第 XIX 部

IP トレースバック・システムの研究開発

第 19 部
IP トレースバック・システムの研究開発

第 1 章　はじめに

IP Traceback WG は、分散型サービス妨害攻撃への対策手法の一つである IP トレースバック技術を研究することを目的として設立されたワーキンググループ（Working Group、分科会）である。

本報告書は、2003 年度に論文として発表された研究成果をまとめたものである。各論文の概要は次の通りである。

- 大江将史、門林雄基、“階層型 IP トレースバック機構の実装と検証”、IEICE trans. commun., vol.J86-B, no.8, Aug.2003.
- M. Oe, Y. Kadobayashi and S. Yamaguchi, “An implementation of a hierarchical IP traceback architecture,” SAINT’03, Florida, USA, Feb. 2003.

  これらの論文は、実インターネットでの IP トレースバック運用における問題への解決策として「階層型トレースバック機構」を提案した。本論文では、各種プロトコルの定義、実証実験に向けてのプロトタイプ実装についてのアーキテクチャ設計、それらの動作検証を行い、そして、結果からより大規模なシミュレーションを行うことが必要であることを明らかにしている。これらについては、それぞれ 2、3 章で述べる。
- H. Hazeyama, Y. Kadobayashi, “An Layer-2 Extension to Hash-based IP Traceback,” IEICE Transactions on Information and Systems: Special Issue on the New Technology in the Internet and their applications, Nov. 2003.

  本論文は、広域に展開されるレイヤー 2 網（たとえば、xDSL やワイヤレス LAN）における IP トレースバック技術の提案と検証を行っている。広域の L2 網における IP トレースバックは、攻撃ノードの具体的位置を特定するために、必須の技術である。筆者らは、A.C. Snoeren らが提案する Hash-based IP traceback をベースに、ルータのインターフェースから、パケット単位で攻撃ノードの接続されたポートの特定を行う手法を提案した。有効性の検証は、提案手法を実装し、IP トレースバックの実行とその評価によって行っている。これについては、4 章で述べる。

我々は、実インターネットでの IP トレースバックを行う上で必要な技術の提案と検証を、レイヤ 3 上での IP トレースバックとレイヤ 2 上での IP トレースバックに分けて研究を行った。そして、個々の IP トレースバック技術は分散型妨害攻撃の攻撃フローを特定可能であるということを示した。

しかしながら、実運用においては、より効率よく短時間に追跡することや、各種抗分散型サービス妨害攻撃技術との連携が求められる。そこで、現在、分散型サービス妨害攻撃の検出技術とこれらの各種 IP トレースバック手法を組み合わせ、より包括的な運用メカニズムの研究開発を行っている。

第 2 章　階層型 IP トレースバック機構の実装と検証

概要

分散型サービス妨害攻撃への対策手法として、IP トレースバック技術がある。この技術は、発信元アドレスが偽装された攻撃フローの通過したルータを特定することができる。この問題の解決策として種々の手法が提案されているが、実インターネットでの運用においては様々な問題を有している。筆者らは、この問題点の解決策として「階層型トレースバック機構」を提案した。本章では、各種プロトコルの定義、実証実験に向けてのプロトタイプ実装についてのアーキテクチャ設計、それらの動作検証を行った。そして、結果からより大規模なシミュレーションを行うことが必要であることが明らかになった。

2.1 まえがき

分散型サービス妨害攻撃（DDoS 攻撃: Distributed Denial of Service Attack）は、インターネットにおける脅威である。2000 年 2 月、米 Yahoo!に対して、分散型サービス妨害攻撃が行われた [167]。この攻撃によって、米 Yahoo!は正常なサービス運用が不可能となり、その被害額は、数百万ドルに達したとされている。分散型サービス妨害攻撃とは、インターネット上に分散して設置された攻撃ノードから、被害ノードに対して大量のパケットを送ることによって、帯域などのリソースを奪い WWW や FTP といった正規のサービスを妨害する攻撃手法である。

分散型サービス妨害攻撃への対策は、事前対策と事後対策がある。事前対策とは、攻撃ノード設置の阻止とその除去である。攻撃ノードの設置は、システムのもつ脆弱性を利用した不正アクセスを通して行われたり、ワームやウイルスを利用したトロイの木馬によって行なわれる。よって、脆弱性に対する対策や、攻撃ノードの発見と分離などで防ぐことができる。しかしながら、CERT などの各インシデント機関を通して脆弱性に関する情報提供は広く行われているにも関わらず、システムの管理者が十分な対策を行っていない場合がある。よって、事前対策のみで分散型サービス妨害攻撃を根絶することは困難である。

事後対策とは、発生したサービス妨害攻撃から攻撃ノードを特定し、インターネットからの隔離や経路上でのフィルタリングによって攻撃パケットの転送を防ぐことである。本章において着目する IP トレースバック技術がこの手法の一つである。被害ノードは、IP トレースバックによって、短期間に分散型サービス妨害攻撃への対策を行うことができ、その被害を最小限にすることが可能となる。

IP トレースバックは、送信元が偽装された攻撃パケットの転送経路を特定することができる。攻撃者は、攻撃ノードの特定を困難にするために発信元アドレスを偽装する。このため、traceroute といった発信元アドレスや経路情報を元にした追跡は不可能である。現在は、ルータのフィルタリング機能やサービス妨害攻撃検知機能などを使い手作業によって、攻撃ノードの特定をおこなっている。しかし、インターネットは、国際的な通信インフラであり、商用 ISP や、企業、研究・学術機関といった様々なポリシーをもった機関が相互に接続して成り立っている。このため手作業による追跡では、その境界を越えた追跡を行う際のコストや国際的な協力が必要となり、追跡時間の短縮に対する大きな障壁となっている。IP トレースバックはこの追跡過程を自動化することで、その追跡に要するコストの削減と分散型サービス妨害に対する対策時間を短縮することができる。

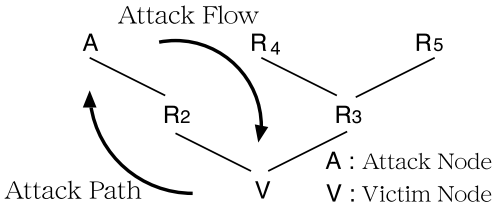

図 2.1.　攻撃フローと攻撃パス

IP トレースバックでは、図 2.1 に示す攻撃ノードから発信された攻撃パケットの流れを「攻撃フロー（Attack Flow）」とし、その攻撃フローが通過した経路を「攻撃パス（Attack Path）」とする。そして、この攻撃フローに対する攻撃パスを特定する技術を IP トレースバックと定める。

本研究では、筆者らが提案している階層型 IP トレースバック機構についての概観を述べ、インターネットでの運用の可能性について論ずる。そして、提案手法の実装アーキテクチャや各種プロトコルについて述べ、その動作検証結果を報告する。

以後、2.2 節では関連研究および階層型 IP トレースバック機構の説明、そして、2.3 節にて提案手法のプロトコル設計と実装アーキテクチャについて述べ、2.4 節にてその実装の動作検証と実用化にむけての考察を行い、2.5 節にてまとめと今後の課題を述べる。

2.2 関連研究

筆者らは、IP トレースバックに関する技術調査を行い文献 [343] においてまとめ、提案されている種々の IP トレースバック技術の問題点を指摘した。そして、その問題の解決策として、筆者らは、文献 [218, 339] において「階層型 IP トレースバック」を提案し、標準化活動を行っている [219]。

本節では、既存の IP トレースバック技術を概観し、筆者らの提案手法について述べる。

2.2.1 既存の IP トレースバック手法

既存の IP トレースバック技術は、その手法毎に 4 つに分類することができる。

1. **リンク検査手法**
   ルータの持つフィルタリング機能を利用し、攻撃フローの特定を行う手法 [272] である。既存の機材を利用して追跡を行うため、その投資コストは最小限である。しかしながら、攻撃フローが発生している期間しかフローの追跡ができない点、管理者の行う手動追跡であるため追跡時間を必要とする点が問題である。
2. **逆探知パケット手法**
   ルータなどが攻撃パスを構成するために必要な情報を専用の IP パケット（これを、逆探知パケットという。）に記録、送信し、これらから攻撃パスの構成を行う手法である [12]。この手法は、既存バックボーン・システムに影響を与えることなく導入することが可能である。しかしながら、逆探知パケットによるインターネットのトラヒック増加が問題である。
3. **マーキング手法**
   各ルータが、IP ヘッダ中に IP トレースバックに必要な情報を記録することで攻撃パスを特定する手法である [253, 268]。IP トレースバックに伴う追加のトラヒックが一切発生しないが、攻撃ノードが偽造マーキングを生成した場合、攻撃パスの構築に必要な計算量が増大する点など抗トレースバック攻撃に対する防御は難しい点が問題である。
4. **ダイジェスト手法**
   ハッシュ関数を利用した記憶容量の効率が高い手法で、ルータ上を通過するパケットを保存する。そして、パケットの保存記録から攻撃パスの特定を行う手法である [265]。この手法は 1 パケット単位での追跡が可能であるが、監視対象のインターフェース毎に、記録データを転送するためのネットワーク帯域と保存のためのストレージ、そしてそれらの管理コストが必要となる点が問題である。

以上の既存研究から、現実のインターネット上で IP トレースバックを実現するには、次に述べる 3 つの問題がある。

1. 現在のインターネットは、AS と呼ばれる経路制御の管理単位毎に独立した管理・運用が行われている。このため、AS 間のトラブルへの対処は、AS 間での連携が必要となる。
2. IP トレースバック・システムの運用コストは、手法毎に異なる。運用要件として、すべてのルータに対する改変を必要とする IP トレースバック手法や IP トレースバック用の情報を保管するためシステムの運用と保守を必要とする手法もある。従って、ISP 等の予算や運用規模に応じた IP トレースバック手法を運用する必要がある。
3. 日々攻撃手法が進化する状況を考えれば、攻撃者は、種々の IP トレースバック手法を分析しその弱点を狙った攻撃手法を開発すると予想される。したがって、それに併せて、IP トレースバック手法も改良や変更を行わなければならない。

以上の 3 点から、一つの IP トレースバック手法をインターネット全体に対して適用することは困難であると考える。そこで筆者らは、現在のインターネット上での経路制御機構に着目し上記の問題点を解決できる IP トレースバック・アーキテクチャの設計を行った。

インターネットでの経路制御は、一つの経路制御プロトコルによって行われているのではなく、ネットワークの規模に応じて、EGP（Exterior Gateway Protocol）と IGP（Interior Gateway Protocol）の二つに階層化されている。AS といった大規模なネットワーク間での経路制御には EGP が用いられ、そのプロトコルとしては BGP（Boarder Gateway Protocol）4/4+がある。一方、AS 内においては IGP として OSPF（Open Shortest Path First）や IS-IS（Intermediate System-to-Intermediate System）が用いられている。EGP 運用では、AS 間において契約に基づいた相互接続によって、経路情報の交換を行っている。

上記の経路制御の機構をトレースバック機構に応用したものが「階層型 IP トレースバック機構」である。

2.2.2 階層型 IP トレースバック機構

本機構は、「Exterior IP（eIP）トレースバック機構」、「Interior IP（iIP）トレースバック機構」、「ITM（IP traceback Manager）ネットワーク」の 3 つの要素で構成されている。各 eIP/iIP トレースバックの対象となるネットワークの範囲は、図 2.2 に示すように経路制御の EGP/IGP の制御範囲と同一である。ITM ネットワークは、eIP トレースバックと iIP トレースバック間を連携するために用いられる。

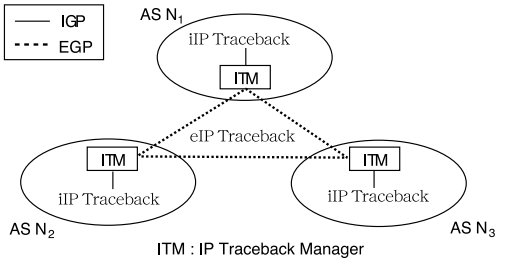

図 2.2.　階層型 IP トレースバック機構

eIP トレースバック機構は、攻撃フローの通過した AS を特定する。本研究では、追跡開始から 30 分以内での攻撃パスの特定を目標としている。これは、CAIDA[201] の報告において、分散型サービス妨害攻撃の初期攻撃段階が 30 分で終了するという点から定めた。しかしながら、現状の研究において、AS を特定する IP トレースバック手法は提案されていないため、筆者らは、逆探知パケット手法の一手法である「IP オプション・トレースバック」を提案した [339]。

iIP トレースバック機構は、AS 内を通過する攻撃フローの通過したルータのアドレスを特定する。攻撃ノードの AS を特定する eIP トレースバックと攻撃ノードの IP アドレスを特定する iIP トレースバックの連携によって分散型サービス妨害攻撃への対策を行う。

筆者らは、既存の IP トレースバックの調査から eIP と iIP トレースバック間の連携に必要なパラメータを 3 つ定めた。これらは、攻撃フローのパケットダンプ記録、記録タイムスタンプ、記録ノード情報（AS 番号／IP アドレス）である。本章では、既存の IP トレースバック技術を iIP トレースバックとして運用し、eIP トレースバックとの連携のために必要な拡張について述べる。

ITM ネットワークは、eIP/iIP トレースバックを運用している AS 間の情報交換のために用いられる。各 ITM は、ITM プロトコル（ITM Protocol、ITMP）を用いて BGP ピアリングを行っている AS の ITM と相互接続を行い、情報交換を行う。

本手法による分散型サービス妨害攻撃への対策は、次に示す過程を経て行われる。

1. **攻撃フローの記録:** 被害ノードの管理者は、所属する AS の管理者へ攻撃フローに対する対策を要求する。AS 管理者は、被害ノードにおいて攻撃フローのモニタリングと記録を行う。
2. **eIP トレースバックの実行:** 記録から eIP トレースバックによって攻撃フローが通過した AS で構成された攻撃パスを求める。
3. **初期対策の実行:** 攻撃パス上の AS は、フィルタリングや帯域のシェーピング等の対策を行い被害者に対する早急な被害の緩和を行う。
4. **iIP トレースバックの実行:** 各攻撃ノードの存在する AS 上で iIP トレースバックを行い、攻撃ノードの IP アドレスを特定しネットワークから遮断する。
5. **攻撃の収束:** 攻撃ノードのネットワークからの分離により、攻撃フローは収束し、分散型サービス妨害攻撃は終了する。

iIP トレースバックは、AS 毎に独立して実行することができる。よって、AS は、各 AS の運用形態に適合した IP トレースバックの手法を選択することができる。そして、AS は、運用中の iIP トレースバック機構に対して脆弱性が発見された場合や運用規模に不適となった場合、別の手法に変更することが可能である。

2.2.3 IP オプション・トレースバックの解析

IP オプション・トレースバック手法は逆探知パケット手法に分類されるため、ネットワークにかかる負担についての考察が必要である。そこで、筆者らは、文献 [254] において、数学モデルの構築を行い、その解析によりネットワークのトラヒック増加量と攻撃パス構築に要する時間のトレードオフを明らかにした。その結果として、筆者らが目標としている 30 分以内での攻撃パスの特定はトラヒックへの影響が小さい範囲（増分が 0.1%以下）において十分可能であることを示した。

解析結果を基に、「階層型 IP トレースバック機構」と eIP トレースバックとして「IP オプション・トレースバック」の実装アーキテクチャの設計と実装を行った。

2.3 提案手法のプロトタイプ実装

本節では、筆者らが FreeBSD 4.7 および、NetBSD 1.6 上で開発した提案手法のプロトタイプ実装とその動作検証について述べる。筆者らは、実装アーキテクチャの設計を通して ITM 間の通信プロトコルである ITMP の定義および、eIP/iIP トレースバックと ITM 間が連携するための情報交換を行うため

の ITM API の定義を行い。それらの構成モデルを設計した。そして、設計を元にプロトタイプの実装と動作確認を行った。

2.3.1 IP オプション・トレースバックの構成

本項では、IP オプション・トレースバックの構成について述べる。IP オプション・トレースバックは、文献 [339] において定義された。しかしながら、実装・運用を行う上で ITM と本手法間での独立性をより高めた構成にしなければならないことが判明した。これをふまえて、IP オプション・トレースバックの機能構成部分に関して仕様変更を行った。攻撃パスの特定アルゴリズムに対する変更は行っていないため、先に示した解析結果には影響は無い。

本手法は、PM（Packet Monitor）と TOG（Traceback Option Generator）の 2 つの機構で構成されている（図 2.3）。PM は、AS 上の各 BGP 境界ルータ（Border Router）上に存在し、AS 外へフォワーディングされる IP パケットの観測とサンプリングを行う。TOG は、各 AS 上に 1 つ存在し、その AS 内の PM の管理と IP オプションを用いた逆探知パケットの生成と解析を行う。

PM は、次に示す 2 つの機能で構成されている。

1. **パケットの選択:** ルータの各インターフェース出力から、確率 P でパケットを選択しコピーする。
2. **TOG との連携:** PM は、TOG から要求に応じた確率 P の設定や選択したパケットの TOG への転送を行う。

次に、TOG の機能について述べる。TOG は、5 つの機能で構成されている。

1. **IP トレースバック・オプションの生成:** TOG は、選択されたパケットとパラメータ（AS 番号、偽装防止用ハッシュ情報）から IP トレースバック・オプションを生成し送信する。
2. **オプション管理:** TOG は、AS メッセージ鍵 X と鍵識別番号 Z を生成し管理する。ハッシュ情報は、AS メッセージ鍵 X の鍵識別番号 Z から HMAC（Keyed-Hash Message Authentication Code）[162] を用いて生成される。
3. **PM との連携:** TOG は、PM でパケットを選択する際に用いる確率 P を各 PM へ送信する。
4. **パケットの検証:** TOG は、IP トレースバック・オプション中の HMAC 認証を検証し、攻撃パスの構築を行う。
5. **ITM 間の連携:** 攻撃パスの構築のために、TOG は、近隣の TOG との間で、攻撃パスや IP トレースバック・オプションを、ITM ネットワークを用いて交換する。

本手法では、AS 番号を HMAC 認証によって MAC データにすることによって、トレースバック・オプションに AS 番号をクリアテキスト形式で保持しないようにしている。これは、ICMP トレースバックの持つ逆探知パケットからのネットワークトポロジー情報が流出するという問題点を解決している。

2.3.2 ITM API の定義

我々は、eIP と iIP トレースバック間での連携を行うために ITM API を定義した。eIP/iIP トレースバックと ITM は、この API を経由して IP トレースバックを実行するための情報や、実行結果についての情報を交換しなければならない。eIP/iIP トレースバック・システムと ITM の構成を図 2.4 に示す。このように ITM API を用いて ITM と接続されており、各 AS の ITM は、ITMP を使用して相互に接続している。

本実装における ITM API の設計は、実在する実装を iIP もしくは eIP トレースバック・モジュールとして ITM と連携させることを目的とした。なお、参照した実装として、iIP トレースバックとして

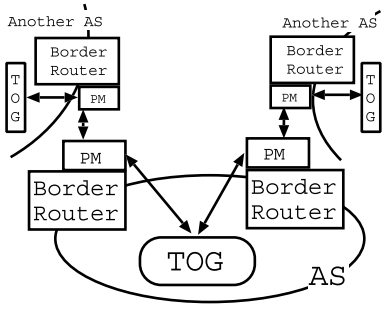

図 2.3. IP オプション・トレースバック機構の構成

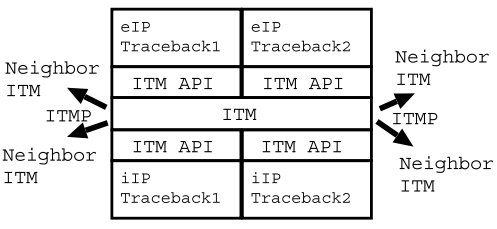

図 2.4. ITM におけるモジュール構成

は、Hash-based IP トレースバックを NP（Network Processor）へ実装した横河電機 PAFFI[326]、また、eIP トレースバックとして、筆者らが開発した IP オプション・トレースバック実装を利用した。

API は、次に示す構成となっている。

- **ITM 情報交換 API:** 接続を確立している近隣 ITM の AS 番号、接続状態、サポートする eIP/iIP トレースバックなどの情報を入出力する。
- **データ交換 API:** 近隣 ITM の eIP/iIP トレースバック・システム同士でのデータ交換（ITM はデータ中継を行うパイプとなる）や、同 IP トレースバック・システムの状態を入出力する。
- **トレースバック要求と応答 API:** 攻撃フローのパケットダンプ情報（記録タイムスタンプ、記録ノード AS 番号/IP アドレス）入力、攻撃パス情報出力、IP トレースバックの制御（中断、状況報告など）を行う。

各 IP トレースバックが ITM API を利用した実装を行うことによって、各 IP トレースバックは eIP-iIP 間の連携性を失うことなく独立性をもつことができる。そして、ITM ネットワークを介して近隣関係にある AS の IP トレースバック・モジュールと連携することも可能である。したがって、本提案手法は、ITM による各 AS との連携を前提とした eIP トレースバックのプロトコル・デザインを実現する。

本実装においては、IP オプション・トレースバック実装を ITM API を使用することによって、eIP トレースバック・モジュールとして利用可能とし、ITM ネットワーク・プラットフォーム上で、隣接 AS と連携し攻撃パス構築を行っている。

2.3.3 ITMP

ここでは、ITM と ITM 間のデータ交換に用いられる ITMP について述べる。ITM ネットワークは、BGP によってピアリングを行っている AS 同士が、ITM についても同様に ITMP を用いてピアリングを行うことで構築されるものである。そのため、ITM ネットワークのトポロジは、AS ネットワークのトポロジと同等になる。

図 2.5 は、ITMP における状態遷移を示している。ITMP は、認証フェーズ（authentication phase）と接続確立フェーズ（connected phase）がある。認証フェーズは、ダイジェスト認証を用いて隣接 ITM との接続を行う。認証完了後、近隣情報（AS 番号、

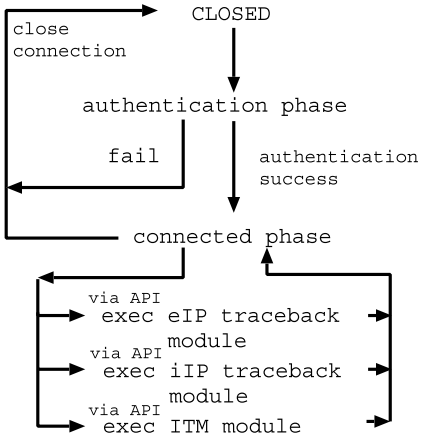

図 2.5.　ITMP の状態遷移

利用可能な eIP トレースバックと iIP トレースバック）の交換を行う。認証フェーズ終了後、接続確立フェーズに移行する。接続確立フェーズにおいては、各 eIP/iIP トレースバック・モジュールの実行や、ITM モジュールによる隣接する AS の IP トレースバック・モジュール同士での ITM ネットワークを介したデータ交換を実行する。

2.3.4 実行コードの構成

本実行コードは、ITM に IP オプション・トレースバック TOG がモジュール化され組み込まれた部分と、パケットの選択と転送などを行う IP オプション・トレースバック PM の部分に分けて実行されている。プログラムは、移植性を重視し C と標準ライブラリを組み合わせて記述されている。なお、IP オプション・トレースバック PM 部においては、libpcap ライブラリ [281] を、TOG 部には libnet ライブラリ [255] を外部ライブラリとしてそれぞれ利用している。しかしながら、論文執筆時点では、PAFFI（iIP トレースバック）との連携部分の設計と検証は行われたが実装は完了していない。

ITMP の通信には、TCP/IP を利用し、IPv4/IPv6 双方での接続が可能となっており、IPv4/IPv6 毎に独立した ITM ネットワークが構築可能となっている。よって、IPv6 上と IPv4 上での ITM および、そのトレースバック・モジュール群は、互いに独立して運用することとなる。

プロトタイプ実装における IP オプション・トレースバックの PM は、NIC 上のトラヒック・モニタリングとパケットの選択を行う。そのために、既存インフラストラクチャへの影響を最小限にしつつ、ルー

タのトラヒック・モニタリングを行う必要がある。よって、ルータのNICをミラーリングが可能なイーサーネット・スイッチングハブ、もしくは、光スプリッタなどへ接続し、それらのトラヒックをPM上でモニタリング可能な環境となっている事が必要である。

2.4 動作検証と実用化にむけての考察

筆者らは、一連の提案手法を実現する上での検証過程について考察し、本章における検証の位置づけを明らかにし、動作検証用の環境を構築し、以上に述べた実装の動作確認を行なった。本節では、検証環境とその結果をふまえた考察について述べる。

2.4.1 提案手法の検証過程

本提案手法の実インターネット上での有効性を示すためには、次に示す過程に沿った検証が必要である。

1. 提案手法を具体化するためのアーキテクチャ設計とその定性的考察
2. アーキテクチャ設計に基づくプロトコルの仕様策定と実装の開発
3. 10程度の小規模なAS間に展開されるITMネットワーク上でのeIPトレースバック(IPオプション・トレースバック）の連携部分の動作検証
4. 各ITM上でeIPトレースバックからITMを介したiIPトレースバックへの連携検証
5. 100を超える大規模なAS間におけるITMネットワーク上でのeIPトレースバックの動作検証
6. 実インターネットでの実証実験

筆者らは、文献[339]において、項目1.については完了している。そして、本章では、項目2.および3.を実現することを目標とした。これは、筆者らが準備を進めているPAFFIとのインターフェース設計の実装が完了していない点と、PAFFIでは既にIPトレースバックの動作検証は完了していることを考慮し、ITMとeIP間の連携の検証が現時点では重要であると考察したからである。

しかしながら、今回行った検証は、実インターネットの環境とは規模の面において乖離している。そのため、筆者らは本実験をあくまで動作検証の枠内での位置づけとした。

そして、実インターネットでの運用規模を考慮した項目5.の検証には大規模なシミュレーションが必要であると考察する。筆者らはその検証規模を、CAIDAのSkitterプロジェクト[287]におけるBGPのピアリング規模の観測結果を基に選定することを想定している。

Skitterは、BGPのピアリングの観測からトラヒックの集中する中心的なASグループを定義している。筆者らは、分散された攻撃フローは、通常のトラヒックと同様にこれらのASを通過せざるを得ないと考え、ITMネットワークは、これらの中心的なASにおいて運用することが必要であると考える。

これらの中心的なAS規模における検証を項目5.として行うことが項目6.における実インターネットでの運用にむけて必要な事だと考察する。筆者らは、実インターネットでの導入を前提とした実験計画を策定し、2.5節において課題としてまとめる。

2.4.2 動作検証

プロトタイプ実装の動作検証は、図2.6のように構成された小規模ネットワークテスト環境で行った。この環境は、ルータとなる10台のPCとそれらを収納するL2スイッチ（L2 Switching HUB、Extreme Networks社製Summit48）および、トラヒック生成機（Traffic Generator、Agilent社製Router Tester)、そして、これらのネットワーク構成変更やトラヒックの制御を行うコントローラ機（Ctrl. Unit）で構成されている。各PCは、2つ以上のネットワーク・インターフェース・カード（NIC）を持ち、すべてのNICは、L2スイッチに収納されている。コントローラ上で、L2スイッチハブのバーチャルLANを設定する事によって、物理的な変更を行うことなく柔軟なトポロジ設計が可能となっている。また、トラヒック生成機とバーチャルLANの組み合わせによって、数十ノード程度の攻撃ノードから生成される攻撃フローの生成と観測が可能となっている。

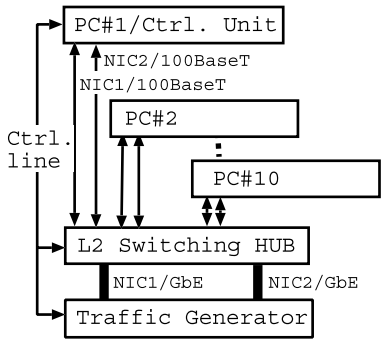

図 2.6. 検証用システムの構成

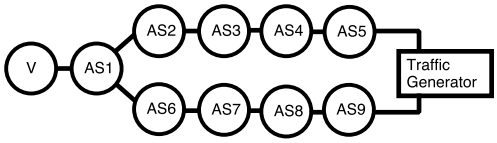

図 2.7.　ネットワーク構成

今回は、本システム上の各 PC を独立した AS と想定し、eIP トレースバックと ITM ネットワーク間の実装動作検証を行った。iIP トレースバックは、eIP トレースバックや ITM ネットワークとは独立に閉じた環境で実行される。また、先に示すように連携部分の実装が未完了である事から、ITM からの iIP トレースバックを行うための情報を受け取るのみのスタブ・モジュールとした。

そして、9 台の PC を AS（図 2.7 の AS1〜AS9）として、また、1 台の PC を被害ノード（図 2.7 中の V）として、図 2.7 に示すネットワークを構築し、次の点についての動作確認を行った。1. トラヒック生成機による攻撃トラヒックに対して、PM がパケットモニタし確率 P に従い TOG と連携する点、2. PM から送信されたパケットを元に、TOG が IP オプション・パケットを送信する点、3. 各 PC 上の ITM 間で認証フェーズから接続確立フェーズへ遷移する点、4. IP オプション・パケットからの攻撃パスの構築が可能な点、以上より、プロトタイプ実装が設計に従った動作をすることを確認した。

2.4.3 実現性に関する考察

ここでは、本提案の実現可能性について述べる。

筆者らは、ITM の実現可能性は高いと考えている。なぜなら、分散型サービス妨害攻撃によって顧客のサイトが被る被害を最小限することがサービス品質と信頼を保つために非常に重要であり、ISP は、短期間に対策可能な手法を必要としているからである。そして、ISP は現在リンク検査手法による IP トレースバックによって、攻撃パスの特定を行っており、このため ISP は、分散型サービス妨害攻撃への対処のために多額の費用、時間、そして、人的資源の消費を強いられている。

本提案手法は、各 IP トレースバック・システムを ITM の元に統合し、ITM 間（AS 間）の追跡過程を自動化することによって、対処に要するコストを減らすことができる。さらに、各 AS は、ネットワーク規模や予算に応じた IP トレースバック手法を選択することから、本提案手法は柔軟性をもった IP トレースバック・システムの運用を実現する。

以上の事から、各 AS は ITM の運用を行うメリットがあり、本手法の実用性は高いと考察する。

2.5 まとめと今後の課題

筆者らが既存問題への解決策として提案している階層型 IP トレースバック機構は、IP トレースバックをインターネットにおける経路制御アーキテクチャである EGP と IGP に従って、eIP トレースバックと iIP トレースバックの 2 つに分離した IP トレースバックを行うものである。

この提案を元に、既存の IP トレースバック機構の分析から既存手法を iIP トレースバック・モジュールとして利用するため、ITM API（ITM 情報交換 API、データ交換 API、トレースバック要求と応答 API）を定めた。また、ITM ネットワーク上で各 eIP と iIP トレースバック間の連携や制御をするための通信プロトコル ITMP を定めた。

ITM API と ITMP の設計に基づきプロトタイプの開発をおこなった。

そして、本提案手法の実インターネットにおける運用を目標とした検証過程を明確にし、本章における検証の位置づけを、10 程度の小規模な AS 間における ITM と eIP トレースバック間の連携の動作検証とし検証を行った。その結果、本提案手法のプロトコル設計や実装は、大規模シミュレーションを行うことが可能であることが明らかになった。

今後、筆者らは、本実装を 500 物理ノード（5000 仮想ノード）でのインターネット・シミュレータである StarBED[282] において、実インターネットを想定したシミュレーションを予定している。この実験では、Skitter に基づく AS の構成を構築し、トラヒック生成機による仮想攻撃フローの生成を行う。そして、ITM と eIP/iIP トレースバックの連携の動作検証を行い、攻撃パス構成時間やネットワークへの負荷といった性能を明らかにする。これにより、我々の提案手法の実インターネットでの運用に向けての問題点を明確にしていきたい。

第 3 章 An implementation of a hierarchical IP traceback architecture

Abstract

The IP traceback technique detects sources of attack nodes and the paths traversed by anonymous DDoS (Distributed Denial of Service) flows with spoofed source addresses. We propose a hierarchical IP traceback architecture, which decomposes the Internet-wide traceback procedure into inter-domain traceback and intradomain traceback. Our proposed method is different from existing approaches in that our method is independent from a single IP traceback mechanism, and domain decomposition is based on existing operational models of the Internet. Moreover, it has the capability of being used for not only the IPv4 network, but also the IPv6 network.

3.1 Introduction

Distributed Denial of Service (DDoS) attacks are one of the threats on the Internet. In DDoS attacks, the attack nodes are widely set up in the Internet, and transmit a large number of packets to the victim's node. These packets consume network resources and server resources, and obstruct network services like World-Wide-Web in the victim's node. In order to keep this damage to a minimum, the technology that identifies attack nodes and the paths of attack packets is required to be as fast as possible.

However, the source address of the IP packet for DDoS attack is spoofed to a random address. Therefore, investigating an attack node using TRACEROUTE, which depends on a source address, is not effective. For this reason, tracking the attack flow is done by hand using a filtering function, a DDoS attack detection function, (which each router has), etc. The Internet, as an international communications infrastructure, is constructed by various organizations

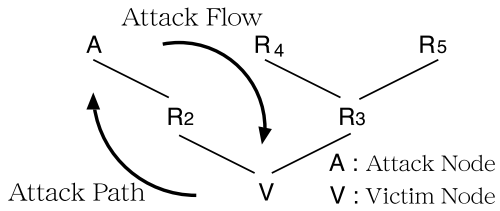

Fig. 3.1. IP traceback: detecting attack flows

connecting each other with various policies, such as ISPs, companies, research institutes, universities, etc. When tracking attack flows, a lot of time is wasted getting cooperation between organizations on attack paths. Tracking by hand is a big barrier to the reduction of time. IP traceback methods are proposed as solution to this issue. IP traceback detects the attack paths as shown in Fig. 3.1 and specifies the true origin of an attack flow with a spoofed source address.

In this chapter, we propose a hierarchical IP traceback architecture, which decomposes Internet-wide traceback procedure into inter-domain traceback and intradomain traceback. Our proposed method is different from existing approaches in that our method is independent from single IP traceback mechanisms, and domain decomposition is based on the existing operational model of the Internet.

The rest of this chapter is organized as follows: In section 3.2, we describe related work. Section 3.3 outlines our proposed technique. In Section 3.4 we describe the implementation of our proposal. Finally, we summarize our findings and future work in Section 3.5.

3.2 Related work

We briefly introduce three typical proposed methods for IP traceback.

3.2.1 Link testing method

This method specifies the IP address of a router that forwards the attack flow by the filtering function and the monitoring function in the router. The attack path is clarified by repeating the search from the victim's node to the attacker node on each router. Additional equipment and

materials for the traceback are not required. However, this method can traceback only while the victim's node is under DDoS attack and the attack flow is active on the Internet.

3.2.2 Hash-based method

Snoeren et al [265] have proposed this method. Every router logs all transmission packets into storage with a hash function for compression of data. This method has a high capability for locating the attacking host. However, this method requires 0.1% of interface bandwidth on each interface for recoding. Large amounts of cost and hardware infrastructure are required to store recoding data.

3.2.3 Passive detection method

This method was proposed by Bellovin[12]. Every router sends the router's information, that is, IP address, MAC address, next hop router's IP address and so on into the passive detection packet. The victim hosts can identify the attack path by collecting these packets. Installation cost is low because this method does not require the storage area of the packet and the processing of each packet. However, the passive detection packets increase traffic on the Internet.

3.2.4 IP traceback difficulty over the Internet

In this section, we describe the analytical result of the existing IP traceback methods and the reason it is difficult to operate IP traceback over the Internet. We believe two issues exist to implement IP traceback on the Internet,

1. Organizations connected with the Internet as Autonomous System (AS) does the management and operation independently for each AS. Therefore, emergency action for traceback requires the cooperation between two or more AS over a large amount of time.

2. We expect the attacker to analyze the weak points of current IP traceback methods and develop anti-IP traceback attacks as a countermeasure.

From the two points above, we know that it is difficult to use only one kind of IP traceback method for the Internet. This chapter proposes the operational architecture of an IP traceback method that can solve these issues.

On designing the architecture, we modeled our method after the routing architecture in the Internet. The Internet is not operated with one routing protocol to manage the network routing tables. It is a hierarchy of EGP (Exterior Gateway Protocol) and IGP (Interior Gateway Protocol) according to the scale of the network. EGP is used for routing between ASes. In the EGP operation, the routing information has been exchanged through interconnections based on the agreement between ASes. Applying the mechanism of the above-mentioned path control to the traceback mechanism was attempted. This is "Hierarchical Architecture for IP traceback" is based on the idea of "Hierarchy" in the routing system.

3.3 Hierarchical architecture for IP traceback

The proposed architecture is made up of a construct with three components: eIP traceback, iIP traceback, and ITM network. Each control area of eIP and iIP traceback is the same as each control area of EGP and IGP, which are the routing control protocols shown in Fig. 3.2. ITM network is used for the association of eIP and iIP. We describe the purpose of each component as follows:

By definition, "Exterior IP (eIP) traceback

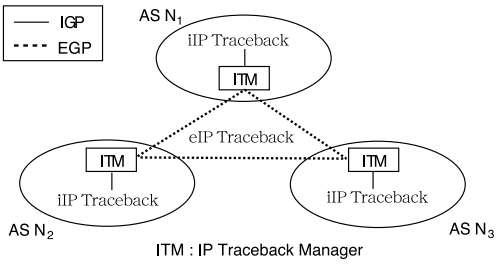

Fig. 3.2. Hierarchical architecture for IP traceback

architecture" designates each AS that the attack flow passed. eIP traceback should have the capability for finding attack paths within 30 minutes. It has been shown that the initial attacking stage ends within 30 minutes, according to the report of CAIDA[CAIDA]. However, in existing research, there is no IP traceback method known that can specify each AS that the attack flow passed. We propose the "IP Option Traceback" for eIP traceback and describe it in section 3.3.1.

"Interior IP(iIP) traceback architecture" designates the router's IP address that the attack flow has passed in the AS. We researched an existing IP traceback and defined 3 parameters that are needed to make relationships between eIP and iIP. These are the packet dumps of the attack flow, the timestamp, and the recording node (AS/IP). In this chapter, we only discuss how to use modified IP traceback as iIP traceback. The technologies of each existing IP traceback are not discussed.

The "IP traceback Manager (ITM)" that exists on each AS exchanges information for the eIP/iIP traceback operation. ITMs are connected with each other and use ITM Protocol (ITMP).

The IP traceback process using this proposed technique is described as follows:

1. Victim's administrator requests a countermeasure to the attack flow to the administrator on a connected network (AS). The administrator of the AS does the monitoring and records the attack flow to the victim's node.
2. eIP traceback, in cooperation with ITM network, calculates the attack path of ASes that the attack flow passed by using this record.
3. ASes in the attack path executes the countermeasure: filtering and bandwidth shaping of the attack flow for easing immediate damage at the victim's site.
4. iIP traceback is executed on each AS in which the attack node has specified the true IP address of the attack node.
5. The DoS attack ends because the attack node is shut off from the network.

Each AS can select a different method of IP traceback because iIP traceback can be executed independently in each AS. We can switch to another method when vulnerability of an IP traceback that is in operation has been discovered.

3.3.1 IP option traceback

We propose "IP option traceback (IP-OPT)" as eIP traceback that uses the IP option header of IPv4 and the destination option header of IPv6. We considered the influence of Internet traffic in the passive detection packet that used the IP option. We make a mathematical model and analyze our "IP Option traceback." For more details, please refer to [217, 219, 254]. In this chapter, the outline of IP-OPT is described.

IP-OPT has two components, TOG (Traceback Option Generator) and Packet Monitor (PM). TOG generates the IP traceback option packet constructed with information to construct the attack path by the passive detection method. Also, TOG constructs the attack path from the tracing packets.

PM is set up on each BGP area border router on the AS, and has the following two functions:

Packet selection: The packet output from the interface on the router is selected at probability P, and the copy of the selected packet is recorded.

Cooperation with TOG: PM receives and uses probability P from TOG and sends the selected packet to TOG.

Next, the function of TOG is described. TOG is constructed with the following function.

1. Generation of the IP traceback option: TOG generates IP traceback option from the selected packet and the parameter (AS number and HASH information), and sends it.
2. The option management: TOG generates the "AS message key X" and "key identification number Z", and stores them.
3. The association of PM: TOG sends probability P to each PM and receives selected packets from each PM.
4. Verification of packet: TOG verifies the HMAC certified data and constructs of the attack path.

5. The association between ITMs: For construction of the attack path, TOG exchanges an attack path or IP options with neighboring TOGs via the ITM network. The TOG requests the execution of iIP traceback at each AS including attack nodes via the ITM network.

The traceback information that TOG generates from parameters is recorded in the "IP destination option header" for IPv6 and the "IP option header" for IPv4[54, 231]. It has the following parameters:

HMAC tag number (16 bit): An HMAC algorithm identifier used in HMAC (Keyed-Hash Message Authentication Code)[162].

Key identification number (64 bit): This is the Key identification number Z to AS message key X that is used with HMAC.

MAC data (algorithm dependence and variable-length): HMAC authentication data H generated from message key X to AS number and AS.

When we construct attack paths, the victim records the IP options O from attack flows. Each AS's TOG verifies the AS's MAC data $HMAC_{AS}$, which is calculated from the AS message key X_n and identification number Z, and MAC data in O. We find out the attack path from the concentration of ASes that are verified O.

3.3.2 ITM and module API

This section describes ITM and ITMP that are used to exchange data between ITMs. ITM network is constructed with a peering connection of ITM to neighboring ITMs as well as the peering of EGP (Fig. 3.3). Thus, the ITM network topology is the same as the AS network topology.

ITMP has an authentication phase and a connected phase. The authentication phase authenticates ITM and exchanges neighbor information (AS number), supporting eIP/iIP traceback under neighbor ASes, and so on. After the authentication phase, comes the connected phase. In the connected phase, each eIP/iIP traceback can transmit data to each other.

In our implementation, eIP and iIP traceback

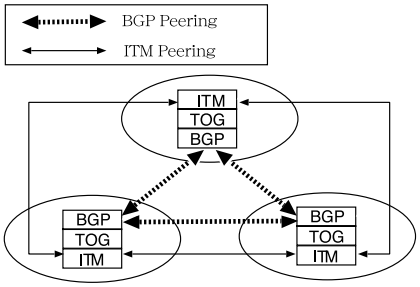

Fig. 3.3. ITM network : interconnection between ITMs

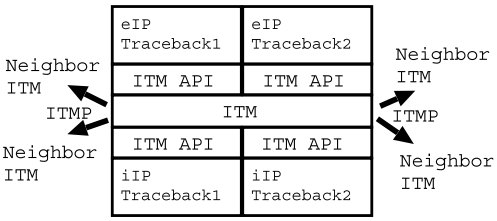

Fig. 3.4. Structure of ITM

systems under AS are connected with ITM through ITM API (Fig. 3.4). We define ITM API from the analysis of the existing implementations, PAFFI as iIP traceback and IP-OPT as eIP traceback. PAFFI, developed by Yokogawa Electric is based on Hash-based IP traceback and uses NP (Network Processor). ITM API is the access to information of neighboring ITM (kind of the AS number, the connectivity, the state, and the supported IP traceback), the data transfers between modules of neighboring ITMs, and status reports of each module.

3.4 Feasibility of deployment

In the achievement of this proposal, ITM operation is required in core AS groups that CAIDA selects from the operation result of Skitter[201].

The authors believe that the feasibility of the ITM network is high. ISPs should do the countermeasure to Denial of Service attack as soon as possible because it results in the least amount of damage that the guest receives.

However, ISPs now use link inspection methods for IP traceback. The cost for this is high and the ISP is spending a lot of money, time and human resources, to stop attack flows. This proposal can reduce the cost because ITM automates the coop-

eration between ASes. Moreover, each AS can select a technique of IP traceback depending on the operation scale and budget of ISP. Therefore, the feasibility of the ITM network is high because there is the advantage that the AS operates ITM.

3.5 Conclusion and future work

We proposed a hierarchical IP traceback architecture and described the feasibility of this proposal on the Internet. The IP traceback technology is designed as one of the countermeasures to the DDoS attack. However, IP traceback requires the cooperation between two or more ASes over a long period of time. The attacker can then analyze the weak points of the countermeasure for developing an anti-IP traceback attack. We believe that it is difficult to use only one kind of IP traceback method for the Internet.

This proposal was split into eIP traceback and iIP traceback according to the relation between EGP and IGP in the routing control architecture. eIP traceback detects ASes that the attack flow has passed. iIP traceback detects the router's IP address passed by the attack flow. We described an implementation architecture for "IP option traceback" that was proposed as eIP traceback. We investigated existing IP traceback and described implementation architecture of the ITM network which was used to make an association between eIP and iIP traceback.

We plan to brush up our implementation and carry out experiments on StarBED[282] which has 500 physical nodes (5000 VM-simulated node) and is a fully programmable Internet simulator. We will then present the experimental results of this simulation.

第 4 章 A Layer-2 Extension to Hash-based IP Traceback

4.1 Introduction

Distributed Denial of Service attacks (DDoS attacks) are widely reported on the Internet, and the damage caused by these attacks is becoming more serious day by day. DDoS attacks overwhelm networks and server systems with an onslaught of data, until they cannot be used. An attack is the result of multiple data flows from thousands of attacker nodes distributed widely on the Internet. To terminate a DDoS attack, all attacker nodes must be detected and isolated.

Unfortunately, the anonymous nature of the IP protocol makes it difficult to accurately identify the IP address of an attacker node, if the attacker wishes to conceal it. IP traceback is one of the techniques used to identify the true source of an attack by reconstructing the path of the attacking flow.

Our research group has proposed a hierarchical architecture of IP traceback[218], which is composed of two parts: interdomain IP traceback and intradomain IP traceback. Interdomain IP traceback identifies an Autonomous System (AS) from which the attacking flow has come, whereas, intradomain IP traceback detects the attacker node inside an AS.

Essentially, the amount of the attacking flow traffic around an attacker node is smaller than that around a victim node. Intradomain IP traceback must detect an attacker node from a small amount of traffic generated by attacking flows.

Hash-based IP traceback[265] is able to trace a single packet. In this technique, each router audits all packets forwarded, and stores *packet digests* instead of the packets themselves. Packet digests are stored in *digest tables*, which are bitmaps based on a space- and time-efficient data structure known as a Bloom filter[15]. Digest tables reduce the required storage size by several orders of magnitude over current log-based techniques[249]. To trace an attacking packet, the attack path is reconstructed by checking digest tables on each router. Hash-based IP traceback is suitable for identifying an attacker on intradomain networks because of its ability to trace a single packet.

An attack path determined by hash-based IP traceback indicates the ingress point of an attacking packet. However, the ingress point is the nearest router of the attacker node on a network, not the attacker node itself, because of limitations of the algorithm used to store packet digests. The technique thus cannot identify the attacker node itself on the subnet.

In this chapter, we propose a layer-2 extension to hash-based IP traceback. To narrow down the existence range of an attacker node, and eventually to detect the attacker node itself, the attacking packet has to be traced on the layer-2 (L2) network under the ingress point router. In our extension, leaf routers on the intradomain network store L2 information corresponding to a packet digest. If an attacking packet passes through an L2 switch, the recorded L2 information distinguishes the ingress port of the packet on the switch. However, holding L2 information with one-to-one correspondence to a packet digest requires a large amount of storage space.

To reduce storage requirements, two different digest tables are provided. One is an *L3 digest table* which stores packet digests themselves, and the other is an *L2 digest table* which records L2 information corresponding to packet digests. These tables permit cooperation between layer-3 (IP) traceback and layer-2 traceback, and enable more detailed tracing of attacker nodes.

The rest of this chapter is organized as follows: In Section 4.2, we describe related work. Section 4.3 describes our proposed extension to hash-based IP traceback. We discuss the issues involved in the implementation of our extension in Section 4.4. Finally, we conclude our proposal and mention future work in Section 4.6.

4.2 Related Work

4.2.1 Intradomain IP Traceback on a Hierarchical Architecture

Intradomain IP traceback is a part of a hierarchical architecture for IP traceback [218]. The aim of intradomain IP traceback is to detect attacker nodes themselves inside an AS. The intradomain IP traceback process starts when a request from a victim AS for tracing is received via interdomain IP traceback.

Techniques for intradomain IP traceback should be able to trace attacker nodes even when the size of the attacking flow is small, because the traffic of attacking flows around an attacker node is usually smaller than the traffic of attacking flows around the victim node. Therefore, probabilistic techniques such as iTrace[13] or the marking approach[253] are not suitable for intradomain IP traceback because they are based on large amounts of traffic generated by a DDoS attack.

Furthermore, an effective technique for intradomain IP traceback should have the ability to trace attacker nodes even if the request comes from the victim AS after the initial stage of a DDoS attack has finished and the attack has become latent; hence, the link-testing method such as CenterTrack [272], specifying the attacking flow by filtering functions or monitoring functions on each router, is not sufficient for intradomain IP traceback because such methods are useful only while the attacking flows are flooding the Internet.

4.2.2 Hash-Based IP Traceback

Snoeren et al.[265] have proposed hash-based IP traceback and Source Path Isolation Engine (SPIE) architecture, which is an implementation of hash-based IP traceback. SPIE uses auditing techniques to support the traceback of individual packets, while reducing the storage requirements by several orders of magnitude over current log-based techniques[249].

In SPIE architecture, a Data Generation Agent (DGA) on each router stores packet digests, which are 32-bit hash values, instead of packets themselves. An input value for hash functions to make packet digests, here called the *packet signature*, is a masked IP header plus the first 8 bytes of payload (Fig. 4.1). A packet signature uniquely represents an IP packet and enables the identification of the packet across hops in the forwarding

path.

To reduce storage requirements, the DGA stores sets of packet digests in a space-efficient data structure known as a Bloom filter[15]. The Bloom filter computes k distinct packet digests for each packet, using independent uniform hash functions, and stores the n-bit results as indices in a digest table, a 2^n-sized bit array constructed by the Bloom filter. The bit array of a digest table is initialized to all zeros, and bits are set to one as packets are received. Fig. 4.2 shows a Bloom filter with k functions. A digest table is dispatched when it is full or when a certain time interval for refreshing hash functions (refreshing time) is reached. The dispatched digest table is stored in a ring buffer on the router, with a time stamp and a set of used hash functions. To avoid hash collisions, each router selects a new set of k distinct hash functions at the refreshing time.

The traceback process starts at the 1-hop upper router of a victim node. First, the DGA on a SPIE-enhanced router computes the k hash digests on the packet in question and checks the indicated bit positions of the appropriate digest table. If any one of them is zero, the packet was not forwarded by the router. However, if all the bits are one, it is highly likely that the packet passed through the router. When it appears that the router forwarded the suspected packet, neighbor routers' digest tables are then tested. This process continues until all branches of the attack graph are checked. Fig. 4.3 shows an example of this traceback processing. Tracing proceeds from R_1 (the victim's neighbor router) towards the attacker node, A. Black arrows represent the path of the attacking packet, and gray arrows represent the reconstructed attack graph, respectively. The ingress point of the attack is R_{10}, this is the nearest router to the attacker A. When an attacker node is in the SPIE-enabled network, the ingress point of an attacking packet is a leaf router within the network. Otherwise, a border router of the SPIE-enabled network is the ingress point of the attack.

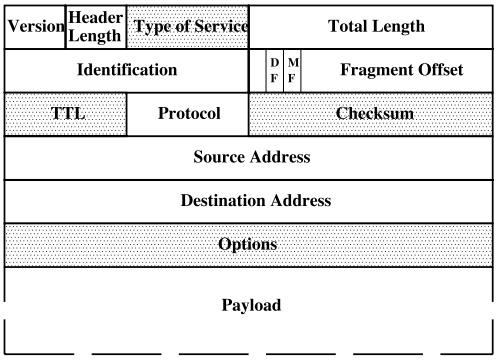

Fig. 4.1. Packet Signature

The false positive rate of a digest is given by several parameters. When a Bloom filter takes m bits of memory, which can store n packets, the effective false-positive rate with k digest functions can be expressed as

$$P = \left(1 - \left(1 - \frac{1}{m}\right)^{kn}\right)^k \approx (1 - e^{-kn/m})^k$$

Tables are available which provide the effective false-positive rates for various capacities and the number of digesting functions[79]. The size of a Bloom filter is defined by a capacity factor c ($= m/n$); that is, to store n packets requires a Bloom

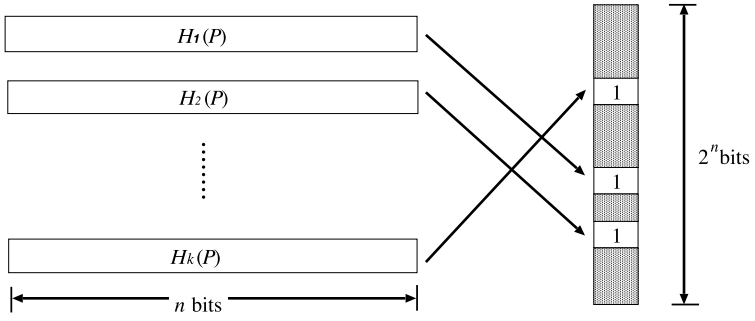

Fig. 4.2. Bloom Filter

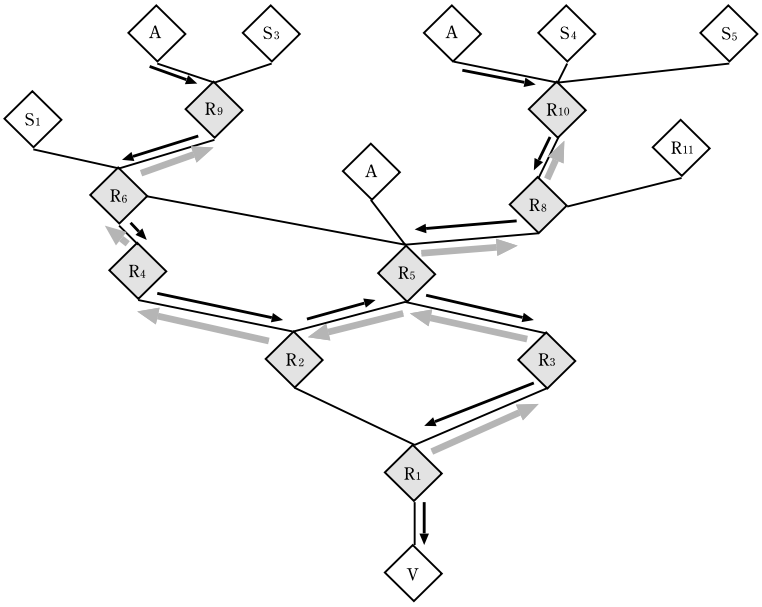

Fig. 4.3. SPIE traceback process

filter of size $m = cn$. If a 2M byte ($= 2^{24}$ bit)-sized digest table is taken with a capacity factor of five), it can store roughly 3.2M packets. Considering the trade-off between the false positive rate and the storage requirements, Snoeren et al.[265] have proposed using a Bloom filter with three digesting functions ($k = 3$) and a capacity factor of five ($c = 5$) in practice. In this case, assuming the average size of a packet is approximately 1000 bits, SPIE requires roughly 0.5% of the total link capacity in digest table storage with a rate of 0.092 false positives.

4.2.3 Limitations of Hash-Based IP Traceback

The ingress point detected by SPIE is either a border router of the SPIE system or the leaf router which has the attacker node on its subnet, not the attacker node itself. Examining digest tables of a leaf router never reveals which node on a subnet is the attacker node, because a digest table shows only whether or not the router forwarded the packet in question.

Specifying only an ingress point leaf router is not sufficient to stop a DDoS attack. Filtering, based on the destination (victim's) IP address, TCP ports, and so on, blocks not only attacking packets but also packets sent from other (normal) nodes; hence, such filtering denies other nodes' communications or services. Leaving attacker nodes on subnets permits later DDoS attacks. To terminate current attacking flows from a network without blocking other nodes' communications, and to prevent other DDoS attacks in the future, attacker nodes themselves must be discovered and physically removed from the network.

Tracing attacking packets on an L2 network, however, is difficult. If the subnet under the leaf router is widely spread, as in an xDSL network or in a Metro-ether network, it requires human resources and a lot of time to identify the attacker node and remove it from the subnet network. Therefore, to identify the attacker node itself, some method is required to trace the attacker node on the subnet L2 network under the leaf router.

Theoretically, if hash checkers like DGA are set against all nodes on a subnet, they can identify the true attacker node. However, from an operational point of view, this is not a realistic approach.

In this chapter, we propose a layer-2 extension of SPIE which narrows down the existence range

of an attacker node on a subnet L2 network, while reducing other storage requirements for the extension.

4.3 A Layer-2 Extension to Hash-based IP Traceback

4.3.1 Limitations of tracing based on MAC Address

Each Network Interface Card (NIC) has a globally unique address, the Media Access Control (MAC) address, in which the upper 24 bits comprise the vendor code and the lower 24 bits comprise an address assigned by the vendor. IEEE, the official global authority, is responsible for handing out blocks of MAC addresses. On Ethernet networks, which constitute today's local area networks, a packet is forwarded on the basis of MAC addresses.

The source MAC address of an incoming packet is available for tracing the attacker node on the subnet of a leaf router. The source MAC address of an incoming packet is the MAC address assigned to the device of the sender node. Moreover, a switch maintains a database of all MAC addresses received on all of its ports, and uses database entries to decide whether a packet frame should be forwarded or filtered. This database is called an L2 forwarding database (FDB).

Each FDB entry consists of the MAC address of the device, an identifier for the port on which it was received, and an identifier for the VLAN to which the device belongs. The switch can learn FDB entries; that is, the system of the switch updates its FDB with the source MAC address from a packet, the VLAN, and the port identifier on which the source packet is received. Looking up the FDB on a switch tells us the port on which a source MAC address has been learned. Whether or not a source MAC address is spoofed, the FDB temporarily records the source MAC address paired with the incoming port. By searching the port which learned the source MAC address of an attacking packet, and tracing switches hop-by-hop, we can detect the location of the attacker node or the port to which the attacker node directly connects.

However, large amounts of storage on the leaf router are required to store source MAC addresses which correspond to packet digests.

On the other hand, the FDB cannot learn more than a certain number of MAC addresses, because the available memory size for the FDB is limited. To prevent the FDB from becoming full, the FDB has an "aging time". Entries in the FDB are removed (aged out) if, after a period of time (the aging time), the device has not transmitted. When an attacker spoofs the source MAC address of an attacking packet, and the spoofed MAC address has been "aged out" from the FDB, there is no way to trace the attacker node by using stored MAC addresses.

In our extension, instead of storing MAC addresses themselves, we use two identifiers and a forwarding database of the layer-2 (L2) switches to trace the attacker node on an L2 network. The two identifiers show the subnet on which an attacker is located, and specify the incoming port on a particular slot of the L2 switch that constitutes the L2 network of the subnet.

4.3.2 Assumptions

Before describing the details of our extension to hash-based IP traceback, we define several assumptions:

- SPIE has detected an ingress point on the intradomain network, that is, a leaf router;
- MAC addresses are used to resolve hosts on subnets under the leaf router;
- All switches along the path from the leaf router to the attacker node support Bridge-MIB[52];
- The source IP address and the source MAC address of a packet may be spoofed;

The first assumption is made because we use SPIE to identify attacker nodes on an intradomain network, and because we focus on tracing an attacker node on an L2 network after detecting the leaf router by SPIE. The second and third

assumptions reflect the fact that our extension is based on MAC addresses and FDBs of L2 switches, and that it uses Bridge-MIB to obtain FDBs from switches. Bridge-MIB is widely supported by today's L2 switches. In the fourth assumption, we anticipate the ability of an attacker.

4.3.3 Identifiers

We use two identifiers instead of storing MAC addresses. Using two conversion tables, we detect the port of a switch through which the attacking packet entered. The two identifiers are as follows:

- NI-ID ($8bit$): a network identifier assigned to either the MAC address of a network interface on a leaf router, or the *VLAN ID* of a virtual interface if the leaf router uses VLAN interfaces.
- Port-ID ($12bit$): a port identifier assigned to a port of a switch. Each port has a different Port-ID.

A network interface identifier (NI-ID) is assigned to each MAC address of the network interface card. The NI-ID specifies which subnet an attacking packet comes from. Currently, a router can equip 802.1Q VLAN interfaces, which means that multiple virtual interfaces can be used on one physical network interface. Although VLAN interfaces, which share one physical interface, have the same MAC address, each VLAN interface has a different *VLAN ID*, so that a router distinguishes VLAN interfaces by their *VLAN IDs*. If a network interface is a VLAN interface, a NI-ID is assigned to its *VLAN ID* instead of the MAC address shared with another VLAN interface.

A port identifier (Port-ID) is assigned to a port on a particular slot of an L2 switch through which a leaf router directly connects. A Port-ID tells us through which port of an L2 switch an attacking packet came in.

A combination of an NI-ID and a Port-ID distinguishes a particular port from other ports if several network interfaces of a leaf router connect to the same L2 switch.

An 8-bit NI-ID, expressed in the range from 0 to 255, is sufficient to represent the network interfaces of a leaf router. A 12-bit Port-ID, whose range is 0 to 4095, is sufficient to express all ports of a current switch.

4.3.4 Conversion Tables

We use two conversion tables: the NI-ID table, and the Port-MAC table.

A leaf router keeps an NI-ID table that is based on the interface configuration. Each NI-ID table entry consists of an NI-ID and the MAC address of a network interface card assigned with the NI-ID. If VLAN interfaces are used, then the entry in the NI-ID table consists of an NI-ID and a *VLAN ID*. Table 4.1 shows the NI-ID table of a leaf router which uses VLAN interfaces on a physical interface whose MAC address is "C".

A Port-MAC table, constructed by the FDB of an L2 switch that connects to the leaf router directly, is used to obtain the Port-ID from the source MAC address of a packet which enters the leaf router via the L2 switch (Table 4.2). The leaf router obtains the contents of the FDB via Bridge-MIB[52]. Bridge-MIB is widely supported by current switches. An entry in the Port-MAC table is composed of a source MAC address and a Port-ID. The source MAC address is that of the incoming packet, and the Port-ID identifies the L2 switch's port that has learned the source MAC address. Recording Port-IDs instead of source

Table 4.1. NI-ID Table

NI-ID	MAC address	VLAN ID
1	A	
2	B	
3	C	101
4	C	102

Table 4.2. Port-MAC Table

NI-ID	Port-ID	MAC address
1	2	F
	3	E
	4	H
	4	K

```
[ recording identifiers in L2DT procedure ]

receive an incoming frames

extract the source MAC address M_s
    and the Destination MAC address M_d from a frame
extract the packet signature PS from an packet
refer to NI-ID table
for (number of entries on the NI-ID table){
  extract an entry NE_i from the NI-ID table
  compare NE_i.ether_addr to M_d
  if (NE_i ≡ M_d){
    N_id = NE_i.id
    break the loop
  }
}
refer to a PortMAC table assigned to the Nid
for (number of entries on the PortMAC table){
  extract an entry PE_i from the PortMAC table
  compare PE_i.ether_addr to M_s
  if (PE_i ≡ M_s){
    P_id = PE_i.id
    break the loop
  }
}
make input values I by concatenating PS, N_id, and P_id
compute hash values H(I)_i   (1 ≤ i ≤ k)
for (number of hash values){
  search the bit indicated by H(I)_i on L2DT
  if (the bit ≡ 0){
    set the bit to 1 (that is, store identifiers on L2DT)
  } else {
    hash collision occurs, increment counter of duplication
  }
}
```

Fig. 4.4. Procedure to store identifiers in L2DT

MAC addresses, and specifying the incoming port of L2 switches by recorded Port-IDs, enables us to trace an attacking packet even when the source MAC address has aged out from the FDB of an L2 switch.

Port-MAC tables are generated for each NI-ID. This indicates that an NI-ID and Port-ID pair identifies the port in which the attacking packet came, even if several network interfaces of a leaf router connect to the same L2 switch. Table 4.2 shows the Port-MAC table paired with NI-ID "1". Two entries with the Port-ID "4" represent two source MAC addresses ("H" and "K") which have been learned on the L2 switch port whose Port-ID is "4".

4.3.5 Algorithms of the Layer-2 Extension

For each incoming packet, extended DGA (xDGA) on the leaf router searches identifiers from conversion tables corresponding to an incoming packet. To reduce memory requirements for storage of detected identifiers corresponding to the packet, xDGA uses an L2 digest table, another Bloom filter for storing identifiers corresponding to the packet. To use L2 digest tables, we provide a storing algorithm and a detecting algorithm for Bloom filters and for L2 digest tables. The former is to encode identifiers into L2 digest tables, and the latter to decode identifiers from L2 digest tables.

```
[ extracting identifiers in L2DT procedure ]

receive a suspected packet transferred from Traceback Manager

extract the pakcet signature PS from the suspected packet
check the time span through the suspected packet
  on the leaf router
search appropriate time L2DTes from ring-buffer for L2DT
for (number of L2DT to test){
  for (number of MAX_NIID_NUMBER){
    for (number of MAX_PORTID_NUMBER){
      make a hash input I by concatnating PS,
        an NI-ID N_i, and a Port-ID P_i
      for (each hash function (1 ≤ i ≤ k)){
        compute hash values H(I)_i
      }
      for (each hash value H(I)_i){
        search the bit indicated by H(I)_i on L2DT
        if (the bit ≡ 0){
          break the loop
        } else if (all bits indicated by hash values are 1){
          N_find = N_i, P_find = P_i
          write down N_find and P_find in syslog
          warn as finding the route of the suspected packet
          show the incoming interface N_find
            and the port P_find
          exit the procedure
        }
      }
      if (all Port-ID has been tested){
        break the loop
      }
    }
    if (all NI-ID has been tested){
      break the loop
    }
  }
  if (all suspected L2DT has been checked ){
    the suspected packet was not stored into any L2DT
    log the warning message about failure of L2 traceback
    exit the procedure
  }
}
```

Fig. 4.5. Procedure to check identifiers on L2DT

4.3.5.1 Algorithm for Storing Identifiers

Fig. 4.4 shows an algorithm to check identifiers against each incoming packet and to store detected identifiers in an L2 digest table.

For an incoming packet, xDGA extracts source/destination MAC addresses from the Ethernet frame, and pulls up the packet signature from the packet. xDGA runs a process of storing packet signatures into L3 digest tables, while running a process to store identifiers in L2 digest tables.

In the process to store identifiers, xDGA first refers to the NI-ID table to search for an entry whose MAC address is the same value as the extracted destination MAC address of the packet, and to get the NI-ID assigned to the detected MAC address entry. Next, xDGA takes a Port-MAC table corresponding to the NI-ID, and

checks entries which have the same MAC address as the extracted source MAC address. If an entry is detected, the Port-ID of the entry represents the incoming port of the packet on an L2 switch. If not, xDGA sets the Port-ID to the value of NOT_FOUND_PORTID.

xDGA makes a hash input by concatenating the packet signature and detected identifiers, and computes hash values using hash functions for the Bloom filter of an L2 digest table. The bit positions on the L2 digest table, which are indicated by those hash values, are set to 1. This ensures that the pair of identifiers is stored in the L2 digest table corresponding to the packet.

4.3.5.2 Algorithm for Traceback Processing

The algorithm to derive two identifiers from L2 digest tables in the traceback process is shown in Fig. 4.5.

When an xDGA leaf router receives a suspected packet from the traceback manager STM, xDGA first generates the packet signature from the suspected packet. xDGA runs the IP traceback process and L2 traceback process (process to detect identifiers) in parallel. In the L2 traceback process, xDGA pulls up L2 digest tables which have time stamps indicating when the suspected packet might have come through the leaf router, and checks each L2 digest table to establish whether it has a record of identifiers with the suspected packet's signature. The test of an L2 digest table is as follows: xDGA chooses an NI-ID and Port-ID pair, and generates a hash input to test the packet signature of the suspected packet and the pair of identifiers. Then, xDGA computes hash values of each hash function, and examines each bit position pointed by hash values on the L2 digest table. If all the bits are 1, the ingress interface on the leaf router and the ingress port on an L2 switch are detected; that is, the suspected packet came through the interface with the NI-ID and through the L2 switch's port with the Port-ID. If any one of the bits is 0, the suspected packet was not stored with the pair of identifiers. Then, xDGA generates hash values by another pair of identifiers and checks the L2 digest table again. If all combinations of identifiers have been tested, but the true pair could not be detected in any trial, the suspected packet was not stored in the L2 digest table. Then, if another L2 digest table is suspected to record the packet signature in question, xDGA tests it.

To reduce the storage requirements, xDGA stores two identifiers in the L2 Bloom filter, though the running time of our searching algorithm is $O(n^2)$.

4.4 Implementation

We have implemented a PC-based prototype of our proposed extension, based on an open source code of hash-based IP traceback, spie-1.0.90[11], on FreeBSD-4.3.

We have added two components in SPIE and extended DGA to deal with L2 digest tables (Fig. 4.6). Fdbgetter is an agent which fetches FDB entries from L2 switches directly connected to the leaf router via SNMP and makes conversion tables. x_spiemem, a loadable kernel module, converts MAC addresses of an incoming packet to two identifiers and stores these identifiers in an L2 digest table. The packet storing processes by spiemem, which is a loadable kernel module to record packet signatures for the purpose of IP traceback, and the processes of recording identifiers by x_spiemem are independent of each other. Therefore, the Bloom filter's parameters for L2 digest tables and L3 digest tables can be set independently to achieve a different size or false positive rate.

For hash functions of L2 digest tables, we reuse MD5, which SPIE computes for L3 digest tables. Therefore, different hash functions are simulated by a salt value seeded on the leaf router. The salt values for L2 digest tables are generated independently from the salt value for L3 digest tables.

xDGA on the leaf router checks L2 digest tables while testing L3 digest tables for IP traceback in parallel. The processes of examining L2 digest

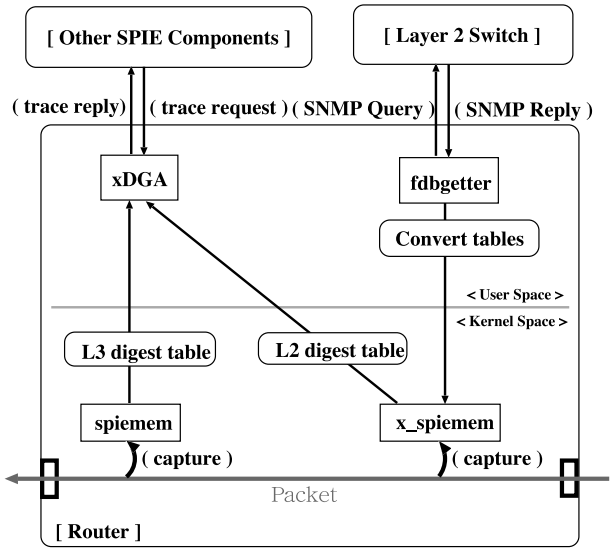

Fig. 4.6. Components of the Layer-2 extension of SPIE

tables do not affect processes of SPIE's IP traceback because these processes are independent of each other.

The information from L2 digest tables is significant for only the administrator of the leaf router and should not be available to anyone else. Therefore, xDGA records the result of checking the L2 digest table in a log file and, shows a warning message on the standard output of the leaf router. xDGA does not send the information of identifiers to SCAR and/or STM.

4.5 Preliminary Evaluation

4.5.1 Memory Requirements

Our extension needs memory spaces for L2/L3 digest tables and for conversion tables. The required memory space for L2 digest tables can be characterized by the duration of record storage and the accuracy of L2 digest tables measured by the number of false positives. Meanwhile, the required memory space for L3 digest tables can be determined by these parameters for L3 digest tables. On the other hand, the required memory spaces for conversion tables are dominated by the number of network interfaces of a leaf router and affected by the sum of the number of FDB entries fetched from L2 switches connected to the leaf router.

4.5.1.1 Memory requirements of the conversion table

In our implementation, an entry in the NI-ID table needs 16 bytes and an entry in a PortMAC table requires 12 bytes. The total memory size for conversion tables is dominated by the number of NI-ID table and the sum of FDB entries on each L2 switch. If a leaf router has three network interfaces, and each interface aggregates a class C (/24) subnet, the total memory size for conversion tables is a maximum of 9192 bytes.

Consider the case when a leaf router connects a BlackDiamond, which is a today's High-end Layer-2 switch developed by Extreme Networks. A BlackDiamond can learn 256,000 entries on its FDB. Therefore, a PortMAC table for a BlackDiamond consumes a maximum of 3 MB memory space. However, only rarely will the entry space of an FDB be full; the average size of the required memory for a PortMAC table will be much smaller.

4.5.1.2 Memory requirements of the L2 digest table

The memory requirements for an L2 digest table are the same as those for an L3 digest table. The available memory size m with a capacity of storing n packets and a false positive rate P can be

Table 4.3. Capacity of a Bloom filter

memory size	max num of entries	average bit rate
8 kB (2^{16} bit)	13,107	1 Mbps
128 kB (2^{20} bit)	209,715	20 Mbps
2 MB (2^{24} bit)	3,355,443	335 Mbps
32 MB (2^{28} bit)	53,687,091	5.3 Gbps
512 MB (2^{32} bit)	858,993,459	8.5 Gbps

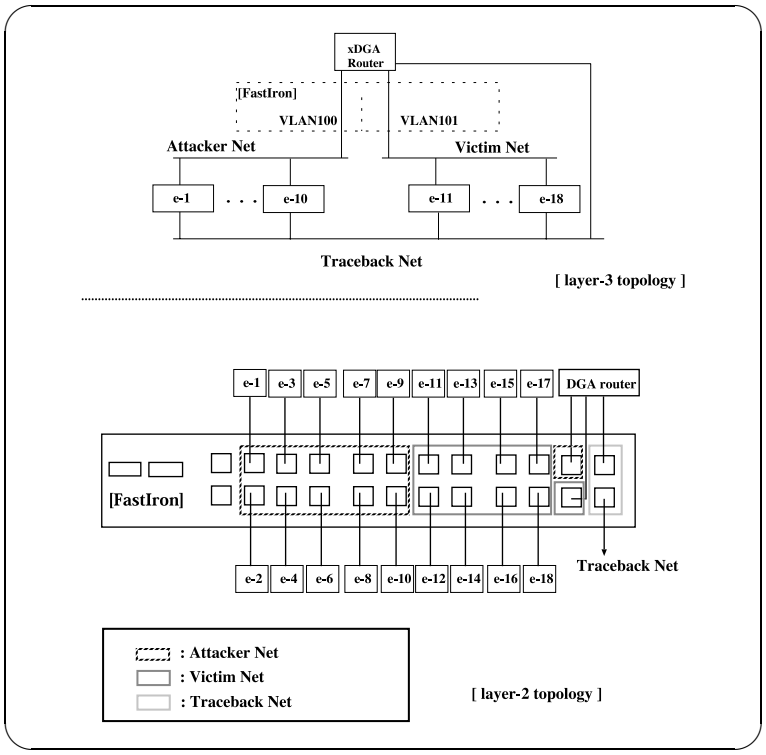

Fig. 4.7. Testbed topology

determined as

$$m = -n \cdot \log(1/P)/\ln(1/2) \approx 1.44n \cdot \log(1/P).$$

A 2^{24} bits (2 MB)-sized L2 digest table with a false positive rate of 0.092 can store 3,355,443 pairs of identifiers. Table 4.3 shows typical memory sizes and their capacity to store entries, when the number of hash functions is 3 and the false positive rate is 0.092. In Table 4.3, we compute the average bit rate stored in a Bloom filter when the average packet size is 1000 bits and the length of time required to use a Bloom filter is 10 seconds.

4.5.2 Preliminary Experiment

4.5.2.1 Experiment environment

We have evaluated our implementation on a testbed network. Fig. 4.7 shows the L3/L2 network topology of the testbed.

We carried out the experiment in the following situation: only one xDGA leaf router, and a traceback manager STM.

On the Victim Net, there were two victim nodes (monitor agents): one received attacking packets from e4 (connected to port 6 on FastIron) and e5 (connected to port 7) on the Attacker Net, and the other received attacking packets from e6 (connected to port 8) and e8 (connected port 10) on the Attacker Net. Attacking packets were TCP SYN packets transferred to the victim's TCP port 7777, generated by a customized SYNflood program based on the libnet sample SYNflood program[255]. Each attacking packet had a spoofed

Table 4.5. Parameters of Bloom Filters

size of hash values	24 bits
size of digest table	2 MB (2^{24} bits)
length of time to refresh digest table	10 sec.
number of digest tables held on ring buffer	6
number of hash functions	3
capacity factor of Bloom filter	5

Table 4.4. Time interval of each attacker node

name	interval
e4	3 (sec.)
e5	5 (sec.)
e6	7 (sec.)
e8	11 (sec.)

source IP address and a spoofed source TCP port. A sample monitor agent of spie-1.0.90 sent trace requests about all packets received by snort 1.9.0 beta4[26]. In this experiment, the snort on each victim node was set as a host IDS to catch only attacking packets on the victim node so that SPIE components traced only attacking packets. Each attacker node sent attacking packets at different time intervals (Table 4.4).

In testbed topology, we separated the network for exchanging traceback queries from networks to exchange normal/attacking traffic; that is, Traceback Net (Fig. 4.7) transmitted only traceback messages.

The specification of the xDGA router is as follows: Pentium III 800 MHz processor, 128 MB sized memory, and four physical network interfaces. To build L2 networks, we used an L2 switch, Foundry's FastIron, firmware version 7.1.26T10. Each network to exchange normal/attacking traffic is connected to the xDGA router.

Parameters of the Bloom filter are described in Table 4.5. In this experiment, we set the values of parameters for L2 digest tables to the same set of parameters for L3 digest tables (Parameters of Bloom filters for L2 digest tables can be set to different values from parameters for L3 digest tables). Through calculation of parameters shown in Table 4.5, each L2/L3 digest table could store about 3.3 million packets at a 0.092 false positive rate, and could hold each digest table for 1 minute.

Table 4.6. Time spent to check a digest table

time	L3	L2
max (micro sec.)	70564	68474
min (micro sec.)	297	333
average (micro sec.)	381	1124
median (micro sec.)	333	1067

The time interval to fetch FDB entries from the L2 switch was set to 60 seconds.

4.5.2.2 Experimental result

In this experiment, each attacker node sent 2000 attacking packets to the victim node, so STM and xDGA dealt with 8,000 packets altogether. As a result, all attacking packets were found on the appropriate L3 and L2 digest tables.

Table 4.6 shows the time spent in finding the record of the suspected attacking packet on the appropriate digest table. Each maximum time value is highly exceptional. In most cases, the time spent to find a packet record is within the range from 297 to 1704 microseconds on the L3 digest table, from 333 to 2733 microseconds on the L2 digest table. Comparing the results for L3 and L2 digest tables, it is found that there is no critical performance overhead for searching for an appropriate ID pair on an L2 digest table in this experiment.

4.6 Conclusion and Future Work

We have proposed a layer-2 extension to hash-based IP traceback, which stores the L2 information of incoming packets on each leaf router. Using the L2 information, the port on a particular slot of the L2 switch that learned the incoming attacking packet can be detected, which leads to a narrowing down of the existence range of an attacker node on an L2 network. Preliminary

evaluation of a prototype implementation shows the potential usability of layer-2 traceback.

In this chapter, we have proposed an extension approach of SPIE for the first step of an L2 traceback technique, and we have developed an algorithm to reduce the memory requirements for storing identifiers based on Bloom filters. The preliminary evaluation shows that all traceback trials found appropriate ID pairs with no critical overhead on the system in a small topology. In future work, we will evaluate our extension in a large topology, while looking for a more space- and time-efficient algorithm.

Our extension essentially employs the FDB information of an L2 switch. The contents of an FDB are changed when a new entry is made or when the oldest entry is aged out. Therefore, the Port-MAC table should be renewed when FDB entries have changed. However, dynamical synchronization by SNMP would cause some overhead on traffic or on a CPU of both a leaf router and an L2 switch. There is a trade-off between the accuracy of the Port-MAC table and the overheads on the system. We will evaluate this trade-off in future work.