情報処理学会論文誌　コンピューティングシステム　Vol.5　No.1　13–26 (Jan. 2012)

ネットワーク侵入検知システムの協調による性能と耐障害性の向上

花岡 美幸[1,†1]　河野 健二[1,2,a)]

受付日 2011年5月11日, 採録日 2011年10月13日

概要：ネットワーク経由の攻撃の検知および防御に，ネットワーク侵入検知・防御システム（NIDS/NIPS）が広く用いられている．しかし，汎用 PC を用いた NIDS が多い中，ネットワークの高速化や攻撃の巧妙化にともなう性能低下や，NIDS 障害によって攻撃を見逃す耐障害性の問題がある．本論文では，組織内に複数設置された NIDS を相互に協調させることで，性能および耐障害性を向上させる手法を提案する．1カ所に高価なマシンや並列に動作する複数のマシンを置くのではなく，他の NIDS の負荷やルール設定をもとに，通常時は過負荷のマシンや冗長なルール設定を減らすように再設定をすることで性能向上を，障害時は他の NIDS が検査を肩代わりすることで耐障害性向上を目指す．実験では，冗長なルール設定が除去され負荷が分散されることで 10%以上の性能向上を，障害時に攻撃検知を他の NIDS が行うことで耐障害性向上ができることを確認した．

キーワード：ネットワークセキュリティ，ネットワーク侵入検知システム

Coordinating Configurations of Network Intrusion Detection Systems for Performance and Fault-tolerance

Miyuki Hanaoka[1,†1]　Kenji Kono[1,2,a)]

Received: May 11, 2011, Accepted: October 13, 2011

Abstract: Network intrusion detection/prevention systems (NIDS/NIPS) are widely used for detecting or preventing network-based attacks. However, there are two issues of current NIDS implementation: performance degradation because of today's increased traffic volume and sophisticated attacks, and fault-tolerance, which means a NIDS failure causes some packets to be unchecked. In this paper, we propose Brownie, a system for performance improvement and fault-tolerant via collaboration between already-existing NIDSs in an organization, instead of installing one expensive hardware or parallel NIDS at a vantage point. Through exchanging their load status and configurations, Brownie improves performance by offloading overloaded NIDS and eliminating redundant rules, and improves fault-tolerance by enabling rules once checked by the failed NIDS so that the other NIDS(s) takes over the failed NIDS. The experimental results with a web server benchmark suggest that Brownie increases the benchmark throughput by more than 10%. They also suggest that detection by a failed NIDS are took over by other NIDSs with Brownie.

Keywords: network security, network intrusion detection systems

1. はじめに

ネットワーク経由の攻撃の検知および防御にネットワーク侵入検知・防御システム（NIDS/NIPS: Network Intrusion Detection/Prevention System）が広く用いられている．特に，ネットワークパケットと攻撃検知のルール（シグネチャ）とのマッチングを行うことで攻撃を検知するシ

[1] 慶應義塾大学
Keio University, Yokohama, Kanagawa 223–8522, Japan
[2] 科学技術振興機構 CREST
CREST, Japan Science and Technology Agency, Chiyoda, Tokyo 102–0075, Japan
[†1] 現在，株式会社日立製作所中央研究所
Presently with Central Research Laboratory, Hitachi, Ltd.
[a)] kono@ics.keio.ac.jp

グネチャ型 NIDS が広く普及している．企業や大学など組織のネットワークに対する攻撃は日常のものとなっており，日々のセキュリティ対策として NIDS/NIPS が果たす役割は大きい．現在，オープンソースや商用で数多くの NIDS が提供されているが，その多くが高価でない汎用 PC をもとに作られている．そのため，NIDS の運用において，次の 2 つの問題がある．

1 つ目は性能の問題である．汎用 PC をもとに作られた NIDS ではすべてのトラフィックを監視するのは難しくなっている．この理由として，ネットワークトラフィックの量や速度の増加が CPU の高速化に勝っていること，攻撃が複雑化しているために以前よりもトラフィックを細かく検査する必要があることなどがあげられる．また，攻撃を防ぐために，パケットの検査が終わるまでそのパケットを止める必要がある NIPS は，NIDS に比べて性能に与える影響が大きい．NIPS が過負荷になれば，その NIPS の配下にあるネットワーク性能が低下する．また，NIDS が過負荷になった場合，パケットの取りこぼしが起き，攻撃を検知できない可能性もある．

2 つ目の問題は耐障害性である．ここでいう耐障害性とは，ある NIDS に障害が起こったときに，攻撃の検知や防御ができるかということを指す．NIDS の障害の原因としては，ハードウェアの故障や NIDS 自身に対する攻撃などが考えられる．通常，NIDS に障害が起こると，その NIDS が検査していたルールは素通りとなり，攻撃の検知はできなくなる．

性能向上や耐障害性向上に対する対策として，複数の NIDS を用いる手法がある．複数の NIDS を用いて，トラフィックの検査を複数のマシンで行うことで，処理を分散し，性能を向上させることができる．また，複数 NIDS 間で処理を冗長化させることで，ある NIDS が障害により攻撃の検知ができなくなっても，別の NIDS で同じ検査処理を行うことで，耐障害性を向上させることができる．しかし，実際に複数の NIDS を 1 カ所に導入するのは容易ではない．多くのマシンを新たに導入するため，多くのコストがかかるためである．マシンの購入費だけでなく，それらを適切に設定する必要もある．また，多数のマシンの設置場所の確保や，十分な電源を供給する必要がある，という問題もある．

そこで，本論文では，組織ネットワーク内に複数の場所に置かれた NIDS どうしを連携させることで，性能向上や耐障害性向上を行う手法を提案する．我々の着目点は，大学や企業など多くの組織では，組織内ネットワークとインターネットの境界だけでなく，内側のネットワークの様々な位置や階層に複数の NIDS を設置している場合が多い，ということである．たとえば大学では，大学の入り口と外のネットワークとの境界以外にも，各学部や各研究室で個別に NIDS を置いていることがある．本論文では，これらの同じネットワーク内にある複数の NIDS が協調することで，性能向上や耐障害性向上ができることを示す．提案システムでは，ネットワークの経路上にある他の NIDS と相互にルール設定や負荷の状況をやりとりしながら，各 NIDS の設定を協調して最適化していく．ある 1 点に置かれた NIDS の性能・耐障害性向上ではなく，複数の NIDS による組織内全体の性能・耐障害性向上を目指す．

ここで，性能と耐障害性は基本的には，トレードオフの関係となる．性能向上のためには処理の分散化が，耐障害性向上のためには処理の冗長化が必要だからである．そのため，本機構では，管理者が性能重視や耐障害性重視の設定を柔軟に選択できるようにした．性能向上を最優先する場合には，NIDS が過負荷にならないように複数の NIDS 間で完全に処理を振り分け，負荷分散を行う．この場合，処理の冗長性を取り除いてしまうため，耐障害性は低下するが，本機構により他の NIDS が肩代わりすることで，短い時間で攻撃の検知を再び行えるようにする．一方，耐障害性を最優先する場合は，NIDS の処理を冗長化するという手法をとる．極端にいえば，すべての NIDS ですべて同じ処理を行えば，完全冗長となり，耐障害性は上がる．しかしこの場合，すべての処理を有効にすることで，NIDS は過負荷になりやすくなり，性能は低下する．これらの中間として，処理を部分的に冗長化させることで，一定の耐障害性を保ちつつ，性能低下を抑える設定もありうる．たとえば，よく攻撃を検知するルールや危険性の高いルールのみを冗長させる，という方法がある．

本システムの有効性を示すために，プロトタイプを実装し，実験を行った．web ベンチマークを用いた実験では，性能重視の設定を行った場合，複数 NIDS のルールの再設定を行うことで，ベンチマーク・スループットが 10%以上向上した．また，NIDS を意図的に停止させ障害を起こした実験では，90 秒程度で別の NIDS でルールが有効になり攻撃が検知されるようになった．90 秒という長さは，設定可能な値であるが，管理者が障害に気付いて対処するよりは十分早いといえる．

本論文の構成は次のとおりである．まず 2 章で関連研究について述べる．次に 3 章で性能向上と耐障害性向上の各手法の概要を述べ，これらのトレードオフについて議論する．4 章では設計と実装，5 章で実験を示し，6 章でまとめる．

2. 関連研究

複数の NIDS を協調させる研究は，主に 2 つのカテゴリに分けられる．1 つ目はネットワーク上に多くの NIDS を設置することで，検知精度を向上させることを目的とした研究である [1]．ネットワーク上（組織内だけでなくインターネット上も含まれる）の様々な場所に NIDS を分散して置き，それぞれの NIDS で発生した警告を収集すること

で，1台のNIDSでは検知できない攻撃を検知する [2], [3]．たとえば，DOMINO [2] ではネットワーク上に分散したNIDSが，互いにP2Pプロトコルで NIDSの警告を交換する．他のNIDSで検出した警告を用いることにより，1台のNIDSで1つのローカルネットワークを監視しているだけでは検知不可能な，インターネット規模の攻撃を検知することができる．さらに，収集した攻撃や警告をもとに新しい検知ルールを作成し，各NIDSに配布することによって，新たな攻撃を検知する手法も提案されている [4]．本研究は，検知精度ではなく，性能や耐障害性向上を目的として組織内のNIDSを協調させることを提案している．

2つ目は，並列に動作する複数のマシンを用いてNIDSを構成することで，性能を向上させることを目的とした研究である．独立に動作する複数のNIDSを協調させるというよりは，複数台のマシンを用いて1つの高性能なNIDSを構成し，組織ネットワーク内の1つの監視点に置くという手法である．たとえば，NIDS Cluster [5]，Active Splitter [6]，Kruegel らによるもの [7] などがある．これらは，基本的にセンサと呼ばれるトラフィックの検査を行う複数のマシンと，センサにトラフィックを割り当てるフロントエンドから構成されている．トラフィック検査を行うセンサを複数置き，処理を分散させることによって，1台のマシンよりも性能を向上させることができる．Kruegel ら [7] は，高速かつ高精度な検査を可能にするために，3層構造のフロントエンドを提案している．Active Splitter [6] は，性能向上を目的としたフロントエンドの最適化手法を提案し，NIDS Cluster [5] は，センサどうしのやりとりを強化したシステムを提案している．しかし，これらのシステムを導入するためのコストは少なくない．管理者は，複数のマシンを購入し，設置や設定をし，管理しなければならない．多くのマシンを設置するのに必要な場所や電源の確保が容易ではない場合もある．本研究は，すでに組織内の様々な場所に置かれているNIDSを利用し，それらを協調させるという，新しいアプローチを提案している．なお，これらの研究は性能向上を目的としており，耐障害性については触れられていない．しかし，すべてのセンサが同じルールを持ち，フロントエンドがトラフィックを割り振るという構成から，センサの耐障害性はあると考えられる．フロントエンドが，障害が起こったセンサにトラフィックを割り振らないようにすればよいからである．しかし，フロントエンドが単一障害点になっており，フロントエンドに障害が起こった場合，NIDS全体に障害が起きてしまう．

複数のマシンを用いず，1台のNIDSの性能を向上させる手法は，様々な観点から長年研究されている．シグネチャとのマッチングアルゴリズムでは，Aho-Corasick [8]，Wu-Manber [9] などのパターンマッチングや，正規表現マッチング [10] の高速化手法が提案されている．また，ネットワークプロセッサ [11], [12], [13] やFPGA [14], [15], [16], [17]，マルチコアプロセッサ [18]，グラフィックプロセッサ [19], [20], [21] といったハードウェアを用いたシステムも提案されている．これらの手法は提案手法と補完的であり，各手法を用いたNIDSを協調させることでさらに性能向上や耐障害性のトレードオフを選択できるようになる．

一方，NIDSの耐障害性に関する研究はあまり行われていない [22]．Kuang らは，攻撃に対する耐性を考慮したNIDSを提案している [23]．複数の攻撃検知部とその動作を定期的に監視する監視用エージェントを用いた構成となっており，監視用のエージェントが攻撃検知部やマシンの障害を検知すると，攻撃検知部の複製を別のマシンで起動させることで，攻撃検知を継続させる．Siqueira らは，エージェント型のNIDSにおいて，監視用エージェントが各NIDSの動作状態を監視し，停止したNIDSを復元する手法を提案している [24]．しかし，これらのシステムは，各提案手法に合致したアーキテクチャのNIDSを採用していなければならない．我々は，広く使われているNIDSを対象とし，組織内ネットワークの木構造型のトポロジを利用して，NIDSどうしが監視し合い障害時の代替をするシステムを提案している．

3. 提案

3.1 概要

NIDSの運用において，性能と耐障害性はともに重要な要素である．本論文での耐障害性とは，NIDSに障害が起こっても，攻撃の検知や防御ができることをいう．多くのNIDSが汎用PCをもとに作られている現在，十分な性能・耐障害性が提供できていない．性能低下の原因は，ネットワークトラフィックの量や速度が急激に増加していることや，攻撃が複雑化しているために詳細な検査が必要となっていることなどがあげられる．NIDSの性能が低下すれば，監視下にあるネットワーク性能の低下や攻撃の見逃しが起こりうる．耐障害性においては，ハードウェアの故障や攻撃などの原因でNIDSに障害が発生する．通常，NIDSに障害が起こると，そのNIDSが検査していたルールは素通りとなり攻撃の検知はできなくなる．

本論文では，組織内ネットワーク内の複数の場所に置かれたNIDSどうしを連携させることで，性能向上や耐障害性向上を行う手法を提案する．本手法では，同じネットワーク内のNIDSどうしがそれぞれの負荷状況やルール設定を交換し合い，適切に再設定していく．同じネットワーク内に設置されたNIDSであるため，NIDSどうしは相互に信頼して連携を行うことができる．通常時はNIDSどうしの負荷を分散させることで性能向上を，障害時は障害が発生したNIDSの攻撃検知を他のNIDSで代替させることで耐障害性向上を行う．従来のように，1カ所に並列に動作する複数のNIDSを置くのではなく，すでにネットワー

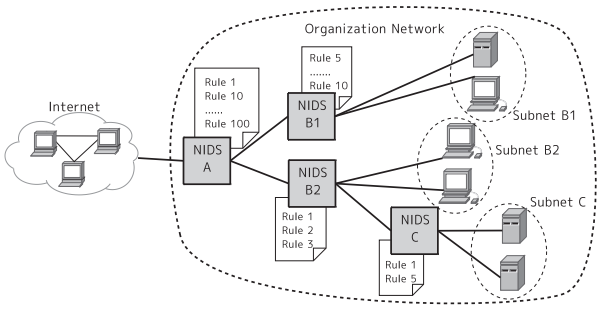


図 1　NIDS の設置例
Fig. 1　Example setting of NIDSs.

ク内にある NIDS を活用することでネットワーク内全体の性能向上と耐障害性向上を目指す．

我々の着目点は，大学や会社など多くの組織では，インターネットと組織のネットワークの間に置かれる NIDS 以外にも，組織内ネットワークの様々な場所に NIDS が置かれていることが多くある，ということである．すなわち**図 1** のように，組織内ネットワークの入り口に置かれた NIDS A 以外にも，その下流に，各部門や課によって独自の NIDS が置かれることがある．たとえば大学では，学科や研究室でそれぞれ NIDS を設置するといったことが考えられる．図 1 の例では，NIDS B1 と B2 は学科によって，NIDS C は研究室によって設置された，と考えられる．本手法では，各 NIDS はそれぞれの 1 つ上流の NIDS（親 NIDS）とすぐ下流の NIDS（子 NIDS）と通信をする．たとえば，NIDS A は NIDS B1 および B2 と，NIDS B2 は NIDS A および C とやりとりをする．

以下，性能向上と耐障害性向上の各手法について述べた後，性能向上と耐障害性の間に本質的に存在するトレードオフについて議論する．

3.2　性能向上

提案手法では，他の NIDS と協調し，1) 過負荷になった NIDS の負荷を減少させ，2) NIDS 間の冗長なルール設定を削除する，の 2 つのアプローチをとることにより性能向上を目指す．2 つのアプローチにより，負荷が NIDS 間で分散され，効能向上を達成することができる．以下，それぞれのアプローチについて述べる．

3.2.1　過負荷 NIDS の負荷の減少

ある NIDS が過負荷になると，過負荷になった NIDS がネットワーク全体のボトルネックとなり性能低下の要因となる．NIDS が過負荷になる要因としては，大量のトラフィックを受け取った，多くのルールに対して検査しなければならない，マシンが低性能であるなどがあげられる．組織内に置かれた複数の NIDS のうち，すべてが過負荷になるという状況はほとんどないが，1 つの NIDS が過負荷になると，その NIDS が全体のボトルネックになる．そこで，過負荷になった NIDS の負荷の一部を，ネットワーク経路上にある他の低負荷である NIDS に分散することで，過負荷になった NIDS の負荷を減らす．これにより，過負荷 NIDS がボトルネックになることを防ぐことができ，結果としてスループットの向上や通信遅延の減少が期待できる．

負荷を分散させるために，提案システムでは，過負荷な NIDS から低負荷な NIDS にルールを移譲する．すなわち，過負荷な NIDS でいくつかのルールを無効にし，代わりに低負荷な NIDS で同じルールを有効にする．トラフィックの量やマシンの性能をすぐに変更することは難しいため，ルールの数を減らすことで負荷を減らす．たとえば図 1 では，NIDS A に多くのルール設定がされており，過負荷であるとする．一方，その下流にある NIDS B1，B2 は設定されているルール数が少なく，低負荷である状況を考える．このとき，NIDS A は NIDS B1，B2 に，たとえばルール 70〜100 を移譲する．つまり，ルール 70〜100 を NIDS A で無効にし，代わりに NIDS B1，B2 で有効にする．こうすることで，NIDS A の負荷の一部を NIDS B1，B2 に移すことができる．

ここで，ルールを移譲した後も，元のセキュリティレベルは維持していることに注意してほしい．すなわち，ルールの移譲前と移譲後で，各パケットに適用されるルールは変わらない．たとえば，組織外からのネットワークについて考えると，提案手法では，元々ルールを有効にしていた過負荷 NIDS を通る組織外からのすべてのトラフィックは，ルールを移譲した先の NIDS のどれかを必ず通る．そのため，組織外からのトラフィックは，宛先マシンに届く前に，移譲されたルールに対して必ずチェックされることが保証されている．たとえば，前述の例のようにルールが上流から下流の NIDS に移譲される場合を考える．これは，図 1 において NIDS A から NIDS B1 と B2 にルールを移譲したときである．このとき，上流 NIDS（NIDS A）を通る組織外からのすべてのパケットは，下流 NIDS のどれか 1 つ（NIDS B1 か B2）を通ることになる．すなわち，組織外からのすべてのパケットは下流 NIDS のどれか 1 つ（NIDS B1 か B2）で，移譲されたルールに対してチェックされることになる．ここでは，組織外からのネットワークトラフィックについて述べたが，本システムでは組織内のネットワークトラフィックも考慮して，すべてのパケットが移譲前と同じルールに適用されることを保証することにより，セキュリティレベルの維持を行っている．セキュリティレベル維持についてのさらに詳しい議論は，4.1.1.3 で述べる．

3.2.2　重複ルールの除去

あるネットワーク経路上に置かれた複数の NIDS が，同じルールを重複して有効にしていることは，珍しいことではない．これは，すべての NIDS が同じ管理者によって管理されているわけではなく，各 NIDS がそれぞれ別々の管理者によって管理されているためである．たとえば前述

の大学の例では，大学の入り口に置かれたNIDSは大学のネットワークの管理者によって管理されるが，学科のNIDSは学科のネットワーク管理者によって管理される．ネットワーク経路上に置かれた複数のNIDSに同じルールが設定されていると，その経路を通るパケットは同じルールに対して何度もチェックされることになる．重複しているルールをなくし，ネットワーク経路上のどこか1カ所だけで検査すれば，通信遅延を少なくすることができると考えられる．ここで，前項と同様に，重複したルールをなくすことで，セキュリティレベルが低下することはない．ルールの無効化は，ネットワーク経路上の他のNIDSでも同じルールが有効にしており，そのNIDSでチェックされるときのみ行うからである．

たとえば図1では，NIDS A，B2，Cの3つのNIDSでルール1を有効にしている．そのため，NIDS C以下にあるマシン宛のパケットは，ルール1に対して3回検査されることになる．そこで，たとえばNIDS B2とCではルール1の設定を無効にして，NIDS Aのみでルール1について検査するようにすれば，その分通信遅延やマシンの負荷を減らすことができる．また，NIDS B2とCでルール1を無効にすることで，セキュリティが低下することはない．なぜなら，NIDS B2とCに届く組織外からのトラフィックは，NIDS Aですでにルール1に対してチェックされているからである．

3.3　耐障害性

NIDSに障害が発生すると，通常，障害が発生したNIDSが検査していたトラフィックは素通りとなり，攻撃の検知ができなくなる．NIDSの障害の原因としては，通常のサーバなどと同様に，ハードウェア故障やソフトウェアバグ，設定のミスに起因する原因がある．また，NIDSを停止させればトラフィックが無検査になることから，攻撃による障害も考えられる[25]．そこで，本手法では障害が起こったNIDSで有効にしていたルールを別のNIDSで有効にすることで攻撃検知を代替させる．障害の検知は，定期的に送り合っている負荷情報をハートビートとして用いることで行うことができる．一定時間ハートビートが送信されてこなければ，何らかの原因によりそのNIDSに障害が起こったと考えられる．そして，障害が起こったNIDSが有効にしていたルールを，自NIDSで有効にする．各NIDSは，自分の上流・下流のNIDSのルール設定を交換しているため，障害が起こったNIDSでどのルールを有効にしていたかは容易に知ることができる．

たとえば図1でNIDS Cが何らかの障害により動作を停止した場合を考える[*1]．障害が起こる前は，サブネットCのトラフィックについて，NIDS Cでルール1と5の検査を行い，攻撃検知をすることができた．NIDS Cに障害が起こった場合，NIDS Cで検査をしていたルール5について，トラフィックの検査はされなくなってしまう．そのため，ルール5で検知できる攻撃の検知・防御ができなくなる．そこで，上流NIDSであるNIDS B2でルール5を有効にし，サブネットCへのトラフィックを検査することで，NIDS Cの代わりに攻撃の検知を行う．ここで，NIDS B2とCは互いに有効にしているルール設定を交換しているため，NIDS B2はNIDS Cで有効にしていたルールを容易に知ることができる．また，NIDS B2とCは定期的に負荷情報を交換しているため，これをハートビートとして用いることで，NIDS B2はNIDS Cの障害を検知することができる．

3.4　性能向上と耐障害性向上のトレードオフ

本論文では，NIDSの協調による性能向上と耐障害性向上の手法を提案している．しかし，性能と耐障害性は簡単には両立できるものではない．なぜなら，これらは基本的に相反するものだからである．高性能を得るためには，できるだけ処理が分散されるようにしなければならない．一方，耐障害性を得るためには，処理を冗長にしてどれかに障害が起きたときも他のマシンで代替できるようにしておかなければならない．

そのため，本機構では，管理者が性能重視や耐障害性重視の設定を柔軟に選択できるようにした．管理者は，性能と耐障害性のトレードオフを考慮し，自組織に合致した設定を行う．たとえば，耐障害性を最優先した設定をした場合，すべてのNIDSで同じルールを有効にすることで完全冗長とする．これは，前述した性能向上や耐障害性向上の手法を動作させることはせず，NIDSどうしがルール設定の交換を行った時点で，相手のNIDSが有効にしているルールすべてを自NIDSでも有効にすることで行う．これにより，あるNIDSが障害で停止しても，すぐに他のNIDSで同じルールに対して検査され，攻撃を検知することができる．しかし，すべてのNIDSで完全冗長のルール設定を行うため，性能は低下する恐れがある．

一方，性能を最優先した設定をした場合，本機構の性能向上手法により，ルールの再配置と冗長ルールの削除が行われる．その結果，負荷分散され性能向上ができる．しかし，あるNIDSが障害で停止すると，一時的に障害が起こったNIDSで有効になっていたルールについて検査されなくなる．本機構では，耐障害性向上の手法により，このルールを他のNIDSで検査することで耐障害性向上を行う．しかし，耐障害性を最優先にしたときの完全冗長に比べると，検査されない期間ができてしまう．ただし，本機構がない場合，管理者が障害を検知して対策を行うまでの時間，検査されないルールがあるということに注意してほ

[*1] なお，ここでは分かりやすさのために，3.2節で述べたルールの移譲や削除を考えず，図1のルール設定の状態で障害が起こったと仮定する．

しい．本機構の耐障害性向上手法による検査復帰までの時間は，NIDS どうしのハートビートの間隔と障害検知の閾値によって変わるが，通常管理者が対策を終了するまでの時間と比べれば十分短いといえる．

また，これらの中間の設定として，冗長度を制限したり，冗長化するルールを重要度が高いものだけに限定したりするなどで，性能と耐障害性のトレードオフを考慮した設定をすることができる．

3.5　本手法導入による影響の可能性

この節では，本手法導入によるシステムへの悪影響の可能性について議論する．たとえば，可用性の問題である．一般に分散システムなどでは，システムが連携し複雑になると，システムの可用性が低くなる可能性がある．しかし，本手法を導入した場合においても，ルール設定を参照しながら，ネットワークトラフィックを検査する部分は個々に動作する．そのため，NIDS の動作に障害が起こった場合においても，他の NIDS にその障害が伝搬することはない．一方，ルール設定に関しては，互いに連携してルール設定を行うため，他の NIDS のルール設定の影響を受ける．たとえば，悪意のある NIDS が存在した場合には，負荷の高いルールを多数有効にしておくことで，他の NIDS が過負荷になる可能性がある．しかし本システムは，組織内での NIDS どうしの協調であるため，互いの NIDS どうしは信頼してよく，悪意のある NIDS は存在しないと仮定できる．ネットワークや NIDS の管理者は，直接接続している上流・下流のネットワーク構成については熟知していると考えられ，認証などの方法により相互に信頼できる NIDS とのみ接続することは難しくない．

4.　設計と実装

NIDS 協調によって性能向上や耐障害性向上ができることを示すため，NIDS 協調システム Brownie を実装した．各 Brownie は各 NIDS を管理し，他の Brownie と通信を行い，それぞれが管理する NIDS を適切に設定することによって，NIDS どうしの協調を実現させる．以降，特に紛らわしい場合を除いて，Brownie はシステム全体および各 NIDS を管理する各インスタンス双方を指すのに用いる．

NIDS を Brownie とともに動作させるには，管理者は親 NIDS の IP アドレスなどの情報を Brownie に与える．NIDS の起動時，対応する Brownie は親 NIDS とともに動作している Brownie とコネクションを確立し，以降それぞれの負荷状況やルール設定を定期的に交換する．障害検出にはこれをハートビートとして用いる．また，コネクション確立時に，それぞれの NIDS で有効または無効にしているルール設定を交換する．

NIDS としては，広く普及しているオープンソース NIDS である Snort [26] を用いた．現在の実装ではすべての NIDS が Snort であるが，将来的には様々な NIDS が動作している環境にも適用できるようにしたいと考えている．もう 1 つのオープンソース NIDS である Bro [27] は，動的ルール書き換えやセンサ間の通信機構を備えているという点で，より本システムに向いていると考えられ，今後用いる NIDS の候補として考えられる．

4.1　性能向上

提案手法では，他の NIDS と協調し，1) 過負荷になった NIDS の負荷を減少させるオフローディング，2) NIDS 間の冗長なルール設定を削除する，の 2 つのアプローチをとることにより性能向上を行う．

4.1.1　過負荷 NIDS の負荷を減少

NIDS の負荷を減少させるオフロード手順の基本的な考え方は，自分と子 NIDS の負荷を比較し，負荷が高い方から低い方にルールを移譲する，ということである．つまり，自分の NIDS の負荷が，どの子 NIDS の負荷よりも高い場合，いくつかのルールを子 NIDS に移譲する．逆に自分の NIDS の負荷がどの子 NIDS の負荷よりも低い場合，いくつかのルールを子 NIDS から自分の NIDS に移譲する．これを，過負荷 NIDS の負荷が十分に減少するまで行う．ここで，1) いつオフロードを行うか，2) どのルールを移譲するかについて説明する．

4.1.1.1　オフロードの開始・終了判断

まず，Brownie はオフロードが必要かを判断するために，NIDS の負荷を測る．NIDS の負荷は資源使用量，NIDS では特に CPU 使用率で知ることができる．これは NIDS の処理のほとんどは CPU インテンシブなパターンマッチングであるためであり，CPU 使用率を用いることでマシンの性能が異なる場合でも適切に負荷を知ることができる．

その後，Brownie は測定した CPU 使用率をもとにオフロードが必要かを判断する．過負荷の NIDS の負荷を減少させるという観点から，NIDS が過負荷になったら（CPU 使用率が 100%近くなったら）オフロードを開始し，CPU 使用率がある設定値を下回ったら終了する，という方法が考えられる．しかしこの方法にはいくつかの問題点がある．まず第 1 に，終了させるための CPU 使用率を決定するのが困難である．負荷を適切に減少させるためには小さい値がよいが，小さすぎると今度は逆にルールを移譲された NIDS が過負荷に陥ってしまう．第 2 に，もしすべての NIDS が過負荷だった場合，互いにルールを移譲しあうにもかかわらず過負荷は解消されないという状態が続いてしまう．最後に，もし他の NIDS の負荷が軽いのであれば，NIDS が過負荷に陥る前にオフロードを開始した方がよいと考えられる．

そこで，過負荷の NIDS の負荷を減少させるというよりむしろ，負荷分散を目指す手法を用いた．すなわち，Brownie は自分が管理している NIDS の CPU 使用率と子

NIDS の CPU 使用率が同じくらいになるように調整していく．Brownie は，自分が管理している NIDS の CPU 使用率と子 NIDS の CPU 使用率との間にある程度の差があれば，ルールを移譲し，CPU 使用率の差が小さくなればルール移譲を終了する．上記の方法と違い，すべての NIDS が過負荷であるときは，ルールの移譲は起こらない．また，Brownie は自分が管理する NIDS が過負荷に陥る前に，負荷の軽い他の NIDS にルールを移譲することで負荷を割り振ることができる．NIDS 間の許容 CPU 使用率差をパラメータ *Diff* として定め，実験では 5 と設定した．

具体的な手順は以下のようになる．まず，Brownie は自分が管理する NIDS と子 NIDS の CPU 使用率を T 秒ごとに収集する．自分が管理する NIDS の CPU 使用率を c_{my}，子 NIDS の CPU 使用率を c_i（$0 \leq i < n$，n は子 NIDS の数）とする．もし，$c_{my} - max(c_i) > Diff$（*Diff* は正の数なので，$c_{my} > max(c_i)$）であれば，Brownie は自分の管理する NIDS からすべての子 NIDS にルールを移譲する．もし逆に，$min(c_i) - c_{my} > Diff$（同様に，$min(c_i) > c_{my}$）であれば，Brownie はすべての子 NIDS から自分の管理する NIDS にルールを移譲する．どちらでもない場合，すなわち $min(c_i) - Diff \leq c_{my} \leq max(c_i) + Diff$ であれば，負荷は分散していると考え Brownie はルールの移譲を行わない．パラメータ T は負荷の変化に対する感度と Brownie による CPU 使用率収集のオーバヘッドなどを考慮して設定する．また，この値は，耐障害性向上機能において負荷情報の収集をハートビートとして用いるため，障害の検知速度にも影響する．実験では，30 秒と設定した．

4.1.1.2　移譲ルール選択

効率良くルールを移譲できれば，それだけ早く過負荷 NIDS の負荷が減る．もちろん，Brownie では負荷分散されるまでオフロードを繰り返すため，1 度のルール移譲でオフロードが完了できる必要はない．しかし，何度もルール移譲を繰り返して長い時間がかかるよりは，できるだけ短い時間で負荷分散が完了できる方が好ましい．

最も単純なルールの選択方法は，一定数のルールをランダムに選ぶことである．しかし，この「一定数」を設定するのは困難である．もしこの値が小さすぎれば（たとえば，1）オフロードが完了するまでの時間が長くなりすぎる．逆にこの値が大きすぎれば，負荷の軽かった NIDS が過負荷になり，同じルールを元々過負荷だった NIDS に移譲し戻すということが起きてしまう．

そこで，Brownie は移譲するルールの数を，自分と子 NIDS の CPU 使用率の差に基づいて決め，この数のルールをランダムに選ぶ．負荷の差が大きいときはより多くのルールを移譲することで，早く負荷を分散させる狙いである．もちろん 1 つのルールが与える負荷は同じであるとは限らないが，有効にされているルールが増えれば，負荷が増える可能性は高いと考えることができる．移譲されるルールの数は，*Factor* と CPU 使用率の差との積とした．ここで，*Factor* は設定パラメータであり，実験では 10 とした．具体的には，ルールが自 NIDS から子 NIDS に移譲される場合（すなわち，自 NIDS の CPU 使用率がすべての子 NIDS の CPU 使用率よりも大きい場合）のルール数は $Factor \times (c_{my} - max(c_i))$，逆に子 NIDS から自 NIDS に移譲される場合のルール数は $Factor \times (min(c_i) - c_{my})$ となる．

4.1.1.3　セキュリティの確保

3.2.1 項で簡単に述べたとおり，Brownie がルールを移譲することによって，セキュリティを低下させることはないようにしている．すなわち，Brownie 導入前に検知できた攻撃は，Brownie 導入後でも検知できることを保証している．なお，Brownie 導入前に検知できなかった攻撃は Brownie 導入後でも検知できない．この項では，組織外および組織内からの脅威を考慮したときの，セキュリティ確保の問題について詳細を議論する．**表 1** に示すように，次の 2 軸にそった 4 つのケースに分けて，ルールの移譲後のセキュリティの確保を考える．1 つ目の軸は，ある NIDS に直接接続された下流のマシンが，すべて NIDS であるか，一部は NIDS でないか，である．2 つ目の軸は，ルールの移譲方向が下流 NIDS から上流 NIDS であるか，上流 NIDS から下流 NIDS であるか，である．

ケース 1 では，上流 NIDS の直下にあるマシンはすべて NIDS であり，ルールは上流 NIDS から下流 NIDS に移譲する．図 1 の例で考えると，NIDS A の直下には NIDS のみ（NIDS B1 と B2）が接続されており，NIDS A から NIDS B1 と B2 にルールが移譲する場合である．この場合，上流 NIDS（NIDS A）を通過するすべてのトラフィックは，下流 NIDS のいずれか（NIDS B1 か B2）を通過する．そのため，すべてのトラフィックは移譲されたルールに対して下流 NIDS で検査される．

ケース 2 では，ケース 1 と同様に上流 NIDS の直下にあるマシンはすべて NIDS であるが，ケース 1 とは逆にルールが下流 NIDS から上流 NIDS に移譲する．図 1 の例では，NIDS B2 から A にルールが移譲する場合である．この場合，組織外からの下流 NIDS（NIDS B2）を通るトラフィックは，必ず上流 NIDS（NIDS A）を通過し検査される．しかし，下流 NIDS の下にあるサブネット間の通信については別に考慮しなければならない．たとえば，ルール 3 が NIDS B2 から A に移譲すると，サブネット B2 と C

表 1　セキュリティ確保を考察する 4 つのケース
Table 1　Four cases for considering security.

		ルール移譲の方向	
		上流から下流	下流から上流
下流マシン	すべて NIDS	ケース 1	ケース 2
	一部 NIDS 以外	ケース 3	ケース 4

間のトラフィックはNIDS Aでは検査できない．このトラフィックを検査するために，下流NIDS B2でルール3を保持し，送信元・宛先IPアドレスに基づきサブネットB2とC間のトラフィックに対してのみ，検査をする．通常組織外からのトラフィックは，サブネット間トラフィックに比べて多いと考えられるため，ルール3をサブネット間トラフィックに対して有効にしていてもNIDS B2のオフロードはできると考えられる．

ケース3では，NIDSの下流に通常のマシンと下流NIDSが共存しており，ルールが上流NIDSから下流NIDSに移譲する．図1の例では，サブネットB2内のホストとNIDS CがNIDS B2の直下に接続されており，ルールがNIDS B2からCに移譲する場合である．この場合，ルール3がNIDS B2からCに移譲されたときに，NIDS B2でルール3を無効にすると，サブネットB2内にあるマシン宛のトラフィックはチェックされなくなってしまう．これを防ぐため，ルール3について上流NIDS B2でサブネットB2宛のトラフィックをチェックする．NIDS B2はサブネットC宛のトラフィックはチェックしないため，ルール3が一部トラフィックについて有効となっていても，NIDS B2のオフロードは可能だと考えられる．

ケース4では，ケース3と同様にNIDSの下流に通常のマシンと下流NIDSが共存しているが，ルールが下流NIDSから上流NIDSに移譲する．たとえば，NIDS CからB2にルールを移譲した場合である．このケースは，ケース2と同様に扱うことができる．下流NIDS CはNIDS Cの下にあるマシン間のトラフィック（サブネットC内のトラフィック）のみをチェックするためにルールを有効にしておき，その他のトラフィックはNIDS B2で検査する．

以上をまとめると，下流NIDSから上流NIDSにルールを移譲する際（ケース2および4）には，移譲元の下流NIDSでおいてサブネット間のトラフィック検査を継続する必要がある．また，直下にNIDSでない通常のマシンが接続されている上流NIDSから下流NIDSにルールを移譲する際（ケース3）には直下の通常マシン宛のトラフィック検査を継続する必要がある．このとき，検査をする必要のあるトラフィックの検出は，トラフィックのIPアドレスに基づいて行う．通常，NIDSの設定においては，組織内および組織外のIPアドレスを指定する．たとえば，Snortでは，`HOME_NET`に監視対象となる組織内ネットワークのIPアドレスを，`EXTERNAL_NET`に組織外ネットワークのIPアドレスを設定する．そのため，送信元・送信先IPアドレスがともに`HOME_NET`内にあれば，サブネット間トラフィックと判断できる．また，自NIDSの`HOME_NET`から下流NIDSの`HOME_NET`を除いたアドレスが直下にある通常マシンのIPアドレスとなるので，このIPアドレスと組織外ネットワーク`EXTERNAL_NET`間がそれぞれ送信先・送信元IPアドレスとなっていれば，直下の通常マシン宛のトラフィックと判断できる．現時点ではIPアドレスに基づく検査は手動での設定変更で行うようになっているものの，上記で述べた組織内および組織外のIPアドレスに基づいてルールを書き換えることで自動化は可能である．

4.1.2　冗長なルール設定を削除

冗長なルールの削除はシンプルである．Brownieは，自分が管理しているNIDSと下流のNIDSの両方で有効にされているルールがあると，下流のNIDSでそのルールを無効にする．これによって，冗長なルールはすべて自分が管理しているNIDSでのみ有効となる．上流と下流のNIDSは，それぞれ有効・無効にしているルールを交換しているため，冗長なルールは容易に見つけることができる．また，冗長なルールを下流のNIDSで無効にすることにより，セキュリティが低下することがないのは，3.2.2項で述べたとおりである．組織外からのトラフィックは，ルールを無効にした下流NIDSに届く前に，ルールを有効にしてある上流NIDSによってチェックされている．また，組織内トラフィックは，4.1.1.3で述べたのと同様に，組織内トラフィックのみを検査するようにしておけばよい．

冗長なルールを削除するために，下流NIDSではなく上流NIDSのルールを無効にし，下流NIDSのみで有効にする，という選択も可能である．しかしながら，下流NIDSのみで有効にする場合，上流NIDSのみで有効にする場合と比べて，セキュリティを保つための処理が複雑になる．上流NIDSでルールを無効にすることによって，組織外からのトラフィックが検査されずに届いてしまうサブネットができてしまうためである．たとえば，図1で，NIDS B2でのみルール1を有効にし，NIDS AとCでは無効にして，冗長なルールを削除したときを考える．冗長なルール1が削除される前は，サブネットB1とB2に向けたトラフィックをNIDS Aにおいて検査していた．しかし，NIDS Aでルール1を無効にし，NIDS B2のみでルール1を有効にすると，サブネットB1へのトラフィックはルール1に対していずれの場所でも検査されない．ルール1に対して検査を行うためには，NIDS B2でルール1を有効にしなければならない．一方，上流NIDSでのみ有効にする場合，他のNIDSでルールを有効にする必要はなく，よりシンプルな処理でできる．

なお，この方法では，すべての冗長なルールが上流NIDSでのみ有効となるが，上流NIDSが突然過負荷になるということは考えにくい．なぜならBrownieが動作していない場合でも，これらのルールは上流NIDSで有効にされているからである．それでももし上流NIDSの負荷が高くなった場合には，4.1.1項で述べた手順に従って，上流NIDSの負荷を下流NIDSに移譲すればよい．下流NIDSでは，冗長なルールが無効とされたために負荷が軽減されている可能性が高く，上流NIDSの負荷を引き受けやすくなっていると考えられる．

4.1.3　移譲元 NIDS におけるルール設定変更に対する対応

Brownie では，負荷状況などによってルールを自動的に移譲および削除する．そのため，各 NIDS を管理している組織のポリシ変更などにより，管理者がその NIDS のルール設定を変更しルールを追加・削除した場合には，Brownie は他の NIDS の設定状況によりルール設定を再調整する必要がある．そこで，Brownie では，各 NIDS の現在のルール設定のほかに，Brownie による設定変更が行われる前の本来のルール設定を保持しておく．NIDS の管理者は本来のルール設定を変更を加え，Brownie は追加または削除されたルールについて，下流・上流 NIDS での本来および現在の設定状況を考慮し，必要があれば再設定を行う．以下，ルールが追加・削除された場合について，それぞれの処理を述べる．ルールの変更については，変更前のルールが削除され，変更後のルールが追加されたとして扱う．

ルールが追加された場合の処理は，比較的シンプルである．下流または上流 NIDS の現在のルール設定で追加されたルールが有効になっていた場合，冗長なルール設定となるため，4.1.2 項と同様の処理により冗長なルールの削除を行う．すなわち，下流 NIDS でルールが有効になっていた場合には下流 NIDS で，上流 NIDS でルールが有効になっていた場合には変更された NIDS で，当該ルールを無効にする．下流および上流 NIDS の双方で，該当ルールが現在無効になっている場合には，そのまま当該ルールを有効にすればよい．ルールが追加されたことにより，負荷状況が大きく変化した場合には，4.1.1 項で述べたオフロード手順に従って，ルールの移譲がおこる可能性がある．

一方，ルールが削除された場合には，上流・下流 NIDS における本来のおよび現在の設定状況を考慮して，ルール削除を反映させる必要がある．**表 2** に各設定状況における変更内容を示す．各設定状況において，上段にルールが削除された NIDS の設定変更を，下段に上流・下流の NIDS の設定変更を示している．たとえば，右上は，上流・下流 NIDS の本来の設定では有効になっており，現在の設定では無効になっている場合である．このとき，ルール変更が行われた NIDS での変更前の設定は，本来の設定は有効（削除するためには，有効になっている必要がある），実際の設定でも有効（上流・下流 NIDS では無効であるため）となっている．このルールが削除された場合，変更された NIDS でこのルールを無効にし，上流・下流 NIDS で有効にすることで，本来の設定と矛盾のない設定となる．設定内容が他の場合も同様に，表 2 のとおりに処理することでルール削除に対応することができる．

表 2　ルールが削除された場合の処理：各設定において，上段にルールを削除した NIDS の実際の設定変更を，下段に上流・下流の NIDS の設定変更を示す

Table 2　Rule setting changes for a rule deletion: For each setting, the upper row shows the acrutal setting change of the NIDS, and the lower row shows the setting changes of the up/down-stream NIDS.

		上流・下流 NIDS の現在の設定	
		有効	無効
上流・下流 NIDS の本来の設定	無効	変更なし（無効）	有効 → 無効
		変更なし（有効）	無効 → 有効
	有効	変更なし（無効）	有効 → 無効
		有効 → 無効	変更なし（無効）

4.2　耐障害性向上

障害が起こった NIDS で有効にしていたルールを別の NIDS で有効にすることで攻撃検知を肩代わりする．障害の検知は，定期的に送り合っている負荷情報をハートビートとして用いることで行う．これは 4.1.1 項で述べたもので，過負荷 NIDS の検出および負荷分散のために送り合っているものである．一定時間ハートビートが送信されてこなければ，何らかの原因によりその NIDS に障害が起こったと判断する．そして，障害が起こった NIDS が有効にしていたルールを，自分が管理している NIDS で有効にする．各 NIDS は，自分の上流・下流の NIDS のルール設定を交換しているため，障害が起こった NIDS でどのルールを有効にしていたかは容易に知ることができる．

障害検知の検知速度は，ハートビートの間隔と，障害と判断するまでにハートビートを待つ時間によって決まる．本手法では，定期的に交換している負荷情報をハートビートとして用いているため，負荷情報の交換間隔がハートビートの間隔となる．4.1.1 項で述べたとおり，実験ではこの間隔を 30 秒と設定した．障害検知速度により影響するのは，ハートビートを待つ時間である．この時間が短いほど障害を即座に検知することができるが，短すぎるとハートビートが少し遅延しただけで障害と誤検知してしまう．NIDS やネットワークの負荷により，ハートビートが遅れる可能性は十分考えられる．ハートビートを待つ時間はこれらを考慮して設定する必要があり，実験では 60 秒とした．なお，ハートビートは負荷情報とは別に送り合うことも可能である．この場合，ハートビートのための通信を別にしなければならないため通信のオーバヘッドが増加するが，ハートビートの間隔を負荷分散の間隔とは別に設定できるというメリットがある．

障害が起こった NIDS が再び動作したときは，まず初期動作時と同様にルール設定などを互いに交換する．その後，障害が起こった NIDS の回復だと判明したら，有効にしておいたルールを再び無効に戻す．むろん，障害発生前後のルールやマシンの設定がまったく同じとは限らないため，障害前のルール設定も参照して一部のルールを有効にしたままにする場合や，負荷分散や冗長ルールの削除を再び行う場合もある．

4.3　警告ログの収集

Brownie は自動的にルールを有効・無効にするため，元々ルールを有効にしていた NIDS とは別の NIDS で攻撃検知が警告されることがある．そこで，元来ルールを有効にしていた NIDS の管理者に別の NIDS で出た警告を知らせるため，Brownie ではどのルールがどの NIDS から NIDS に移譲されたかを記憶しておく．NIDS が警告を出したら，その NIDS を管理している Brownie は警告を元々ルールを有効にしていた NIDS の Brownie に転送する．警告を受け取った Brownie は，NIDS の代わりに警告を出す．たとえば，NIDS の警告ログに警告を書き出す．これによって，管理者は実際にはその NIDS で有効にされていない警告についても，警告に気付くことができる．

5.　実験

5.1　実験環境

Brownie により，性能や耐障害性が向上することを示すため，実験を行った．3 台の NIDS，2 台のクライアント，2 台のサーバの計 7 台のマシンを用い，**図 2** に示すように 1 Gbps イーサネットで接続した．上流 NIDS マシンは，2 つの Intel Dual-Core Xeon 2.33 GHz CPU（1 コアのみ有効），2 GB メモリ，250 GB 7200 rpm HDD で構成され，他のマシンは，Pentium 4 2.8 GHz CPU，512 MB メモリ，36 GB 7200 rpm HDD で構成されている．これは，下流 NIDS より上流 NIDS の方が高性能な構成となっている．オペレーティングシステムには Fedora 8（Linux 2.6.24），web サーバには Apache 2.2.8 を使用した．NIDS としては，Snort 2.8.0.1 [26] を inline モードで用い，ルールセットは 2008 年 1 月 28 日現在のものをデフォルトのまま用いた．

5.2　実験結果：性能向上

Brownie による性能向上を示すため，Web サーバベンチマークを用いた実験を行った．上下流 NIDS のルールの初期設定として，1) 上流 NIDS ですべて無効，下流 NIDS でデフォルトすべて有効という設定（DOWN）と，2) 上下流双方の NIDS でデフォルトすべて有効という設定（BOTH），の 2 つの設定で実験を行った．前者は過負荷 NIDS をオフロードすることによる性能向上，後者は冗長ルールを削除することによる性能向上を測定することを目的とした初期設定である．実験では，各 NIDS での有効ルール数，CPU 使用率，およびベンチマークのスループットを 10 秒ごとに計測した．最初の 30 分間は，初期状態の性能を測定するため Brownie は動作させないようにした．ワークロードには WebStone 2.5 を各クライアント上で実行し，それぞれ同時接続数は 10 とした．Apache，WebStone，Snort のその他の設定はデフォルトのままとした．

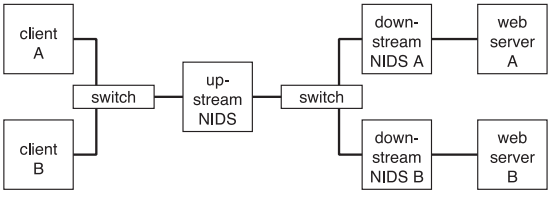


図 2　実験環境

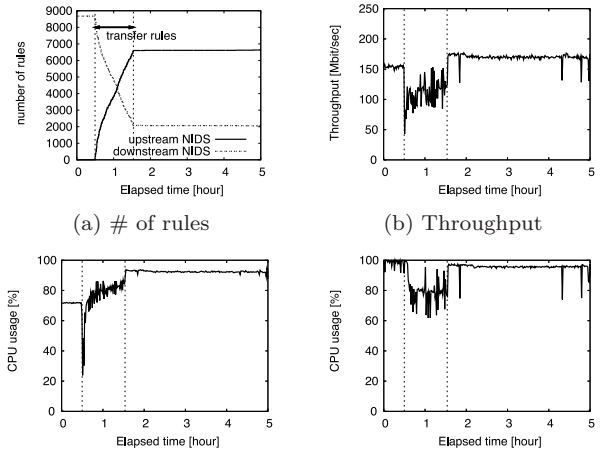
Fig. 2　Experimental network setting.

5.2.1　過負荷 NIDS の負荷軽減

過負荷 NIDS の負荷を削減することによる効果を示すため，NIDS の初期ルール設定を，下流 NIDS はデフォルトルールをすべて有効（ルール数 8,676），上流 NIDS はすべてのルールを無効とした．この設定では，下流 NIDS が過負荷となり，ネットワーク全体のボトルネックとなる．

図 3 に実験結果を示す．グラフ内の 2 つの縦線は，それぞれ Brownie がルール移譲を開始および終了した時刻を示す．図 3 (a) は，各 NIDS で有効にしているルール数を示している．開始 30 分後から Brownie が動作し始め，負荷分散のためのルールの移譲が開始する．下流 NIDS のルール数が減少し，上流 NIDS のルール数が増加していることが分かる．図 3 (c) と (d) に，それぞれ上流 NIDS と下流 NIDS の CPU 使用率を示す．2 つの下流 NIDS の CPU 使用率はほぼ同じであるため，片方のみを載せた．初期ルール設定では，下流 NIDS の CPU 使用率がほぼ 100%に達している一方，上流 NIDS の CPU 使用率は 80%未満となっている．Brownie 動作開始後約 1 時間で，双方の CPU 使用率はほぼ同じとなり，Brownie はルール移譲を停止した．その後の CPU 使用率はすべての NIDS で 90～95%となり，負荷が分散されたことが分かる．

なお，この実験では 4.1.1.3 で述べた IP アドレス制限つきのルール移譲については考慮せず，移譲元では移譲したルールを完全に無効としているが，IP アドレス制限つきのルール移譲を行った場合には，移譲時間が少し長くなる可



(a) # of rules　(b) Throughput

(c) CPU usage of upstream NIDS　(d) CPU usage of downstream NIDS

図 3　実験結果：過負荷 NIDS の負荷軽減（初期設定：DOWN）

Fig. 3　Results of offloading overloaded NIDS (Initial setting: DOWN).

能性がある．一部のトラフィックに対しての監視を続けることで，完全に無効にしたときに比べて，CPU 使用率の低下が小さい可能性があるからである．その結果，より多くのルールを移譲する必要が出てくる可能性がある．しかし，5.2.3 項で示すように，トラフィックの半分について監視を続けても十分に負荷を下げることができる．通常サブネット間トラフィックなど IP アドレス制限での監視をするトラフィックは，その他のトラフィックに比べて少ないため，このトラフィックに対する監視の負荷は小さく，大きな影響はないと考えられる．

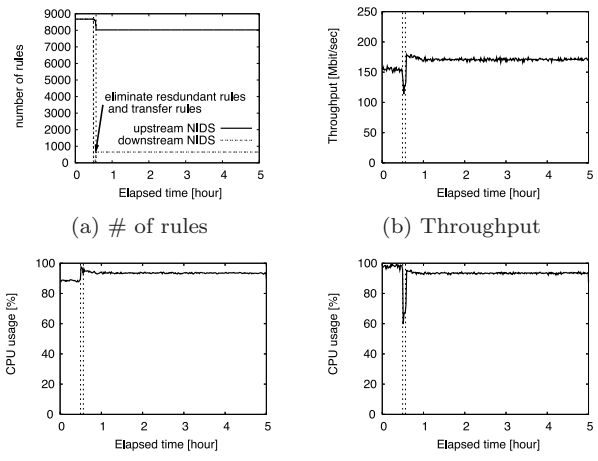
図 3 (b) に Web サーバベンチマークのスループットを示す．初期設定でのスループットは 154 Mbit/sec，負荷分散後のスループットは 174 Mbit/sec となり，13%増加した．現在の実装ではルールの再設定のために NIDS を再起動させなければならないため，ルール移譲中スループットは一時的に減少する．これは，Snort の設定再ロード処理を変更することで再ロード時間を 20%削減した Elephant [28] を用いることで改善できると考えられる．

ルール移譲後のルール数は，上流 NIDS で 6,600，下流 NIDS で 2,076 となった．上流 NIDS は下流 NIDS より高性能であるため，負荷が同程度であるとき，より多くのルールが上流 NIDS で有効となっている．

5.2.2　重複ルールの削除

重複ルールの削除による性能向上の効果を示すため，NIDS の初期ルール設定を，上流・下流双方ともデフォルトルールをすべて有効として，実験を行った．すべての NIDS が同じルール設定となっているため，トラフィックは必ず同じルールに対して 2 回チェックされる．

図 4 に実験結果を示す．図 4 (a) に各 NIDS での有効ルール数の推移を示す．実験開始 30 分後に Brownie が動作を開始すると，まず冗長ルールをすべて削除する．初期設定 BOTH では，すべてのデフォルトルールがすべての NIDS で有効にされているため，下流 NIDS のルールすべてが無効にされる．その結果，すべてのルールが上流 NIDS でのみ有効となる．その後，一部のルールが上流 NIDS から下流 NIDS に移譲され，約 5 分でオフロードが終了した．図 4 (b) に示すように，ベンチマークスループットは，初期設定時 155 Mbit/sec から 173 Mbit/sec に，12%増加した．この性能向上は，同じルールに対して複数検査する必要がなくなったためであると考えられる．



(a) # of rules　(b) Throughput

(c) CPU usage of upstream NIDS　(d) CPU usage of downstream NIDS

図 4　実験結果：冗長ルール削除（初期設定：BOTH）

Fig. 4　Results of eliminating redundant rules (Initial setting: BOTH).

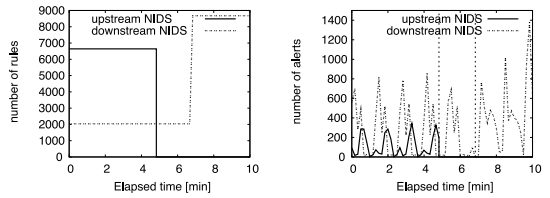
図 4 (c) と (d) に，それぞれ上流と下流 NIDS の CPU 使用率を示す．初期設定では，すべてのルールが有効となっているため，下流 NIDS の CPU 使用率は 100%近くになっている．Brownie が冗長ルールを削除すると，下流 NIDS の CPU 使用率は下がり，上流・下流 NIDS で同程度の CPU 使用率となった．

5.2.3　IP アドレス制限つき移譲

ルールが IP アドレス制限つきで移譲元 NIDS に残った場合においても，NIDS の負荷が軽減することを示すため，すべてのトラフィックを検査した場合と一部のトラフィックを検査した場合とを比較する実験を行った．ここでは，4.1.1.3 で述べたうち，NIDS の下流に通常マシンと下流 NIDS が共存している場合に，上流 NIDS から下流 NIDS にルールが移譲される場合（ケース 3）を想定した．この場合，上流 NIDS には，ルールが IP アドレス制限つきで残ることとなる．

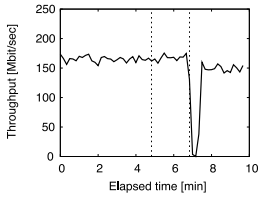
実験では，図 2 において，下流 NIDS B のマシンが NIDS 機能を持たない場合を想定し，上流 NIDS の CPU 使用率とスループットを計測した．下流 NIDS B が存在しないため，上流 NIDS から下流 NIDS A にルールを移譲した場合，上流 NIDS に web サーバ B 向けのトラフィックを検査するためにルールが残ることになる．すべてのトラフィックを検査する場合の設定では，上流 NIDS ですべてのルールを有効にし，すべてのトラフィック（web サーバ A とクライアント A 間のトラフィックおよび web サーバ B とクライアント B 間のトラフィック）を検査する．一方，一部のトラフィックを検査する場合の設定では，下流 NIDS A にすべてのルールを移譲し，web サーバ B とクライアント B 間のトラフィックのみを検査する．web サーバ A とクライアント A 間のトラフィックは，下流 NIDS A で検査するため，検査しなければならないトラフィックは半分となる．

実験の結果，CPU 使用率は，すべてのトラフィックを検査する場合で 100%，半分のトラフィックを検査する場合で 80%となった．また，スループットは，すべてのトラフィックを検査する場合で 186 Mbit/sec，半分のトラフィックを検査する場合で 263 MBit/sec となった．検査するトラフィックが半分になったことにより，CPU 使用率は 20 減少し，スループットは 41%増加した．



(a) # of rules

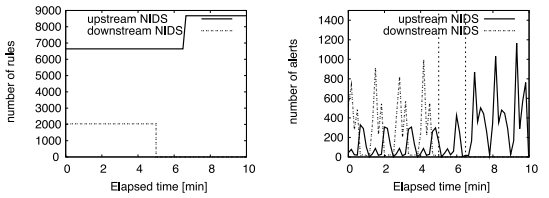
(b) # of alerts



(c) Throughpu

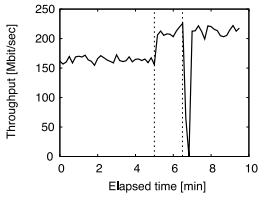
図 5　実験結果：耐障害性（初期設定：BALANCED，上流 NIDS を停止）

Fig. 5　Results of fault-tolerance (Initial setting: BALANCED, Stopped upstream NIDS).



(a) # of rules

(b) # of alerts



(c) Throughput

図 6　実験結果：耐障害性（初期設定：BALANCED，下流 NIDS を停止）

Fig. 6　Results of fault-tolerance (Initial setting: BALANCED, Stopped downstream NIDS).

5.3　耐障害性向上

Brownie による耐障害性向上を示すため，攻撃生成ツールと Web サーバベンチマークを用いた実験を行った．ルールの初期設定として，5.2.1 項でのオフロード終了後のルール設定（BALANCED：ルール数は上流 NIDS が 6,600，下流 NIDS が 2,076）とした．攻撃生成ツールと Web サーバベンチマークを動作させ，開始 5 分後に 1 つの NIDS を停止させたときの，有効ルール数，警告数，ベンチマークスループットを測定した．停止させる NIDS は上流 NIDS を停止させる場合と，下流 NIDS を停止させる場合の実験とした．攻撃生成ツールとして，Nikto 2.1.2 [29] を用いた．

初期設定を負荷分散された後のルール設定で，上流 NIDS を停止させたときの実験結果を**図 5** に示す．図 5 (a) に示すように，実験開始 5 分後に上流 NIDS が停止したため，上流 NIDS の有効ルール数が実質 0 になった．その後 100 秒後に障害を検知し，上流 NIDS で有効にされていたルールを下流 NIDS で有効にしたことにより，下流 NIDS での有効ルール数が増加した．このときの警告の数を図 5 (b) に示す．縦線は，1 つ目が上流 NIDS が停止した時刻，2 つめが下流 NIDS がルール設定を変更した時刻を表す．実験開始時は上流・下流 NIDS 双方で攻撃を検知し警告を発しているが，上流 NIDS を停止させると下流 NIDS のみで警告が発生するようになる．停止後しばらくの間は下流 NIDS で発生する警告数は変化しないが，障害を検知し上流 NIDS でのルールも有効にすると，警告数が増加し，上流 NIDS での障害をカバーできていることが分かる．また，ベンチマークスループットを図 5 (c) に示す．前節の実験でも示したとおり，デフォルトルールをすべて有効にすることは，下流 NIDS には過負荷となるため，障害発生前に比べてスループットは減少している．

初期設定を同じく負荷分散した後のルール設定で，下流 NIDS を停止させたときの実験結果は，**図 6** のとおりである．上記の場合と下流 NIDS と下流 NIDS が逆になった以外は大きな違いはない．なお，下流 NIDS が停止した後のスループットが，停止する前のスループットよりも増加している原因として，下流 NIDS では Snort が停止しているためと考えられる．NIDS で Snort が稼働している場合，パケットを検査するためにユーザレベルに 1 度上げる必要があるが，NIDS が停止している場合そのまま転送するからである．

この実験では，ハートビート 30 秒，障害検知までのハートビート待機時間を 60 秒としたため，障害発生からルール再設定まで 1 分半程度要した．この期間，障害が発生した NIDS で有効になっていたルールに対する攻撃は検知されない．ハートビートの間隔およびハートビート待機時間を短くすることで，この期間を短くすることができる．もちろん，完全冗長のルール設定を行っていれば，この期間はない．しかし，5.2.2 項の実験で示したとおり，完全冗長は性能が低下する．また，今回は性能をスループットで計測するために NIPS として inline-mode で動作させたが，NIDS の場合過負荷になるとパケットの取りこぼしがおきるため，ルールを有効にしていても攻撃を検知できない可能性がある．また，Brownie がない場合は，障害が起こっていない NIDS が元々有効にしていたルールのみを検知する期間が非常に長くなる．これは，たとえば管理者が NIDS の障害に気づき対処するまで，という時間となり，Brownie による障害検知および他 NIDS による肩代わりよりも長い時間になることが予想される．

6.　まとめ

ネットワーク経由の攻撃の検知および防御にネットワーク侵入検知・防御システム（NIDS/NIPS）が広く用いられ

ているが，汎用 PC を用いた NIDS が多い中，ネットワークの高速化や攻撃の巧妙化にともなう性能低下や，NIDS 障害によって攻撃を見逃す耐障害性の問題がある．本論文では，組織内に複数設置された NIDS を相互に協調させることで，性能および耐障害性を向上させる手法を提案する．1 カ所に高価なマシンや並列に動作する複数のマシンを置くのではなく，他の NIDS の負荷やルール設定をもとに，通常時は過負荷のマシンや冗長なルール設定を減らすように再設定をすることで性能向上を，障害時は他の NIDS が検査を肩代わりすることで耐障害性向上を目指す．また，性能向上と耐障害性はトレードオフの関係となるため，管理者がこれらを柔軟に選択できるようにした．実験では，冗長なルール設定が除去され負荷が分散されることで性能向上を，障害時に攻撃検知を他の NIDS が行うことで耐障害性向上ができることを確認した．

今後は，NIDS を協調させることによる他の応用を検討する予定である．たとえば，ルールをある NIDS に集めることで，NIDS の省電力化ができると考えられる．上流 NIDS のみで処理できる程度の負荷であれば，すべてのルールを上流 NIDS で処理し，下流 NIDS を止めることで，下流 NIDS の電力を削減できると考えている．

参考文献

[1] Zhou, C.V., Leckie, C. and Karunasekera, S.: A Survey of Coordinated Attacks and Collaborative Intrusion Detection, *Elsevier Computers & Security*, Vol.29, No.1, pp.124–140 (2010).

[2] Yegneswaran, V., Barford, P. and Jha, S.: Global Intrusion Detection in the DOMINO Overlay System, *Proc. 11th Annual Network and Distributed System Security Symposium* (*NDSS '04*) (2004).

[3] DShield.org, available from ⟨http://www.dshield.org⟩.

[4] Costa, M., Crowcroft, J., Castro, M., Rowstron, A., Zhou, L., Zhang, L. and Barham, P.: Vigilante: End-to-End Containment of Internet Worms, *Proc. 20th ACM Symposium on Operating Systems Principles* (*SOSP '05*), pp.133–147 (2005).

[5] Vallentin, M., Sommer, R., Lee, J., Leres, C., Paxson, V. and Tierney, B.: The NIDS Cluster: Scalable, Stateful Network Intrusion Detection on Commodity Hardware, *Proc. 10th International Symposium on Recent Advances in Intrusion Detection* (*RAID '07*), pp.107–126 (2007).

[6] Xinidis, K., Charitakis, I., Antonatos, S., Anagnostakis, K.G. and Markatos, E.P.: An Active Splitter Architecture for Intrusion Detection and Prevention, *IEEE Trans. Dependable and Secure Computing*, Vol.3, No.1, pp.31–44 (2006).

[7] Kruegel, C., Valeur, F., Vigna, G. and Kemmerer, R.: Stateful Intrusion Detection for High-Speed Networks, *Proc. 2002 Symposium on Security and Privacy* (*S&P '02*), pp.285–293 (2002).

[8] Aho, A.V. and Corasick, M.J.: Efficient String Matching: An Aid to Bibliographic Search, *Comm. ACM*, Vol.18, No.6, pp.333–340 (1975).

[9] Wu, S. and Manber, U.: A Fast Algorithm for Multi-pattern Searching, Technical report, TR-94-17 (1994).

[10] Yang, L., Karim, R., Ganapathy, V. and Smith, R.: Improving NFA-Based Signature Matching Using Ordered Binary Decision Diagrams, *Proc. 13th International Symposium on Recent Advances in Intrusion Detection* (*RAID '10*) (2010).

[11] Clark, C., Lee, W., Schimmel, D., Contis, D., Koné, M. and Thomas, A.: A Hardware Platform for Network Intrusion Detection and Prevention, *Proc. 3rd Workshop on Network Processors & Applications* (*NP3*) (2004).

[12] Bos, H. and Huang, K.: Towards Software-Based Signature Detection for Intrusion Prevention on the Network Card, *Proc. 8th International Symposium on Recent Advances in Intrusion Detection* (*RAID '05*), pp.102–123 (2005).

[13] de Bruijn, W., Slowinska, A., van Reeuwijk, K., Hruby, T., Xu, L. and Bos, H.: SafeCard: A Gigabit IPS on the Network Card, *Proc. 9th International Symposium on Recent Advances in Intrusion Detection* (*RAID '06*), pp.311–330 (2006).

[14] Sidhu, R. and Prasanna, V.K.: Fast Regular Expression Matching using FPGAs, *Proc. 9th Symposium on Field-Programmable Custom Computing Machines* (*FCCM '01*), pp.227–238 (2001).

[15] Baker, Z.K. and Prasanna, V.K.: Time and Area Efficient Pattern Matching on FPGAs, *Proc. 12th International Symposium on Field Programmable Gate Arrays* (*FPGA '04*), pp.223–232 (2004).

[16] Song, H., Sproull, T., Attig, M. and Lockwood, J.: Snort Offloader: A Reconfigurable Hardware NIDS Filter, *Proc. 15th International Conference on Field Programmable Logic and Applications* (*FPL '05*), pp.493–498 (2005).

[17] Gonzalez, J.M., Paxson, V. and Weaver, N.: Shunting: A Hardware/Software Architecture for Flexible, High-Performance Network Intrusion Prevention, *Proc. 14th Conference on Computer and Communications Security* (*CCS '07*), pp.139–149 (online), DOI: 10.1145/1315245.1315264 (2007).

[18] Sommer, R., Paxson, V. and Weaver, N.: An architecture for exploiting multi-core processors to parallelize network intrusion prevention, *Conurrency and Computation: Practice and Experience*, Vol.21, No.10, pp.1255–1279 (2009).

[19] Jacob, N. and Brodley, C.: Offloading IDS computation to the GPU, *Proc. 22nd Annual Computer Security Applications Conference* (*ACSAC '06*), pp.371–380 (2006).

[20] Vasiliadis, G., Antonatos, S., Polychronakis, M., Markatos, E.P. and Ioannidis, S.: Gnort: High Performance Network Intrusion Detection Using Graphics Processors, *Proc. 11th International Symposium on Recent Advances in Intrusion Detection* (*RAID '08*), pp.116–134 (2008).

[21] Vasiliadis, G., Polychronakis, M., Antonatos, S., Markatos, E.P. and Ioannidis, S.: Regular Expression Matching on Graphics Hardware for Intrusion Detection, *Proc. 12th International Symposium on Recent Advances in Intrusion Detection* (*RAID '09*) (2009).

[22] Kaur, P., Rattan, D. and Bhardwaj, A.K.: An Analysis of Mechanisms for Making IDS Fault Tolerant, *International Journal of Computer Applications*, Vol.1, No.24, pp.22–25 (2010).

[23] Kuang, L. and Zulkernine, M.: An Intrusion-Tolerant Mechanism for Intrusion Detection Systems, *Proc. 3rd*

International Conference on Availability, Reliability and Security (*ARES '08*), pp.319–326 (2008).
[24] Siqueira, L. and Abdelouahab, Z.: A Fault Tolerance Mechanism for Network Intrusion Detection System based on Intelligent Agents (NIDIA), *Proc. 4th IEEE Workshop on Software Technologies for Future Embedded and Ubiquitous Systems and 2nd International Workshop on Collaborative Computing, Integration, and Assurance* (*SEUS-WCCIA '06*), pp.319–326 (2008).
[25] US-CERT: Vulnerability Note VU#175500: Snort Back Orifice preprocessor buffer overflow, available from ⟨http://www.kb.cert.org/vuls/id/175500⟩.
[26] Roesch, M.: Snort – Lightweight Intrusion Detection for Networks, *Proc. 13th USENIX Systems Administration Conference* (*LISA '99*), pp.229–238 (1999).
[27] Paxson, V.: Bro: A System for Detecting Network Intruders in Real-Time, *Computer Networks*, Vol.31, No.23–24, pp.2435–2463 (1999).
[28] Merideth, M.G. and Narasimhan, P.: Elephant: Network Intrusion Detection Systems that Don't Forget, *Proc. 38th Annual Hawaii International Conference on System Science* (*HICSS '05*), p.309c (2005).
[29] Nikto, available from ⟨http://cirt.net/nikto2⟩.

花岡 美幸　（正会員）

2005 年電気通信大学電気通信学部情報工学科卒業．2007 年慶應義塾大学大学院理工学研究科開放環境科学専攻修士課程修了．2010 年慶應義塾大学大学院理工学研究科開放環境科学専攻後期博士課程所定単位取得満期退学．同年より，（株）日立製作所中央研究所に所属．システムソフトウェア，インターネットセキュリティ等に興味を持つ．IEEE/CS，ACM，USENIX 各会員．

河野 健二　（正会員）

1993 年東京大学理学部情報科学科卒業．1997 年東京大学大学院理学系研究科情報科学専攻博士課程中退，同専攻助手に就任．現在，慶應義塾大学理工学部情報工学科准教授．博士（理学）．1999 年度，2008 年度，2009 年度情報処理学会論文賞受賞．2000 年度山下記念研究賞受賞．オペレーティングシステム，システムソフトウェア，ディペンダブルシステム，インターネットセキュリティ等に興味を持つ．IEEE/CS，ACM，USENIX 各会員．