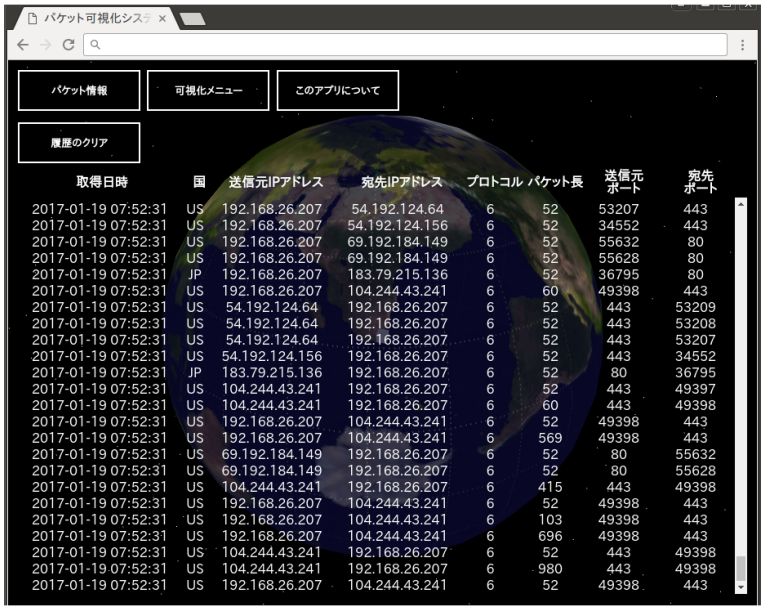

図 4.3　実行結果

4.4　システム評価

提案システムの使いやすさを評価するために，本研究では **System Userbility Scale** (**SUS**) [6] を用いる．SUS は John Brook が 1986 年に開発したアンケートベースの評価尺度である．回答結果をもとにスコアを算出することで，全体的な使いやすさのレベルを数値で把握することができる．SUS は自由に使えるよう自由公開されており，また少ないサンプル数（8 〜 10 人程度）でも比較的信頼できるデータを得ることができることから，様々なシステムの評価に用いられている [7]．

評価方法

SUS で用いる項目を図 4.4 に示す．被験者には各項目について，1 （全くそう思わない）から 5 （とてもそう思う）の 5 段階評価を実施してもらう．その後，項目の番号が奇数のものは回答番号から 1 を引き，偶数のものは 5 から回答番号を引き（各項目 0 〜 4 点），それらを合計した点数に 2.5 を掛けた値（0 〜 100 点）をシステムのスコアとする．スコアが 100 点に近いほど使いやすいシステムと言える．

1. このシステムを頻繁に使いたいと思う
2. このシステムは無駄に複雑であると思った
3. このシステムは簡単に使えると思った
4. このシステムを使えるようになるには，専門家の支援を必要とするかもしれない
5. このシステムにある様々な機能がよくまとまっていると感じた
6. このシステムでは一貫性のないところが多くあると思った
7. ほとんどの人々はこのシステムの使い方をすぐ学べるだろうと思う
8. このシステムはとても扱いづらいと感じた
9. このシステムを使える自信があると感じた
10. このシステムを使い始める前に多くのことを学ぶ必要があった

図 4.4　System Userbility Scale の項目

評価結果

8 人の被験者に提案システムを利用してもらい，SUS による評価を実施した．結果を表 4.2 に示す．SUS が示す平均スコアは 68 点であること [7] が知られているが，提案システムの平均スコアは 70.3125 点となった．このことから，提案システムが使いやすさの点で優れているといえる．

表 4.2　SUS によるユーザビリティ評価

項目		評価								平均	×2.5
1	このシステムを頻繁に使いたいと思う	2	2	3	2	2	3	1	1	2	5.0
2	このシステムは無駄に複雑であると思った	4	4	2	3	4	3	1	4	3.125	7.8125
3	このシステムは簡単に使えると思った	4	4	4	3	4	4	2	3	3.5	8.75
4	このシステムを使えるようになるには，専門家の支援を必要とするかもしれない	3	1	4	3	2	3	0	4	2.5	6.25
5	このシステムにある様々な機能がよくまとまっていると感じた	4	2	4	2	4	3	0	2	2.625	6.5625
6	このシステムでは一貫性のないところが多くあると思った	4	4	1	1	2	3	1	4	2.5	6.25
7	ほとんどの人々はこのシステムの使い方をすぐ学べるだろうと思う	4	4	4	3	4	3	3	3	3.5	8.75
8	このシステムはとても扱いづらいと感じた	4	4	4	2	4	3	2	3	3.25	8.125
9	このシステムを使える自信があると感じた	3	4	3	3	3	4	0	4	3	7.5
10	このシステムを使い始める前に多くのことを学ぶ必要があった	4	3	1	3	2	1	0	3	2.125	5.3125
合計		36	32	30	25	31	30	10	31	28.125	-
SUS スコア		90	80	75	62.5	77.5	75	25	77.5	-	70.3125

項目別では「このシステムは簡単に使えると思った」「ほとんどの人々はこのシステムの使い方をすぐ学べるだろうと思う」の項目について，特に評価の高い結果となった．このことから，システムの使いやすさという点で優位性を確認することができた．

一方,「このシステムを使えるようになるには，専門家の支援を必要とするかもしれない」の項目について，評価値は 6.25 点となり，他の項目と比較してやや低い数値にとどまった．このような結果となった要因として，異常パケットの有無を発見するのは簡単だが，線の色の意味や異常パケットの情報を可視化結果から簡単にアクセスできないことが理由として考えられる．この項目を改善するために，ユーザに配慮した機能をさらに付加する必要がある．

第5章　結論

本研究では，pcap を用いてリアルタイムパケットを取得し，事前に作成したホワイトリストと照合して異常パケットかどうかを判定し，WebGL によって直感的なグラフィックで表現するシステムの開発を行った．

従来のツールと同様に文字ベースのパケット情報を表示し，さらに可視化情報を有することで直感的理解において優位なシステムとなった．SUS によるユーザビリティ評価では，SUS 全体の平均を上回るスコアを得ることができた．特に，システムの学習コストの低さを表す項目で高い評価となった．一方で，専門家の支援を必要とするかもしれないという項目については線の色の意味や異常パケットの情報を可視化結果から簡単にアクセスできないという点について改善が必要である．

今回用いたホワイトリストによる異常パケット判定は個人の PC において適した手法であるものの，検知精度の問題は依然として残っているため，軽量さを維持しつつもより精度の高い手法を検討する必要がある．また，近年のスマートフォンやタブレット端末の普及とこれらを対象にしたマルウェアの増加を踏まえ，一般的な PC 以外のデバイスに対しても利用できるようなシステム開発も今後の課題である．

謝辞

本研究を進めるにあたり，常日頃より丁寧かつ熱心なご指導を頂きました甲斐 博准教授に深く感謝いたします.

また，本研究に際してご査読いただいた野口 一人教授，木下 浩二講師に感謝の意を表します.

最後に，本研究において多大なご協力を頂きました神戸大学大学院の森井 昌克教授，ならびに日頃より励ましの言葉を頂きました諸先輩方，同研究室の諸兄に厚くお礼申し上げます.

参考文献

［1］ 情報通信白書平成 28 年度版，総務省，
http://www.soumu.go.jp/johotsusintokei/whitepaper/

［2］ Wireshark. https://www.wireshark.org/

［3］ Sandro Etalle, Clifford Gregory, Damiano Bolzoni, Emmanuele Zambon: Self-configuring deep protocol network whitelisting, Security Matters Whitepapers, pp.1-27 (2014).

［4］ 中尾康二, 松本文子, 井上大介, 馬場俊輔, 鈴木和也, 衛藤将史, 吉岡克成, 力武健次, 堀良彰: インシデント分析センタ nicter の可視化技術, 電子情報通信学会技術研究報告. ISEC, 情報セキュリティ, Vol.106, Number 176, pp.83-89 (2006).

［5］ 鈴木未央, 井上大介, 衛藤将史, 宇多仁, 中尾康二: 大規模ダークネット観測に基づくアラートシステムの実装と運用, 電子情報通信学会技術研究報告. ICSS, 情報通信システムセキュリティ, Vol.110, Number475, pp.59-64, (2011).

［6］ Brooke, J.: SUS: A quick and dirty usability scale, Usability Evaluation in Industry, pp.189-194 (1996)

［7］ Tom Tullis, Bill Albert: Measuring the User Experience, Morgan Kaufmann Publishers (2008)

付 録A　プログラム

app.js

```
// Express モジュールを読み込む
var express = require('express'),
http = require('http'),
app = express();
// index.html を表示する
app.use(express.static(__dirname + '/public'));
// 10000 ポートで接続を待つ
var server = http.createServer(app).listen(10000, function() {
console.log(' 接続待機中...');
});
// Socket.IO モジュールを読み込む
var io = require('socket.io'),
io = io.listen(server);

// 接続時の処理
io.sockets.on('connection', function(socket) {
console.log(' 接続開始');

var spawn = require('child_process').spawn,
python = spawn('sudo', ['python', 'capture.py', 'eth0']);

python.stdout.on('data', function(data) {
console.log('python stdout: ' + data.toString());
socket.broadcast.emit('packet', data.toString());
});
python.stderr.on('data', function(data) {
console.log('python stderr: ' + data);
});

socket.on('capture', function() {
console.log('yahoo');
//startCapture(socket);
});
socket.on('disconnect', function() {
console.log(' 接続切断');
//python.kill();
});

});



function startCapture(socket) {
}
```

whitelist.py

```
#! /home/tamura/.virtualenvs/capture/bin/python
# -*- coding: utf-8 -*-
```

```
#import MySQLdb
import mysql.connector
import pcapy
import dpkt
import socket
import netifaces
import threading
import time


"""
指定のデバイスを流れるパケットを取得するスレッド
"""
class CaptureThread(threading.Thread):

    def __init__(self, device):
        super(CaptureThread, self).__init__()
        self.device = device
        self.dev_addrs = netifaces.ifaddresses(device)
        try:
            self.dev_IPv4 = self.dev_addrs[netifaces.AF_INET][0]['addr']
        except:
            return None

    def run(self):
        print "start capture " + self.device

        '''
        デバイスを開く
        pcapy.open_live(
            device,
            snaplen, (パケットあたりの取得可能な最大サイズ)
            promiscious mode, (1 for true)
            timeout (ミリ秒)
            )
        '''
        cap = pcapy.open_live(self.device , 65536 , 1 , 60000)

        # パケット取得開始
        cap.loop(-1, self.parse)


    def parse(self, header, packet):
        # イーサネットパケットにパース
        eth = dpkt.ethernet.Ethernet(packet)

        # IPv4 パケット以外は無視
        if eth.type != dpkt.ethernet.ETH_TYPE_IP:
            return

        # 取得パケットの送信元・宛先 IP アドレス
        src_ip = socket.inet_ntoa(eth.data.src)
        dst_ip = socket.inet_ntoa(eth.data.dst)

        # 通信相手の IP アドレスを求める
        if src_ip == self.dev_IPv4:
            addr = dst_ip
        else:
            addr = src_ip

        self.insert_db(addr)


    # IP アドレスを MySQL に記録
    def insert_db(self, addr):
        connection = mysql.connector.connect(host="localhost", db="white_list", user="root", passwd="root", charset="utf8")
        cursor = connection.cursor(buffered=True)

        sql = "INSERT INTO IPv4(ip, time) VALUES(INET_ATON('"+addr+"'), CAST(NOW() AS DATETIME)) ON DUPLICATE KEY UPDATE time = CAST(NOW() AS DATETIME);
```

```
        try:
            cursor.execute(sql)
        except:
            pass

        connection.commit()

        cursor.close()
        connection.close()
        print addr


def main():
    # devices = pcapy.findalldevs()
    # NIC を取得する。
    # 上の関数のほうがキャプチャ可能な NIC の数は多いが、下のほうが NIC の詳細情報を取得しやすいのでこちらを利用
    interfaces = netifaces.interfaces()
    print interfaces

    for interface in interfaces:
        thread = CaptureThread(interface)
        thread.start()

if __name__ == '__main__':
    main()
```

capture.py

```
# -*- coding: utf-8 -*-

'''
パケットを取得するための Python スクリプト。
node.js プログラムの子プロセスとして実行します。
タイムアウト時以外は無限ループなので、Ctrl+Z で強制終了してください。

実行するには、下のコマンド（管理者権限が必須）
$ sudo python capture.py eth0

bash のローカル環境変数を反映させたい場合は、-E オプション
$ sudo -E python capture.py eth0
'''

# モジュール群 (pip 等を使ってインストールしてください)
import sys
import socket
import netifaces
import pcapy
import dpkt
import json
import datetime
import time
import geoip2.database
import mysql.connector

dev_IPv4 = None

# GeoIP のデータベースを読み込む
reader = geoip2.database.Reader('./GeoIP/GeoLite2-City.mmdb')

# MySQL に記録しているホワイトリストを参照する準備
conn = mysql.connector.connect(host="localhost", db="white_list", user="root", passwd="root", charset="utf8")
cursor = conn.cursor(buffered=True)

def main(argv):
global dev_IPv4
```

```
TIMEOUT_MS = 1000

# デバイス名（ネットワークインタフェース名のこと）
dev = argv[1]
# デバイスの IPv プロトコル番号 4 アドレス
try:
dev_IPv4 = netifaces.ifaddresses(dev)[netifaces.AF_INET][0]['addr']
except:
return

# デバイスを開く
cap = pcapy.open_live(dev, 65536, 0, 5000)

# パケット取得開始
cap.loop(-1, parse_packet)
#while(1):
#(header, packet) = cap.next()
#parse_packet(packet, dev_IPv4)


def parse_packet(header, packet):
global dev_IPv4, reader, cursor

# 現在日時の取得
today = datetime.datetime.today()
now = today.strftime('%Y-%m-%d %H:%M:%S')

# イーサネットパケットにパース
eth = dpkt.ethernet.Ethernet(packet)
# IPv4 パケット以外は無視する
if type(eth.data) != dpkt.ip.IP:
return

ip = eth.data
# パケットの送信元・宛先 IP アドレス
src_addr = socket.inet_ntoa(ip.src)
dst_addr = socket.inet_ntoa(ip.dst)

# 送信パケットか受信パケットかそれ以外か
if src_addr == dev_IPv4:
direction = 0 # 送信パケット

elif dst_addr == dev_IPv4:
direction = 1 # 受信パケット
else:
direction = -1 # その他のパケット（ブロードキャストパケットなど）

# 取得した情報を json 形式にまとめる
json_data = {'datetime': now,
 'address': (src_addr, dst_addr),
 'direction': direction,
 'protocol': ip.p,
 'length': len(str(ip))}

# TCP/UDP ならポート番号も追加
if type(ip.data) == dpkt.tcp.TCP or type(ip.data) == dpkt.udp.UDP:
json_data['port'] = (ip.data.sport, ip.data.dport)

try:
record = reader.city(json_data['address'][1-direction])
json_data['country'] = record.country.iso_code
json_data['latlng'] = (record.location.latitude, record.location.longitude)
except:
json_data['country'] = '??'

# ホワイトリストに IP アドレスか存在するかどうか
if direction == 0:
sql = "SELECT EXISTS(SELECT * FROM IPv4 WHERE ip=INET_ATON('"+dst_addr+"'));"
elif direction == 1:
```

```
sql = "SELECT EXISTS(SELECT * FROM IPv4 WHERE ip=INET_ATON('"+src_addr+"'));"
else:
sql = "SELECT EXISTS(SELECT * FROM IPv4 WHERE ip=INET_ATON('"+src_addr+"') AND ip=INET_ATON('"+dst_addr+"'));"
try:
cursor.execute(sql)
except:
pass
result = cursor.fetchone()[0]
json_data['white'] = result;

# json 形式にエンコード
encode_json_data = json.dumps(json_data)

sys.stdout.write(encode_json_data)
sys.stdout.flush()
time.sleep(0.01)


if __name__ == "__main__":
main(sys.argv)
```

index.html

```
<!DOCTYPE html>
<html lang="ja">
<head>
<meta charset="utf-8">
<title>パケット可視化システム</title>
<link rel="stylesheet" href="style.css">
<script src="js/jquery-3.1.1.min.js"></script>
<script src="js/three.min.js"></script>
<script src="js/OrbitControls.js"></script>
<script src="js/Detector.js"></script>
<script src="/socket.io/socket.io.js"></script>
<script src="main.js"></script>
</head>
<body>

<!-- 可視化上に文字情報を載せるための領域 -->
<div id="textArea">
<div id="buttonArea">
<input type="button" class="button" id="infoTableButton" value="パケット情報"/>
<input type="button" class="button" id="menuButton" value="可視化メニュー"/>
<input type="button" class="button" value="このアプリについて"/>
</div>

<div id="menuArea" style="display: none">
可視化モデル選択<br>
<input type="radio" name="model" value="1" checked="checked"/>モデル 1
<input type="radio" name="model" value="2"/>モデル 2
<input type="radio" name="model" value="3"/>モデル 3<br><br>
色分け方法<br>
<input type="radio" name="color" value="1" checked="checked"/>異常の有無で色分け
<input type="radio" name="color" value="2"/>プロトコルで色分け
<input type="radio" name="color" value="3"/>モデル 3
</div>

<div id="infoTableArea" style="display: none">
<input type="button" class="button" id="initTableButton" value="履歴のクリア"/>
<table id="infoTable">
<thead>
<tr>
<th class="datetime">取得日時</th>
<th class="country">国</th>
<th class="s_addr">送信元 IP アドレス</th>
<th class="d_addr">宛先 IP アドレス</th>
```

```
<th class="protocol">プロトコル</th>
<th class="length">パケット長</th>
<th class="s_port">送信元<br>ポート</th>
<th class="d_port">宛先<br>ポート</th>
</tr>
</thead>
<tbody id="infoTableBody"></tbody>
</table>
</div>
</div>

<!-- 可視化領域 -->
<div id="canvasArea"></div>

</body>
</html>
```

main.js

```
// HTML が読み込まれた後の処理
$(function(){
"use strict";

// "パケット情報テーブル"の高さ設定
$(document).ready(function(){
var height = $(window).height() - 190;
$("#infoTableBody").height(height);
});
$(window).on("resize", function(){
var height = $(window).height() - 190;
$("#infoTableBody").height(height);
});

// "パケット情報"ボタンクリック時
$("#infoTableButton").click(function(){
var $area = $("#infoTableArea");

if( $area.is(":visible") ){
$area.fadeOut("slow");
}else{
$("#menuArea").fadeOut("fast");
$area.fadeIn("slow");
}
});

// "可視化メニュー"ボタンクリック時
$("#menuButton").click(function(){
var $area = $("#menuArea");

if( $area.is(":visible") ){
$area.fadeOut("slow");
}else{
$("#infoTableArea").fadeOut("fast");
$area.fadeIn("slow");
}
});


});

(function() {
"use strict";

// ------------------------------------------- 変数宣言
var renderer;
var scene;
```

```
var camera;
var controls;
var baseTime;

var radius = 20;

var ip = [];

var flow = [];

var socket = null;

var marker = 0;
var MAX_FLOW = 1000;  // 表示可能なパケットフローの上限



// ---------------------------------------------- window イベント
// 読み込み時のイベント
$(function(){
init();
});

// リサイズ時のイベント
$(window).resize(function() {
renderer.setSize(window.innerWidth, window.innerHeight);
camera.aspect = window.innerWidth / window.innerHeight;
camera.updateProjectionMatrix();
});


// ---------------------------------------------- 読み込み時の処理
function init() {
initThree();
}


// 可視化モデルの準備
function initModel() {
// シーン作成
scene = new THREE.Scene();

// ライト作成
var ambient = new THREE.AmbientLight(0xffffff);
scene.add(ambient);

// テクスチャ読み込み
var textureLoader = new THREE.TextureLoader();
var texture = textureLoader.load('earth.jpg');
// 地球儀（裏面）
var earthGeometry = new THREE.SphereGeometry(radius, 36, 18);
var earthBackMaterial = new THREE.MeshStandardMaterial({
color: 0xffffff,
roughness: 1.0,
metalness: 0,
opacity:0.5,
transparent: true,
map: texture,
side: THREE.BackSide });
scene.add( new THREE.Mesh(earthGeometry, earthBackMaterial) );
// 地球儀（表面）
var earthFrontMaterial = new THREE.MeshStandardMaterial({
color: 0xffffff,
roughness: 1.0,
metalness: 0,
opacity:0.5,
transparent: true,
map: texture,
side: THREE.FrontSide });
```

```
scene.add( new THREE.Mesh(earthGeometry, earthFrontMaterial) );
// 背景が地球儀に透けてしまわないための措置
var geometry = new THREE.SphereGeometry(radius+0.1, 36, 18);
var material = new THREE.MeshStandardMaterial({
color: 0x000000,
side: THREE.BackSide });
scene.add( new THREE.Mesh(geometry, material) );

// 宇宙
var spaceGeometry = new THREE.Geometry();
for (var i = 0; i < 2000; i++) {
var phi = Math.random() * Math.PI * 2;
var theta = Math.random() * Math.PI * 2;
spaceGeometry.vertices.push(new THREE.Vector3(
1000 * Math.cos(phi) * Math.cos(theta),
1000 * Math.sin(phi),
1000 * Math.cos(phi) * Math.sin(theta)));
}
var spaceMaterial = new THREE.PointsMaterial({size: 2, color: 0xffffff});
var space = new THREE.Points(spaceGeometry, spaceMaterial);
scene.add(space);


// 自分の PC に見立てた立方体
var geometry = new THREE.CubeGeometry(1, 1, 1);
var material = new THREE.MeshStandardMaterial();
var mesh = new THREE.Mesh(geometry, material);
scene.add(mesh);


return scene;
}

function initThree() {
// canvas 領域の取得
var canvasFrame = document.getElementById("canvasArea");

// レンダラ初期化
renderer = new THREE.WebGLRenderer({ antialias: true});
renderer.setSize(window.innerWidth, window.innerHeight);
renderer.setClearColor(0x000000, 1);
canvasFrame.appendChild(renderer.domElement);

// カメラ作成
camera = new THREE.PerspectiveCamera(60, window.innerWidth / window.innerHeight);
camera.position.set(0, 0, 50);
// カメラコントロール作成
controls = new THREE.OrbitControls(camera);
controls.autoRotate = true;

scene = initModel();

var loader = new THREE.JSONLoader();
loader.load('monitor.json', function (geometry, materials)
{
     var geo = geometry;
     var mat = new THREE.MeshFaceMaterial(materials);
     var mesh = new THREE.Mesh(geo, mat);
     scene.add(mesh);
});

initFlow();

baseTime = +new Date;
render();

startNetwork();
}
```

```
// ---------------------------------------------- レンダリング
function render() {
requestAnimationFrame(render);
// カメラコントロールの状態を更新
controls.update();

updateFlow();

renderer.render(scene, camera);
};


// パケットフローオブジェクトをあらかじめ準備しておく
function initFlow() {
for(var i=0; i < MAX_FLOW; i++) {
var packet = {};

// パケットの経路に見立てた線分
var lineGeometry = new THREE.Geometry();
lineGeometry.vertices.push(new THREE.Vector3(0, 0, 0));
lineGeometry.vertices.push(new THREE.Vector3(0, 0, 0));
var lineMaterial = new THREE.LineBasicMaterial({color: 0xffffff, linewidth: 1, transparent: false, opacity: 0.1});
var line = new THREE.Line(lineGeometry, lineMaterial);

// パケットに見立てた球体
var sphereGeometry = new THREE.SphereGeometry(0.4);
var sphereMaterial = new THREE.MeshStandardMaterial({color: 0xffffff});
var sphere = new THREE.Mesh(sphereGeometry, sphereMaterial);

line.visible = false;
sphere.visible = false;

scene.add(line);
scene.add(sphere);

// オブジェクトに格納
packet.line = line;
packet.lineG = lineGeometry;
packet.lineM = lineMaterial;
packet.sphere = sphere;
packet.sphereG = sphereGeometry;
packet.sphereM = sphereMaterial;
packet.progress = 1.0;
packet.available = true;

flow.push(packet);
}
}


// パケットフローオブジェクトを追加する
function setFlow(pkt) {

var packet = flow[marker];
marker = (marker + 1) % MAX_FLOW;

// 仰角
var phi = pkt.latlng[0] * (Math.PI / 180);
// 方位角
var theta = pkt.latlng[1] * (Math.PI / 180) * -1;

// ホワイトリストに存在するパケットなら色を緑
if(pkt.white == 1) var packetColor = 0x00ff00;
else var packetColor = 0xff00cc;

// パケットの経路の設定
var line = packet.line;
var lineGeometry = packet.lineG;
```

```
lineGeometry.verticesNeedUpdate = true;
lineGeometry.vertices[1].x = radius * Math.cos(phi) * Math.cos(theta);
lineGeometry.vertices[1].y = radius * Math.sin(phi);
lineGeometry.vertices[1].z = radius * Math.cos(phi) * Math.sin(theta);
var lineMaterial = packet.lineM;
lineMaterial.color.setHex(packetColor);

// パケットに見立てた球体の設定
var sphereMaterial = packet.sphereM;
sphereMaterial.color.setHex(packetColor);

// シーンに追加
if(packet.available == true) {
packet.line.visible = true;
packet.sphere.visible = true;
}

// パケットフローオブジェクトに情報を追加する
packet.phi = phi;
packet.theta = theta;
packet.progress = 0.0;
packet.available = false;
packet.direction = pkt.direction;
}


// パケットフローオブジェクトの状態を更新する
function updateFlow() {
for(var i=0; i < MAX_FLOW; i++) {
var packet = flow[i];

// パケットフローが空なら何もしない
if(packet.available == true) {
continue;
}

if(packet.progress < 1.0) {
var phi = packet.phi;
var theta = packet.theta;
var direction = packet.direction;
var progress = packet.progress;

var sphere = packet.sphere;
if(direction == 0) {
sphere.position.x = radius * Math.cos(phi) * Math.cos(theta) * progress;
sphere.position.y = radius * Math.sin(phi) * progress;
sphere.position.z = radius * Math.cos(phi) * Math.sin(theta) * progress;
} else {
sphere.position.x = radius * Math.cos(phi) * Math.cos(theta) * (1.0 - progress);
sphere.position.y = radius * Math.sin(phi) * (1.0 - progress);
sphere.position.z = radius * Math.cos(phi) * Math.sin(theta) * (1.0 - progress);
}

packet.progress += 0.01;
} else {
packet.sphere.visible = false;
packet.line.visible = false;
packet.available = true;
}
}
}


// ---------------------------------------------- socket 処理
function startNetwork() {
// 接続開始
socket = io.connect();

// 接続時
```

```
socket.on('connect', function() {
console.log('start socket.io');
socket.emit('capture');
});

// 受信時イベント
socket.on('packet', function(data) {
var packet = JSON.parse(data);

if(packet.country != '??') {
setFlow(packet);
} else {
setFlow({latlng:[-90, 0], white: 0, direction: packet.direction});
}
insertInfoTable(packet);
});

socket.on('disconnect', function() {
console.log('stopped socket.io');
});
}

// パケット情報を表に追加する
function insertInfoTable(packet) {
// 行の設定
var $row = $("<tr></tr>")
.append($("<td class='datetime'></td>").text(packet.datetime))
.append($("<td class='country'></td></td>").text(packet.country))
.append($("<td class='s_addr'></td></td>").text(packet.address[0]))
.append($("<td class='d_addr'></td></td>").text(packet.address[1]))
.append($("<td class='protocol'></td></td>").text(packet.protocol))
.append($("<td class='length'></td></td>").text(packet.length))
.append($("<td class='s_port'></td></td>").text(packet.port[0]))
.append($("<td class='d_port'></td></td>").text(packet.port[1]));

// テーブル取得
var $table = $('#infoTable tbody');
// テーブルに行を追加
$table.append($row);

// 最下部までスクロール
$table.scrollTop($table[0].scrollHeight);
}


})();
```

付 録B　対外発表リスト

1. 田村尚規，甲斐博，森井昌克，“パケット情報を用いたトラフィック可視化システムの作成”，FIT 2015 第 14 回情報科学技術フォーラム，2015.