段階的通信制限システムを実現する Openflow コントローラとスイッチの試作

M2012MM041 鈴木 雅喬
指導教員：河野 浩之

1 はじめに

近年，様々なベンダーによって多種多様なクラウドサービスが提供されている．その中で，仮想化する端末の増加や，増大するトラフィックの管理，セキュリティ確保など様々な課題を抱えている．それらを解消する手段として近年，Software Defined Network(以下，SDN) という技術が注目されている．OpenFlow[1] は，SDN の中の代表的なもので，米スタンフォード大学を中心とし，2008 年の“OpenFlow: Enabling Innovation in Campus Networks”にて提案された技術である．

本研究では，増大するトラフィックの対策として，SDN の技術の一つである Openflow を利用し，研究室で開発をしている段階的通信制限システム (以下，GK)[2][3][4] の実現をする．研究では Open vSwitch というオープンソースのソフトウェアスイッチを使用する．Open vSwitch は Openflow に対応したスイッチとしての動作も可能であり，Openflow の研究やコントローラの開発などに用いら入れている．Open vSwitch に遅延などを模倣することが可能である netem を組み合わせることで GK を実現するスイッチを実装する．Openflow コントローラを開発するさいに使用するフレームワークには Trema を用いる．通信異常を検知する IDS には Snort を使用する．

本研究の実験では，DoS 攻撃時の通信制限効果の検証，攻撃ホストに通信制限を掛けることでのスループットの変化，本研究で作成システムを用いて向上したスループットなどを調べる．実験には LOIC や apache bench などを用いてスループットの測定や疑似攻撃をする．本研究によって Openflow を用いてシステムを構築することにより，ネットワークの構成の変更が容易になり，煩雑化を防ぐことが予想できる．また，同じパケットに対して複数の通信制限を同時に掛けることが可能になる．加えて，本研究にて作成するシステムは全てオープンソースソフトウェアで構築されるため，安価にシステム構築することが期待できる．

2 既存システムの概要

本節では，既存の GK について説明する．

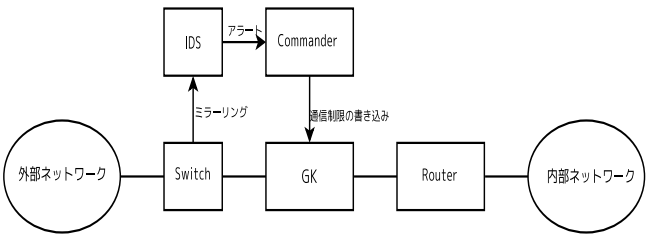

図 1 既存システムのネットワーク構成図

GK は図 1 のように内部ネットワークと外部ネットワークを IP アドレス無しでブリッジとして動作させ，通過時に通信を制限する．IP アドレスをもたないので，攻撃対象にならないという利点がある．IDS を起動させた PC にパケットをミラーし，ネットワーク上で流れるパケットを監視し，検知した攻撃の種類や発信元の IP などの情報を GK に渡し，これらの情報に基づいて GK がリアルタイムに通信制限をする．外部ネットワークから内部ネットワークへの攻撃を制限することを目的としているが，その逆の攻撃も同時に制限することができる．

3 提案する段階的通信制限システムの概要

本研究では Openflow を利用して図 2 のネットワーク構成の GK を実現させる．Openflow は通信機器の制御を行うコントローラと実際にパケット転送をするスイッチで構成されている．コントローラに IDS のアラートやスイッチによって収集されるパケットの統計情報を元に，Openflow スイッチにフローテーブルにフロー書き込むことで，GK を実現する．

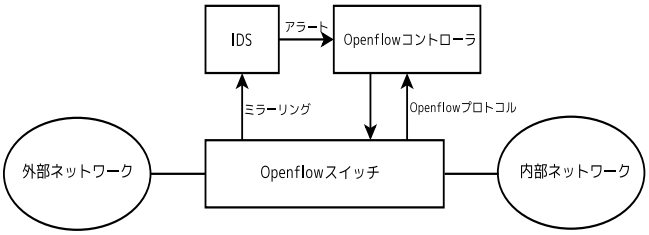

図 2 新段階的通信制限システムのネットワーク構成

3.1 Openflow の概要

Openflow のネットワーク機器は従来のネットワーク機器と異なり，制御部分と転送部分を分離したアーキテクチャである．転送部分を Openflow スイッチ，制御部分を Openflow コントローラ，このふたつの間のやりとりを定義する Openflow プロトコルから成り立っている．

Openflow スイッチはフローテーブルに基づいてパケットの参照と転送を実行する．Openflow スイッチは DataPath ID という 64bit の識別子を持つ．コントローラ側から識別するために，Openflow スイッチはそれぞれ一意的な DataPath ID を持つ．

フローテーブルは以下の 3 つの要素から構成される．

- フローを特定するヘッダフィールド
- 処理内容を決めるアクション
- フローに関する統計情報

Openflow プロトコルは Openflow コントローラと

Openflow スイッチとの間のやりとりの内容を規定するものである．

Openflow コントローラは，Openflow の制御部分を担っており，コントローラのプログラムを記述することで様々なネットワーク処理が実現できる．コントローラを最初からで開発するのは困難なため，コントローラソフトウェアの開発にはフレームワークを用いるのが一般的である．

3.2 新システムと既存システムの違い

既存システムでのシステムの稼働方法やパケット情報の管理は以下のようになっている．

- パケットの情報はルール毎に管理している
- ネットワークの設定が煩雑化しやすい
- ネットワーク機器それぞれに設定しなければならない

Openflow を用いた新システムでは次のようになる．

- フロー毎にパケット情報が管理されていて，コントローラで集計できる
- 既存システムの機能をネットワーク機器に持たせることができる
- コントローラで構成を管理するため，変更が容易に可能である

4 提案する段階的通新制限システムの実現

本節では，本研究で実装する GK を実現するコントローラとスイッチについて記述する．

4.1 スイッチの実現

Openflow 関連の機器は高価なものが多く，製品の種類も少ない．そのため、Open vSwitch を用いて PC を Openflow スイッチとして動作するブリッジとする．表 1 の通り，段階的通信制限には 5 種類の処理があるが，Openflow の標準仕様では 5 種類の中の遅延とパケット欠損を処理することができない．

表 1 Openflow の仕様と GK

通信制限	内容	Openflow	GK
THROUGH	素通し	○	○
DELAY	遅延	×	○
LOSS	パケット欠損	×	○
LIMIT	帯域制限	○	○
DROP	パケット破棄	○	○

今回はスイッチで作られる queue に traffic control の netem を利用して，デバイスに遅延やパケット欠損の処理を設定することで GK の実現に必要な通信制限の queue を用意する．表 2 の 6 段階通信制限を用意して状況によって使い分けることで段階的な通信制限を実現する．

通信制限の遷移は 10 秒毎にフローが書き込まれてから経過した時間とフローで処理されたパケット数やバイト数を照し合せて，異常な数値と思われるものは段階的に通信制限を厳しくしていく．通信制限の監視対象となったフローは素通しの状態から開始される．今回の設定では，

表 2 Queue 設定

queue_id	内容
1	1 秒の遅延
2	1 秒の遅延と 10%のパケット欠損
3	2 秒の遅延
4	2 秒の遅延と 10%のパケット欠損
5	3 秒の遅延
6	3 秒の遅延と 10%のパケット欠損

LOIC を用いて HTTP リクエストの大量送信をしたさいのパケット数とバイト数を参考にして，10 秒毎のフロー毎の統計情報から 1 秒間あたり 200 パケット以上，または 10000 バイト以上の通信があった場合に通信制限が 1 段階厳しくなり，条件を満たさない場合は 1 段階通信制限が緩和されるように設定した．

4.2 netem の概要

netem はネットワーク上でのプロトコルテストなどのために提供されている．遅延とパケット欠損とパケット重複などがエミュレ ート可能である．netem はネットワークデバイスにルールを設定し，そのデバイスから出ていくパケットに対してパケットの処理をする．netem は遅延とパケット欠損を同時に設定することなどもできる．

4.3 コントローラの実装

実装するコントローラは以下の流れの処理をする．

1. ネットワーク上で流れるパケットをスイッチから IDS を起動させたホストへミラーする．
2. IDS がミラーされたパケットの中から異常を検知した場合，異常を検知したこととその内容を Openflow コントローラにアラートとして送信する．
3. Openflow コントローラが読み込んだアラートを元に Openflow スイッチにフローを書き込む．
4. コントローラが登録された Openflow スイッチからフロー毎の統計情報を取得する．
5. コントローラが統計情報を元に，必要な場合はフローのアクションを変更する命令をスイッチに出す．

フローテーブルを制御をするコントローラには Trema フレームワークを利用する．IDS には先行研究 [3] でも使われている Snort を使用する．

次にコントローラを構成するメソッドについて説明する．

start

コントローラ起動時に呼び出される．switch_ready で接続した Openflow スイッチの datapath_id を管理するインスタンス変数を作る．IDS のアラートを読み込む read_alert スレッドを起動する．

read_alert

IDS からのアラートを読み込み，アラートの情報とスイッチに書き込まれているフローの情報を照合する AggregateStatsRequest を送信する．出した命令は stats_reply を通じて処理される．IDS のアラート

を読み込む read_alert スレッドを起動する．

switch_ready
コントローラとスイッチが接続した時に呼び出される．接続したスイッチにスイッチングハブとして動作するフローの書き込みと，捕捉したスイッチの datapath_id を記憶する．

switches_flow_check
コントローラに接続されているスイッチに一定間隔毎にフローテーブルの状態を問い合わせる FlowStatsRequest を送信する．このメソッドで送った命令の結果を stats_reply を通じて表示する．

stats_reply
接続されているスイッチから送られてくる応答があったときに呼び出される．AggregateStatsReply の場合，一致するフローがない場合は新たにフローの書き込みを行う．FlowStatsReply の場合，フロー毎のパケット統計情報を参照して通信制限の変化をする．

flow_mod
フローに新たに通信制限を書き込むために利用する．

queue_modify
フローに書き込まれている通信制限を変更するために利用する．queue_id 毎に通信制限の内容が変化するように設定する．

drop_modify
フローに書き込まれている通信制限をパケット破棄に変更するためのメソッド．パケット破棄の場合は action に何も指定しないフローを書き込むことでパケット破棄を実現している．そのため，queue_modify では queue_id やポート番号を指定するなど，引数が異なるため別のメソッドを用意した．

5 性能評価実験

本節では，実験と評価に関して記述する．図 3 の実験環境を構築する．

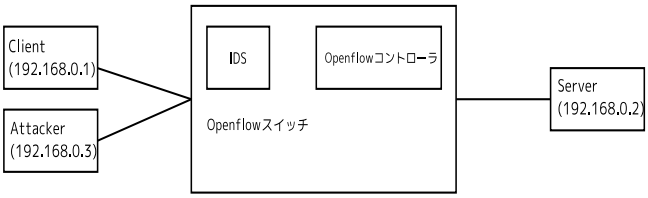

図 3 実験ネットワーク

5.1 通信制限効果評価

通信制限を攻撃ホストに掛けたときにサーバーへの負荷の変化を確認した．Attacker から Server へと 150 クライアントから合計 100000 HTTP リクエストを apache bench を利用して発生させた．負荷を掛けた 2 分間の間，CPU の使用率とメモリのスワップが発生量を計測する．本研究で作成した GK では複数の通信制限を同時に設定できるようになっている．図 4 では，2 秒遅延の通信制限とそれに 10%のパケット欠損を比べたときに CPU システム使用率を示す．

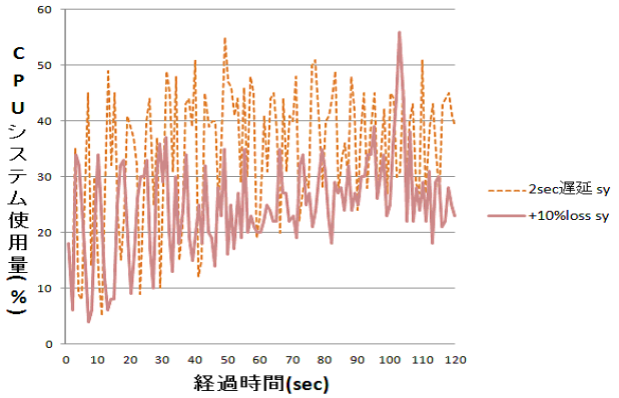

図 4 2 秒遅延時の CPU システム使用率の比較

遅延のみを掛けた場合と比べて使用率が低下しており，サーバー負荷がより軽減されていることが分かる．パケット欠損では 10%，30%，50%の 3 種類で計測したが，50%以外は効果が見られなかった．

5.2 通信制限によるスループット改善

正常な通信として Client から Server へ 10 クライアント，合計 200 リクエストを送信し，それを妨害する通信として Attacker から Server へ apache bench を用いて 140 クライアント，合計 100000 リクエストを送信する．Attacker からの通信に制限を変化させることで Client-Server 間の通信パフォーマンスがどのように変化するかを測定する．攻撃用ホストの通信を素通しした場合，5 秒の遅延を掛けた場合，50%のパケット欠損を掛けた場合の 3 パターンを想定し，それぞれ 10 回計測する．Rps(Request per Second) は 1 秒間あたりの処理されたリクエストの数を示していて，スループットの目安となる．測定した結果を表 3 に示す．

表 3 Rps の測定結果

通信制限	素通し	遅延	パケット欠損
平均	2.21	20.59	25.24
最小値	1.99	14.96	15.32
最大値	2.33	30.74	28.36
分散	0.01	25.84	13.74

グラフと表から攻撃ホストの通信を素通ししたときは 1 秒間あたり 2.2 リクエストしか処理されなかったのが，5 秒の遅延を掛けた場合は約 9.5 倍，50%のパケット欠損を掛け場合は約 12 倍にまで向上している．また，遅延よりパケット欠損の方が分散の値が小さく，通信が安定していることがわかる．

5.3 試作システムの評価

本研究で作成したシステム全体が正確に動作するか，IDS アラートと Openflow の統計情報で段階的に通信制限できることを確認する．コントローラを起動した状態

から HTTP DoS 攻撃として Attacker から合計 100000 リクエストを送信する．開始から 2 分間 Server の CPU 使用率の変化を図 5 に示す．

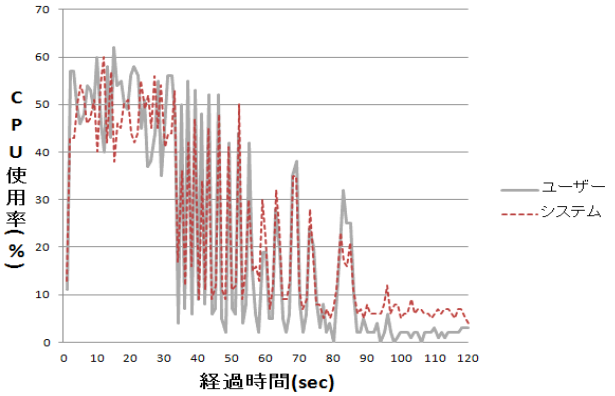

図 5　システム起動時の攻撃による CPU 使用率

コントローラのログから通信開始すぐに Snort がアラートを発し，Attacker が通信制限の監視対象となった．その後，図 5 の経過時間が横に移動していくと CPU の使用率が段々と低くなっているのが分かる．10 秒毎のパケット統計情報のチェックで通信制限が厳しくなり，70 秒経過後，6 段階の通信制限を経て Attacker からの通信が破棄されるようになった．その後 1 度突発的に CPU 使用率があがったものの，サーバーに負荷が掛かっていないのが分かる．

次に，コントローラを起動した状態で Attacker から Server へ apache bench で 140 クライアントから合計 100000 リクエストを送信開始すると同時に Client から Server へ 10 クライアントから合計 2000 リクエストの通信を開始する．その Client からの通信結果を測定する．また，Attacker から攻撃がない状況での Client からの同じ条件での通信も測定し，その結果を表 4 に示す．

表 4　1 リクエストあたりの処理時間と Rps

条件	最小値	平均	中央値	最大値	Rps
攻撃あり	62	545	346	10093	18.31
攻撃なし	62	264	250	889	37.74

システムの構成上，Snort がアラートを発して疑わしいホストを監視対象とした場合，最初のパケット統計情報をチェックするまで通信制限を掛けることはない．そのため，攻撃がある場合とない場合のベンチマークを比べたさい，1 リクエストに掛かった処理時間の最大値に大きな開きがあることが表 4 から分かる．図 6 は横軸に示されているパーセントで表示されている割合の数のリクエストのうちの最大の処理時間の推移を示したものであり，50%での値が中央値にあたる．95%から最大値の間は攻撃の有無の差は大きいことが分かる．表 4 と図 6 から，中央値に近づけば近づく程，差は小さくなっており攻撃ホストの影響を抑えることに成功 しており，表 3 の攻撃を素通しした場合の Rps と攻撃をシステムによって防御した状態と比べて約 8 倍になっている．

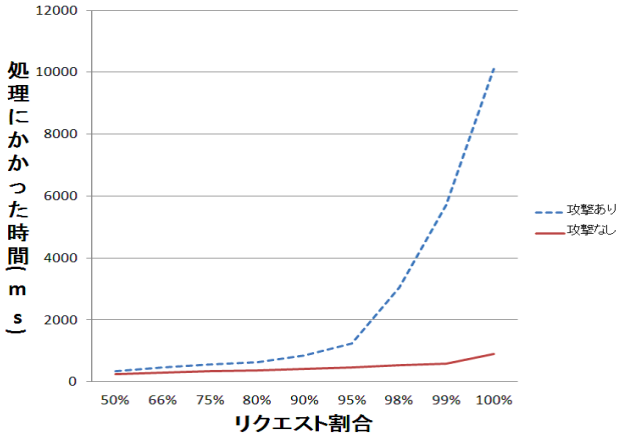

図 6　一定時間内にリクエストが処理された割合

6　おわりに

本研究では，Openflow を利用した GK の試作と HTTP DoS 攻撃を対象とした通信制限効果の評価，本研究で作成したシステムの評価を行った．DoS 攻撃への通信制限効果の評価をし，通信制限をどのように設定することで DoS 攻撃に対して効果が現れるか，スループットがどのように変化するかが分かった．本研究で作成したシステムの評価で HTTP リクエストを用いた DoS 攻撃に効果があることが分かった．

今後の課題として，DoS 攻撃や spam メーラー以外の様々な攻撃に対して実ネットワーク上での実験，それに対応するコントローラの設定追加．内部ネットワークだけでの実験ではなく，外部ネットワークからの攻撃を用いた実験をする．現在用いているフレームワークの Trema がサポートしている Openflow のバージョンは 1.0 であり，このバージョンでは Ipv6 に対応していない． コントローラ開発に用いるフレームワークを Trema-edge にすることで Ipv6 への対応を考える．また，スイッチに使用する PC の性能を上げることで実際のネットワーク環境での効果が期待できる．

参考文献

[1] N. Mckeown, et al., “OpenFlow: Enabling Innovation in Campus Networks”, *ACM SIGCOMM Computer Communication Review*, pp.69–74 (2008).

[2] 伊藤遼平, 嶋田伊吹: IPS の実現とネットワークエミュレータ上での評価, 2009 年度卒業論文, 南山大学数理情報学部情報通信学科 (2010).

[3] 青山正樹, 小島正和: 段階的通信制限を実現するゲートキーパーの提案と試作, 2006 年度卒業論文, 南山大学数理情報学部情報通信学科 (2007).

[4] 福井麻美: 通信制限システムにおける TCP セッションの途中切替と安全なリモートアクセス機能の実装, 2009 年度修士論文, 南山大学大学院数理情報研究科数理情報専攻 (2010).