同時アクセス数制御機構における複数サービスへの対応とクライアント識別機構の改良

17G472　利根 大樹 （最所研究室）

特定のサービスに対して，同時アクセス数を制御する機構において，クライアント識別アルゴリズムの改良，単位時間あたりのアクセス数制御，さらに複数の Web サービスへの対応と HTTPS によるセキュアな通信の実現について述べる．

1　はじめに

当研究室では，ある特定のサービス (特定サービス) を安定的に提供するため，IP アドレスとクライアント単位のフィルタリングを組み合わせた同時アクセス数制御機構を開発している．クライアント単位のフィルタリングは，IP アドレスのみのフィルタリングでは防ぐことのできない NAPT 環境やプロキシを介した許可されないアクセスを防ぐためである．これらの機構により，許可されないアクセスはサーバに届かず，安定したサービスの提供が可能となる．先行研究 [1][2][3] では単一の特定サービスを対象とした実装と，複数サービスを対象とした検討がなされたが，クライアント識別アルゴリズムの処理が遅い問題，特定サービスへの DoS 攻撃が可能となる脆弱性があった．本研究では，クライアント識別アルゴリズムの改良，DoS 攻撃対策，複数の特定サービスへの対応を目指し，本稿ではこれらについて述べる．さらに HTTPS によるセキュアな通信の実現について述べる．

2　先行研究におけるサービス利用の流れ

先行研究での同時アクセス数制御機構における認証からサービス利用までの流れを図 1 に示す．Auth サーバはユーザ認証機能を提供し，IPF サーバは IP アドレス，CI サーバはクライアント単位でフィルタリングする．SS サーバでは特定サービスを提供する．

SS サーバへのアクセスは Auth サーバでユーザ認証を行い (1)，Auth サーバは認証に成功すると，IPF サーバにアクセスの可否 (同時アクセス数の上限を超えていないか) を問合せる (2)．Auth サーバは，アクセス可が返ってくれば，セッション情報を登録し (3)，IPF サーバにフィルタリングルール変更を指示する (4)．IPF サーバはセッション DB を参照し (5)，フィルタリングルールを変更してアクセスを許可する．その後 Auth サーバはクライアントに CI サーバへのリダイレクトを指示し (6)，それによりクライアントはリダイレクトされる (6)(7)．IPF サーバは IP アドレスによるフィルタリングを行う．CI サーバはセッション DB を参照し (8)，アクセスの許可されたクライアントであれば，SS サーバにアクセスし結果をクライアントに返す (9)．許可されていないクライアントの場合は，認証させるため Auth サーバにリダイレクトする．次回以降のアクセスでは，手順 (7) 以降を繰り返す．

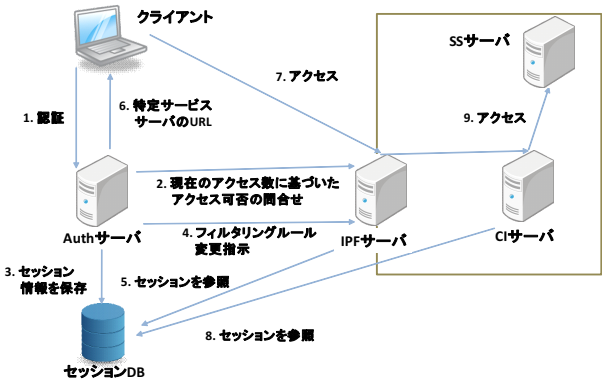


図 1: 先行研究におけるサービス利用の流れ

3　クライアント識別アルゴリズムの改良

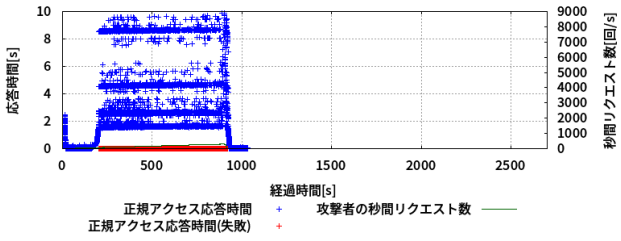
効率的に識別するために CI サーバにセッション情報の複製を持たせるが，先行研究の手法 (旧手法) では，セッション情報を 1 つのファイルに保存し，全探索してクライアント単位のフィルタリングをしていた．しかしこれでは正規のアクセスであるかどうかを判断するのに多くの時間がかかり，攻撃に対する耐性が低くなる．この問題を改善するために，ハッシュを用いてファイル分割する手法 1 と DB(MySQL) を用いる手法 2 を考案した．紙面の都合により性能の良かった手法 1 についてのみ述べる．手法 1 ではセッション情報からハッシュ値を求め，ハッシュ値をファイル名としたファイルに同じハッシュ値のセッション情報を格納した．これによりハッシュ値の個数分の 1 に検索時間を短縮できる．

旧手法と手法 1 に対し攻撃者のアクセスが正規アクセスの応答時間に及ぼす影響を調べる．旧手法では，正規アクセス数を秒間 10，攻撃者のアクセス数を秒間 10 から 300 まで増加させ，手法 1 では，正規アクセス数を秒間 100，攻撃者のアクセス数を秒間 100 から 9000 まで増加させて実験した．どちらの実験も IPF サーバは介さずに行った．なおセッション情報は 3900 個作っており，手法 1 におけるハッシュ値の個数は 1296 である．

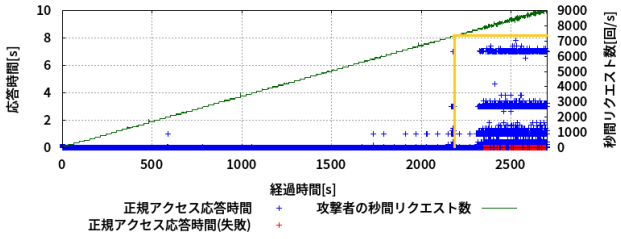
実験結果を図 2 に示す．青と赤の十字はそれぞれ正規アクセス成功時と失敗時の応答時間，緑線は攻撃者の秒間リクエスト数を表す．図 2(b) の黄色の線は補助線である．図 2(a) では秒間 60 リクエスト付近，図 2(b) では補助線より秒間 7500 リクエスト付近からアクセス失敗や応答時間の著しい増加が見られる．以上から手法 1 が旧手法を改善できていることがわかる．

4　単位時間アクセス数制御

先行研究では，認証後であれば SS サーバへ DoS 攻撃ができてしまう．これに対処するために本研究では，IPF サーバと CI サーバで単位時間あたりのアクセス数 (単位時間アクセス数) を制御することにした．CI

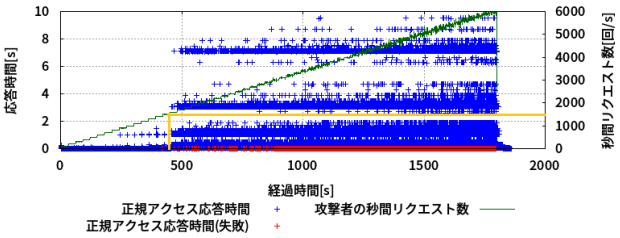


(a) 旧手法

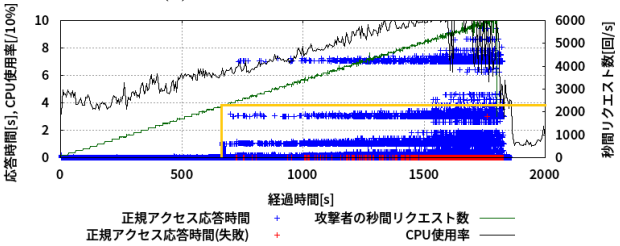


(b) 手法 1

図 2: 正規と攻撃者のアクセスによる実験



(a) アクセス制御をしない場合



(b) アクセス制御する場合

図 3: CI サーバにおけるアクセス数制御の実験結果

サーバ，IPF サーバそれぞれで単位時間アクセス数上限を持たせる．CI サーバの上限を上限 1，IPF サーバの上限を上限 2 とする．CI サーバでは単位時間アクセス数が上限 1 を超えるとアクセス制限し，上限 1 を下回ると制限解除する．IPF サーバでは上限 2 を超えるとアクセス制限し，制限から一定時間経過すると制限解除する．上限 1，上限 2 ともに CI サーバで超過を検知し，IPF サーバでのアクセス制限は CI サーバからの指示によって行われる．本研究では CI サーバでの単位時間アクセス数制御を実装した．

本機能の導入前と導入後で同じ実験を行った．実験は，複数の上限 1 を超えない正規アクセスから秒間合計 100 リクエストでアクセスさせ，上限 1 を超える攻撃者のアクセスを秒間 100 から 6000 まで増加させる．図 3 に実験結果を示す．緑線は攻撃者の秒間リクエスト数，青と赤の十字はそれぞれ正規アクセスの成功時と失敗時の応答時間を表す．黄色の線は補助線である．またアクセス制御による処理量を見るため，図 3(b) に CI サーバの CPU 使用率 (黒線) を追加している．

各図の補助線を見ると，図 3(a) では秒間 1500 リクエスト付近，図 3(b) では秒間 2000 リクエスト付近からアクセス失敗と応答時間の増加がみられる．このことから，アクセス数制御機能により DoS 攻撃への耐性を向上できたことが確認できる．しかし，図 3(b) では攻撃クライアントの秒間リクエスト数の増加に伴い CPU 使用率も増加しており，CI サーバだけでの単位時間アクセス数制御では大きな改善はできていない．

5　複数の特定サービスへの対応

クライアントは利用したい特定サービスの認証用ドメインで，Auth サーバへアクセスしログインする．認証用ドメインは各特定サービスによって異なる．さらに各サービスごとに同時アクセス数の上限が設定されており，各サービスごとに同時アクセス数制御が行われる．ログイン後，クライアントは CI サーバへリダイレクトされるが，アクセス時のドメインはサービスごとに異なる．CI サーバではそのアクセス時のドメインに基づいて，それぞれの SS サーバへリバースプロキシする．CI サーバでは特定サービス情報を DB に保持しており，ここからリバースプロキシ先を取得する．本研究ではサービスごとの同時アクセス数制御，CI サーバでの DB を利用したリバースプロキシ先の決定を実装し，設計通りの動作確認を行った．

6　HTTPS を用いたセキュアな通信の実現

CI サーバはリバースプロキシとして動作する．そのためクライアントと SS サーバ間の通信をセキュアにするには，クライアント－ CI サーバ間，CI サーバー SS サーバ間の両区間で HTTPS を用いる必要がある．先行研究ではこれを実現できなかったが，Nginx モジュールの lua-nginx-module[4] を用いてクライアント識別機能を実装することで，両区間の HTTPS による通信を実現した．

7　まとめと今後の課題

本稿ではクライアント識別アルゴリズムの改良，CI サーバの単位時間アクセス数制御の実装，複数の Web サービスへの対応および HTTPS を用いたセキュアな通信の実現を述べた．今後は CI サーバを多重化しクライアント識別においてより多くのリクエストを処理できるようにする予定である．さらに IPF サーバでの単位時間アクセス数制御の実装を目指す．

参考文献

[1] 大川昌寛, “ファイアウォールを用いた Web サーバのための同時アクセス数を動的に制御するアクセス制御機構の設計” 情報処理学会第 77 回全国区大会講演論文集, 5X-4, pp.3-169 - 3-170, 2015 年.

[2] 近藤裕基, “ファイアウォールを用いた同時アクセス数制御機構におけるアクセス権管理機能の実装” 学士論文, 2017 年.

[3] 利根大樹, “同時アクセス数制御機構におけるクライアント識別機構の実装” 学士論文, 2017 年.

[4] lua-nginx-module, https://github.com/openresty/lua-nginx-module