マルチユーザシステムにおける利用者単位でのネットワークアクセス制御手法

山井成良[†1] 眞鍋宏隆[†2] 岡山聖彦[†1]
宮下卓也[†3] 松浦敏雄[†4]

教育用計算機環境においては，様々なレベルの利用者が存在するため，ネットワークに対する利用者単位でのアクセス制御機能が重要である．しかし，UNIX，LINUX などのマルチユーザシステムでは，ほとんどが利用者単位でのアクセス制御機能を備えておらず，またこのような機能を備えているマルチユーザシステムであっても膨大な数のアクセス制御ルールを設定する必要があるため，管理コストが高かったり，性能が劣化したりする問題があった．そこで本論文ではこれらの問題を解決するために，アクセス制御ルールを利用者単位に分割して，パケット所有者に関するルールのみを参照する方法を提案する．また，本方法では TCP 通信だけでなく UDP 通信に対してもフロー単位でアクセス制御を行うようにする．これにより，利用者間でのルールの共有による管理の省力化や動作の高速化が可能となる．提案手法に基づいた試作システムの性能評価の結果，多数の利用者が登録されている場合でも十分高速にアクセス制御を行うことができ，提案手法の有効性が確認された．

Design and Implementation of User-based Network Access Control Mechanism on Multiuser Systems

Nariyoshi Yamai,[†1] Hirotaka Manabe,[†2] Kiyohiko Okayama,[†1]
Takuya Miyashita[†3] and Toshio Matsuura[†4]

On educational computer environment, a user-based network access control mechanism is important since there exist many kinds of users. However, as for existing multiuser systems such as UNIX and LINUX, most of them have no such a mechanism or otherwise they have some problems on this access control mechanism, such that administrative cost becomes considerably large, and the performance of network degrades, since a huge number of access control rules are required. In this paper, in order to solve these problems, we propose a method that divides the whole rules into individual rule sets and that refers to only the rule set of the packet owner. In addition, access control per flow is performed on both TCP and UDP communications. Accordingly, this proposed method reduces administrative cost by sharing of rule sets among users and improves performance. According to the result of performance evaluation of a prototype system based on the proposed method, the performance of access control is improved significantly even if many users exist on the system, and consequently we confirm that the proposed method works effectively and efficiently.

1. はじめに

UNIX，LINUX などに代表されるマルチユーザシステムは，複数の利用者が同時に使用できることから，大学などの教育機関では教育用計算機環境における基幹システムとして広く活用されている．たとえば，著者らが所属している岡山大学総合情報基盤センター（以下，当センターと記述する）では，教育用計算機システムの基幹サーバとして Sun Enterprise 3500（OS: Solaris 2.6）を導入しており，プログラミング教育あるいは数式処理，統計処理など，性能や機能などの面で PC では実行できないようなアプリケーションを利用した教育のために使われている．

マルチユーザシステムでは権限の異なる様々な利用者が存在するため，共用ディレクトリや利用者の個人用ファイルなどの計算機資源に対してそれぞれ適切な

†1 岡山大学総合情報基盤センター
Information Technology Center, Okayama University
†2 キヤノンシステムソリューションズ株式会社
Canon System Solutions Inc.
†3 津山工業高等専門学校
Tsuyama National College of Technology
†4 大阪市立大学大学院創造都市研究科
Graduate School of Creative Cities, Osaka City University

アクセス制御を行う必要がある．通常，この機能はオペレーティングシステムにより提供され，ログイン時に認証した利用者名およびその利用者が所属するグループ名と，各ディレクトリやファイルに設けられた利用者およびグループ単位（以下，特に断らない限り，これらを代表して単に利用者単位と記述する）でのアクセス許可設定とを照合することによりアクセスの可否が判断される．

一方，特に教育用計算機環境においては，ネットワークに対する利用者単位でのアクセス制御機能が重要である．これには管理上重要なネットワークや計算機を一般利用者が利用する教育用 PC から保護することだけでなく，たとえばネットワーク利用に関して十分な教育を受けていない利用者が学内ネットワークしかアクセスできないように制限したり，教育用計算機環境を利用して試験を行う場合に受験生による不正行為を防止するために試験用サーバしかアクセスできないように制限したりすることも含まれる．このような機能を実現するため，当センターでは教育用 PC に対してプライベートアドレスを割り当てたり，ルータによるフィルタリングを実施したりするだけでなく，利用者の多い WWW に関しては特に利用者単位によるアクセス制御機能[1] を開発し，学外への最初のアクセス時に注意事項を表示してこの遵守に同意したものだけにアクセスを許可するなどの設定で実際に運用している．このような機能は，教育用計算機環境以外においても重要であり，このような機能があれば，企業においてもたとえば正社員と派遣社員との間でアクセス可能な範囲に差を設けたい場合に適用できる．

ところが，マルチユーザシステムでは，ネットワークに対する利用者単位でのアクセス制御が困難である．すなわち，マルチユーザシステムで送受信されるパケットには利用者に関する情報が含まれていないため，1 台の教育用 PC を同時にはたかだか 1 人の利用者しか使用しない場合とは異なり，ルータによるフィルタリングを行うことはできない．また，多くのマルチユーザシステムでは，上記のような利用者単位でのネットワークアクセス制御機能は提供されておらず，同機能を提供しているシステムでも多くの利用者が存在する教育用計算機環境では管理コスト面や性能面で不十分である．したがって，たとえば統計処理などの演習を行うためにマルチユーザシステムを受講者に使用させるような状況では，たとえ教員が受講者には外部ネットワークへの目的外アクセスを拒否したい場合でも，受講者は教員を含む一般利用者と同じ権限でネットワークにアクセスできることになる．その結果，無許可でのファイル転送や他の計算機への不正アクセスなど，受講生によるネットワークを介した不正行為を許してしまう危険性が生じる．

そこで，本論文では上記の問題を解決するため，マルチユーザシステムにおける利用者単位での効率的なアクセス制御手法を提案する．本手法では広範囲で利用されている，ユーザプロセスレベルでのパケットフィルタリング機能と同様の仕組みで動作するため，多くの種類のマルチユーザシステムにおいて利用可能である．また，本手法では，フィルタリングルールをあらかじめ利用者ごとに分割して記述しておき，コネクション確立時にその終端ソケットの所有者に対するフィルタリングルールのみを参照することにより，多数の利用者が登録されているシステムにおいても評価するルール数を抑え，効率的なフィルタリングが可能となる．

以下，2 章では従来のアクセス制御手法の問題点について議論し，3 章では提案手法の詳細を述べる．また，4 章では提案手法の有効性を検証するために実施した性能評価実験について述べる．

2. 従来のアクセス制御手法と問題点

多くの OS ではパケットフィルタリング機能が OS の標準機能あるいは外部プログラムとして提供されており，これを用いれば一般利用者のネットワークアクセスをある程度制御することが可能である．たとえば IP Filter[2] は Solaris や FreeBSD など多くの OS で利用可能なパケットフィルタリング用ソフトウェアであり，これを用いればプロトコル，IP アドレス，ポート番号，各種フラグなどパケットヘッダに含まれる情報に基づいたアクセス制御が可能となる．しかし，利用者情報はパケットヘッダに含まれていないため，IP Filter では利用者を指定したアクセス制御は行えない．

一方，たとえば FreeBSD で標準機能として利用可能なパケットフィルタリング用ソフトウェア ipfw[3] では，フィルタリングルールとして利用者名やグループ名を指定できる機能が備えられており，この機能を利用すれば利用者単位でのアクセス制御を行うことが可能である．しかし，この機能は FreeBSD のカーネルレベルで実装されているため，他の OS では容易には利用できないという問題がある．また，この機能は登録利用者数が多数存在するような大規模な環境（当センターの例では約 21,000 人）では管理面あるいは性能面で重大な問題が生じる．すなわち，ipfw では各ルールについて指定できる利用者名やグループ名がたかだか 1 つに限られるため，大規模な環境では各利

用者について同一のルールをそれぞれ設定する必要が生じ，管理者がフィルタリングルールを管理する負担が大きくなる．また適用されるフィルタリングルールは原則として線形探索により決定されるため，平均でルール数に比例した探索時間がかかり，特にルール数が膨大になるとフィルタリングのオーバヘッドが大きな問題となる．

これらの問題は，フィルタリングルールに対して利用者単位ではなくグループ単位での適用を指定することにより，ある程度軽減することが可能である．すなわち同一の内容のアクセス制御を受ける利用者集合に対して 1 つのグループを割り当てることにより，利用者ごとの設定が必要であったルールリストを 1 つにまとめて設定することができ，これにより管理者への負担やフィルタリングのオーバヘッドを軽減できる．しかし，たとえば教育用計算機環境を利用した試験での利用を考慮すると，少なくとも講義ごとに異なるグループを割り当て，それぞれのグループに対してフィルタリングルールを設定する必要が生じるため，上記の問題はそれほど大きくは改善されない．さらにこの方法では各利用者は履修するすべての講義についてその講義に割り当てられたグループに所属する必要があるが，多くの OS では 1 人の利用者が所属可能なグループ数が最大でも 16 程度に制限されている点も新たな問題となる．

オーバヘッドを軽減する別の方法として，ipfw における skipto などの分岐ルールの利用が考えられる．すなわち，パケットが特定利用者に属していればその利用者に適用されるルールリストに分岐するようなルールを作成することにより，ルールリストを利用者間で共有化したり，無関係なルールの探索を省略したりできる．しかし，この方法でも各分岐ルールには条件として 1 利用者しか指定できないため，ルールリストの先頭には利用者数分の分岐ルールを列挙する必要が生じる．この場合，分岐ルール自身は線形探索されるため，平均で利用者数に比例した探索時間が必要となり，登録利用者が多数存在する環境ではオーバヘッドが大きすぎる状況に変わりはない．

さらに，この方法ではルールリストの共有化が柔軟には行えず，管理コストが大きくなる点も問題である．すなわち，共有ルールリストを評価した後に再び利用者単独のルールリストを評価したい場合，共有ルールリストの末尾で再び分岐ルールを列挙する必要が生じるため効率の面で問題となりうる．また ipfw ではルールを番号で管理しており，このルール番号空間は 1 つしかないため，利用者数やルール数が大きくなると，番号と利用者との対応や利用可能な番号の把握が管理者にとって負担となる．

以上のような理由から，従来のネットワークアクセス制御手法では，多数の利用者が存在する大規模な教育用計算機環境において利用者単位でアクセス制御を行うには不十分といえる．

3. 効率的なアクセス制御手法

3.1 提案手法の概要

2 章で述べたような問題を解決するため，本論文では適用範囲が広く，かつ効率的な利用者単位でのネットワークアクセス手法を提案する．本手法は，ipfw のようにマルチユーザシステム上で利用者に応じたフィルタリング機能を提供するが，カーネルレベルではなくユーザレベルで動作するため，多くの OS で動作する．

まず，効率的なアクセス制御を行うため，提案手法で採用した技法について述べる．

ipfw が持つ 2 章で述べたような問題の根本的な原因としては，次の 2 点があげられる．原因の 1 つは，適用対象となる利用者あるいはグループを指定したルール（分岐ルールを含む）は，他の利用者やグループには明らかに適用されないにもかかわらず毎回評価されるためオーバヘッドが大きくなってしまう点で，もう 1 つは，複数の利用者あるいはグループに対して共通に適用されるルールリストの柔軟な共有が困難な点である．

そこで，提案手法ではこれらの原因を排除するため，利用者ごとに適用するルールリストを個別のファイルに格納しておき，フィルタリング実施時にはまず通信の終端となるソケットの所有者を調べ，その結果に応じて必要なルールリストのみを評価する技法を採用する．これにより，前者の問題点については，他の利用者用のルールリストは参照されないため，原理上は全体のルール数や利用者数に影響を受けることなくオーバヘッドを大幅に軽減することが可能になる．また，後者の問題についても，他の利用者用のルールリストとの干渉を考慮する必要がないだけでなく，ルールリストの共有を（シンボリック）リンクなどのファイル共有により実現可能となる．さらに利用者間でルールリストの大部分が共通であるが一部が異なるような場合でも，たとえばマクロプロセッサを活用して共通部分を共有化することが可能になり，管理者の負担を大幅に軽減することが可能になる．なお，本論文では利用者の所属グループに基づいたアクセス制御は考慮していないが，提案方法をそのように拡張することは容易である．

以下では提案手法を実現するうえで問題となる，パケットの取得方法，利用者情報の取得方法，オーバヘッド削減のためのルールリストの記述方法について記述する．

3.2 パケットの取得

パケットフィルタリングを行うためには，対象となるパケットを取得する必要がある．そこでフィルタリングの対象となるパケットをカーネル内で横取りしてアクセス制御プログラムに渡せばよい．その際，高速化のためには，たとえば TCP 通信についてはコネクション確立時に送出される SYN フラグ付きパケット（ただし，ACK フラグは設定されていないもの）のみをカーネル内で横取りしてアクセス制御プログラムに渡すなどの工夫を行う必要がある．

このようなパケット取得機能は多くの OS で標準で提供されており，たとえば FreeBSD では ipfw の divert 機能[4] がこれに該当する．また，たとえば Solaris など標準ではこのような機能が提供されていない OS でも，IP Filter と同様に LKM（loadable kernel module）などの利用により同等の機能を実現することは多くの OS で可能であり，提案手法の実現の障害にはならないと思われる．

3.3 利用者情報の取得

利用者単位でのアクセス制御を行うためには，通信を行おうとしている利用者を特定する必要がある．ところがパケットのヘッダには利用者情報がいっさい含まれておらず，パケットから直接利用者名を特定することはできない．このため，まずパケットから宛先 IP アドレス，宛先ポート番号，送信元 IP アドレス，送信元ポート番号の 4 つ組を取り出して TCP コネクションまたは UDP フローを特定し，これらを終端するソケットの所有者名を，カーネル内のソケットテーブル（ファイルテーブル）およびプロセステーブルから取得する必要がある．

このような利用者名の取得方法はカーネルの内部構造に依存するため OS によって若干の違いがあるが，TCP については，IDENT プロトコル[5] の実装に必要であるため，多くの OS について知られており，たとえば主要な OS で利用可能なフリーの IDNET サーバである pidentd[6] のソースコードにも含まれている．UDP についても似たような方法によって利用者名を取得することができる．また FreeBSD では OS の標準機能として利用者名の取得機能が提供されており☆，これを用いることもできる．

☆ TCP では `sysctlbyname("net.inet.tcp.getcred", ...)`，UDP では `sysctlbyname("net.inet.udp.getcred", ...)` を用いる．

3.4 UDP への対応

UDP は TCP と違い，コネクションという概念を持たない．そのためパケットの取得および利用者情報の取得の段階で，TCP とは異なる動作が必要となる．パケットの取得の段階では，TCP ではコネクション確立時に送出される SYN フラグ付きパケット（ただし ACK フラグは設定されていないもの）のみを取得すればよいが，UDP ではすべてのデータグラムを取得する必要がある．利用者情報の取得の段階でもまた，TCP の場合は 1 コネクションに対し 1 つのパケットの利用者を特定すればよいが，UDP ではすべてのデータグラムの利用者を特定する必要がある．このように UDP では，TCP に比べてより多くの処理が必要となり，フィルタリングによるオーバヘッドが著しく大きくなる可能性がある．

そこで同一の宛先 IP アドレス，宛先ポート番号，送信元 IP アドレス，送信元ポート番号を持つ UDP データグラムは，一定時間の間，同一利用者の同じ通信と見なし，最初の 1 データグラムの通信が許可されれば，以降，同じ通信のデータグラムに対しては利用者の特定やルール探索を一定時間のあいだ省略する．このようにフロー単位で UDP 通信を制御することによって，TCP と同じように効率的なアクセス制御を行うことができる．以下では，この機能を UDP フローキャッシュ機能と呼ぶことにする．

なお，UDP では，同一のフローに対してある利用者の使用終了直後に他の利用者がこれを再利用できるため，上記の一定時間の間にアクセス制御が無効になる危険性がある．これについては 4.2.4 項で考察する．

3.5 利用者単位でのルールの指定

提案手法では利用者ごとに個別のファイルとしてルールリストを記述することから，どのようにルールリストを記述するか，また設定ファイルをいつ読み込むかが性能面に大きな影響を与えることになる．

たとえば，1 つの方法として，アクセス制御プログラムの起動時に全利用者について設定ファイルを一括して読み込む方法が考えられる．この方法ではパケット取得時にはファイルアクセスが発生しないため，オーバヘッドは小さくなる反面，利用者数が多いと起動時にかなりの時間がかかるだけでなくメモリ上に格納するルールの数が膨大になり，メモリの利用効率が悪い点が問題となりうる．一方，プログラム起動時には設定ファイルをまったく読み込まず，パケット取得時にソケット所有者の設定ファイルだけを読み込む方法も考えられる．しかし，この方法ではパケットを取得す

```
0 from any to any
1 tcp from 172.16.1.1 to 172.17.0.0/16 22
2 udp from 172.16.1.1 to 172.17.0.0/16
3 from 192.168.1.0/24 to 192.168.1.0/24
```

図 1 ルールファイルの例
Fig. 1 An example of rule description file.

```
reject 1
allow 2
allow 3
reset 0
```

図 2 利用者制御ファイルの例
Fig. 2 An example of user control file.

るたびに設定ファイルをオープンしてルールを解析する必要が生じるため，オーバヘッドが大きくなってしまうことが予想される．

そこで，提案手法ではいずれかの利用者に適用する可能性があるルール群を記述するファイル（ルールファイル）と，利用者ごとにどのルールをどのような順番で適用するかを指示するファイル（利用者制御ファイル）を分離し，前者をプログラム起動時に，後者をパケット取得時に読み込むようにする．また，ソケット所有者に対する利用者制御ファイルが存在しなかった場合や，存在した場合でも合致するルールが見つからなかった場合に使用する，デフォルト制御ファイルも指定できるようにする．これにより，複数の利用者制御ファイルで共通に参照されるルールの記述が一度だけで済み，オーバヘッドを抑えながらメモリの利用効率を高めることが可能となる．さらに，一度読み込んだ利用者制御ファイルやデフォルト制御ファイルについて，キャッシュを行うようにする．これにより，同一利用者が今後行う通信についてはファイルへのアクセスが省略されるため，オーバヘッドのさらなる削減が可能になる．

アクセス制御を行う計算機が 172.16.1.1 と 192.168.1.1 の 2 つのアドレスを有している場合におけるルールファイル，利用者制御ファイルおよびデフォルト制御ファイルの記述例をそれぞれ**図 1**，**図 2**，**図 3** に示す．

この例では，まずルールファイルにおいて 0 番はすべての通信，1 番は当該計算機からネットワーク 172.16.0.0/16 の 22/TCP ポートへの通信，2 番は当該計算機からネットワーク 172.16.0.0/16 への任意の

```
deny 0
```

図 3 デフォルト制御ファイルの例
Fig. 3 An example of default control file.

UDP 通信，3 番はネットワーク 192.168.1.0/24 内の任意の通信☆を表し，利用者制御ファイルやデフォルト制御ファイルでこれらを参照している．すなわち利用者制御ファイルの設定では，まず 1 番のルールに該当するパケットは中継を拒否され，送信元に ICMP メッセージ（type=3，code=13）が送られる（reject）．一方，2 番，3 番のルールに該当するパケットは中継が許可される（allow）．それ以外の通信は 0 番のルールに該当し，パケットは中継を拒否され，通信が TCP であれば送信元に RST フラグ付きパケットが送られる（reset）．同様にデフォルト制御ファイルの設定ではすべてのパケットが中継を禁止され，何のパケットも送り返すことなく，パケットは破棄される（deny）．

3.6 アクセス制御の手順

これまでに述べた技法に基づき，提案手法におけるアクセス制御の全体手順を示す．

(1) アクセス制御プログラムは，起動されるとまず送信元あるいは宛先が自身の計算機で，TCP については SYN フラグ付きのパケット（ただし ACK フラグは設定されていないもの）および UDP についてはすべてのパケットを OS から自身に引き渡されるように設定する．引き続いてアクセス制御プログラムはルールファイルを読み込み，各ルールを解釈してメモリ内に記憶する．

(2) OS からパケットを受け取ると，アクセス制御プログラムはパケットのヘッダから宛先 IP アドレス，宛先ポート番号，送信元 IP アドレス，送信元ポート番号を読み取り，これらの情報に基づいて TCP コネクションまたは UDP フローの終端となるソケットの所有者名を OS から取得する．

(3) ソケット所有者に対する利用者制御ファイル（このファイルが存在する場合）およびデフォルト制御ファイルがキャッシュされているかどうかを調べ，もしキャッシュが存在すればそれらを利用する．そうでなければ利用者制御ファイル

☆ 実際には当該計算機と 192.168.1.0/24 内との任意の通信を表す．

あるいはデフォルト制御ファイルを読み込み，その内容をキャッシュする．

(4) 利用者制御ファイルおよびデフォルト制御ファイルをこの順で 1 行ずつ読み取り，(1) で記憶したルールのうち各行で指定されたものとパケットを順に照合して最も早く判明した適用可能なルールを特定する．

(5) 特定したルールに対する動作（allow，deny，reject，reset のいずれか）に従ってパケットを処理する．動作が allow の場合にはパケットを中継し，deny，reject，reset の場合にはパケットを破棄する．さらに，reject の場合には ICMP メッセージを，reset の場合（TCP のみ有効）には RST フラグ付きパケットを送信元に送り返す．なお (4) で適用可能なルールが見つからなければ，deny と同じ処理を行う．

4. 試作システムの実装と性能評価

4.1 試作システムの実装

前章で述べた手法に基づき，我々は試作システム purifier（per user IP filter）の実装を行った．試作システムは Pentium 4 （2 GHz）を搭載した計算機上で動作し，その OS は FreeBSD 4.11-RELEASE である．したがって前章で述べたようにパケットの取得には ipfw の divert 機能を用い，また利用者情報の取得には `sysctlbyname()` を用いた．また ipfw の設定としては**図 4** に示したものを用いた．

試作システムでは，利用者制御ファイル用のキャッシュやルールファイルの格納にはすべて静的な配列を用いて実装した．すなわち，前者については，本システムで想定した最大利用者数（32,000 人）分の配列を用意し，利用者 ID（UID）を添字にして検索できるようにした．1 ルールあたりの記憶領域は 12 バイトであり，1 利用者あたりのルール数を最大で 64 に制限しているため，システム全体では約 25 M バイトの記憶領域を利用していることになる．一方，後者については 1 ルールあたりの記憶領域は 32 バイトであり，最大で 256 ルールを登録できるようにしているため，必要な記憶容量は 8 K バイト程度である．なお，これらの記憶領域はハッシュを用いて動的に確保することも可能であり，その場合にはルール数にも依存するが，最大同時利用者数分の記憶領域（100 人の場合で 80 K バイト程度）があれば十分であると思われる．

```
0 allow tcp from any to any established
1 divert tcp from any to any setup
2 divert udp from any to any
```

図 4 試作システムにおける ipfw の設定
Fig. 4 Configuration of `ipfw` on the prototype system.

4.2 試作システムの性能評価

4.2.1 実験の概要

次に提案手法の有効性を検証するため，試作システムの性能評価実験を行った．実験におけるシステム構成を**図 5** に示す．

本実験では purifier を導入した計算機（Host A）と別の計算機（Host B，CPU: Pentium3 500 MHz，OS: FreeBSD 4.1-RELEASE）との間で，TCP については TCP コネクションの確立，解放を 100 回繰り返す自作プログラムを動作させ，これに要する時間を 1,000 回測定してその平均を求め，UDP については TFTP [7] を用いて約 4 M バイト（UDP データグラム約 16,000 個分）のデータ転送を行い，これに要する時間から UDP データグラム 100 個分に相当する時間を求めた．その際，いずれの実験でも，登録利用者数を 10,000 人，利用者あたりの適用ルール数を 10 とした．

4.2.2 ipfw との比較

まず，試作システムの性能を，従来の ipfw を単独で使用した場合と比較する実験を行った．この実験では比較の条件として，最もオーバヘッドが小さい場合である，最初のルールが適用される場合（以下，最速と表記する）と，最もオーバヘッドが大きい場合である，最後のルールが適用される場合（以下，最遅と表記する）の 2 つの場合を想定し，各場合について TCP および UDP の通信時間を測定した．

性能評価実験における purifier のルールファイルを**図 6** に，purifier の利用者制御ファイルを**図 7**，**図 8** に，また ipfw を単独で使用した場合のルールリストを**図 9**，**図 10** にそれぞれ示す．ここで，図 5 に示すように Host A，Host B の IP アドレスはそれぞ

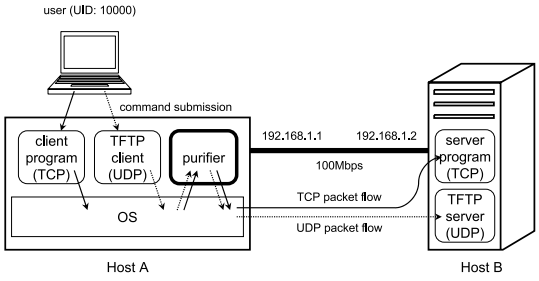

図 5 性能評価実験のシステム構成
Fig. 5 System configuration of the performance evaluation experiment.

```
0 from 192.168.0.0 1 to 192.168.0.1
1 from 192.168.0.0 2 to 192.168.0.1
...
8 from 192.168.0.0 9 to 192.168.0.1
9 from 192.168.1.1 to 192.168.1.2
```

図 6 性能評価における purifier のルールファイル
Fig. 6 The rule file of *purifier* used for performance evaluation.

```
allow 9
```

図 7 purifier の利用者制御ファイル（最速の場合）
Fig. 7 The user control file of *purifier* used for performance evaluation (for the best case).

```
deny 0
deny 1
...
deny 8
allow 9
```

図 8 purifier の利用者制御ファイル（最遅の場合）
Fig. 8 The user control file of *purifier* used for performance evaluation (for the worst case).

```
0 allow tcp from any to any established
1 allow ip from 192.168.1.1 to 192.168.1.2 uid 10000
```

図 9 ipfw のルールリスト（最速の場合）
Fig. 9 The rule list of `ipfw` used for performance evaluation (for the best case).

れ 192.168.1.1，192.168.1.2，利用者の UID は 10000 である．したがって本実験における通信は，いずれの場合でも図 7〜図 10 の最後の行が適用される．なお，ipfw における最遅の場合には，まず 1 番から 10000 番までのルールを順に評価し，10000 番のルールで初めて UID が一致するため skipto の動作に従って 20001 番のルールから評価を再開し，最終的には 20010 番のルールが適用される．

実験結果を**表 1** に示す．この表より以下のことが分かる．

まず最速の場合では，ipfw がカーネル内で動作するのに対して，purifier はユーザプロセスとして動作するため，TCP，UDP 共に purifier のほうが若干オーバヘッドが大きい．しかしその差は TCP では 3.4 ms，UDP では 2.8 ms とほとんど無視することができる値

```
0 allow tcp from any to any established
1 skipto 10001 ip from any to any uid 1
2 skipto 10001 ip from any to any uid 2
3 skipto 10001 ip from any to any uid 3
...
9999 skipto 10001 ip from any to any uid 9999
10000 skipto 20001 ip from any to any uid 10000
10001 deny ip from 192.168.0.0 1 to 192.168.0.1
10002 deny ip from 192.168.0.0 2 to 192.168.0.1
...
10009 deny ip from 192.168.0.0 9 to 192.168.0.1
10010 deny ip from 192.168.0.0 10 to 192.168.0.1
10011 deny ip from any to any
20001 deny ip from 192.168.0.0 1 to 192.168.0.1
20002 deny ip from 192.168.0.0 2 to 192.168.0.1
...
20009 deny ip from 192.168.0.0 9 to 192.168.0.1
20010 allow ip from 192.168.1.1 to 192.168.1.2
```

図 10 ipfw のルールリスト（最遅の場合）
Fig. 10 The rule list of `ipfw` for performance evaluation (for the worst case).

表 1 アクセス制御手法による通信時間の比較
Table 1 Comparison of communication time with different access control methods.

通信時間 (ms)		アクセス制御手法	
		ipfw	purifier
TCP	最速	58.5	61.9
	最遅	222.1	63.2
UDP	最速	51.6	54.4
	最遅	239.0	54.6

であり，実用上問題にならないといえる．一方，最遅の場合では ipfw では TCP，UDP の通信時間がそれぞれ 3.8 倍，4.6 倍と大幅に増加したのに対し，purifier では，TCP，UDP ともほとんど無視できる程度の増加にとどまり，ipfw に比べてオーバヘッドを大きく削減する効果が確認できる．

以上の考察から，登録利用者数の多い教育用計算機環境では，ipfw はオーバヘッドが大きくなり実用上問題があるが，purifier はこのような環境でも十分小さなオーバヘッドでアクセス制御を行うことができ，実用的であるといえる．

4.2.3 利用者制御ファイルキャッシュ機能の評価

次に，利用者制御ファイルのキャッシュ機能の有効性を評価する実験を行った．この実験では，TCP，UDP のいずれについても最も利用者制御ファイルが大きい最遅の場合において，キャッシュ機能を使用した場合と使用しない場合の通信時間を比較した．その際，UDP についてはパケットごとに利用者制御ファイルにアクセスさせるため，UDP フローキャッシュ機能を用いないように設定した．なお，本機能を使用しない場合でも OS が有するファイルキャッシュ機能は有効であ

表 2 利用者制御ファイルキャッシュ機能の有無による通信時間の比較
Table 2 Comparison of communication time between with and without user control file cache mechanism.

通信時間 (ms)	キャッシュ機能	
	使用	不使用
TCP	63.2	69.4
UDP	55.1	62.8

るため，実際にはディスクへのアクセスはたかだか 1 回しか発生せず，事実上無視できる．

実験の結果を**表 2** に示す．この表より，キャッシュ機能により TCP では 9%程度，UDP では 12%程度通信時間が減少し，キャッシュ機能の有効性が認められる．

4.2.4 UDP フローキャッシュ機能の評価

最後に，UDP フローキャッシュ機能について，その有効性を評価する実験を行った．この実験でも前項と同様に最遅の場合において同機能を使用した場合と使用しない場合の通信時間を比較した．その際，利用者制御ファイルキャッシュ機能は有効となるように設定した．

実験の結果を**表 3** に示す．この表より，UDP フローキャッシュ機能の有無では性能にほとんど違いがないことが分かる．したがって，3.4 節で述べたアクセス制御が無効になる危険性を考慮すると，この実験環境では UDP フローキャッシュ機能を用いないほうが望ましい．

しかし，この実験で用いた FreeBSD では利用者情報の取得に `sysctlbyname()` システムコールを用いてカーネルメモリ空間を直接検索できるためオーバヘッドが小さく，このようなシステムコールを有していない他の OS ではオーバヘッドが大きいことが懸念される．そこで，別の計算機（Sun Microsystems 社 Sun Enterprise 3500，CPU: 400 MHz UltraSPARC-IIs（8 MB）× 4，主記憶容量：4 GB，OS: Solaris 2.6）上で pident を動作させ，その中の利用者情報取得部分の実行時間を測定した結果，1 回の情報取得につき約 10 ms と比較的長い時間を要することが判明した．この結果より，OS によっては UDP フローキャッシュ機能が性能改善に大きな効果を持つと思われる．

UDP フローキャッシュ機能を利用する場合には，3.4 節で述べたようにキャッシュの有効時間が問題となる．これはアクセス制御無効状態による危険性と性能改善効果の両方を勘案して決定されるべきであるが，1 つの目安として 1 秒程度が妥当と思われる．この場合，UDP 通信が連続して行われると，1 秒のうちの

表 3 UDP フローキャッシュ機能の有無による通信時間の比較
Table 3 Comparison of communication time between with and without UDP flow cache mechanism.

通信時間 (ms)	キャッシュ機能	
	使用	不使用
UDP	54.6	55.1

10 ms が利用者情報取得のオーバヘッドにより通信を行えず，残りの 990 ms で通信が行われるため，利用者情報取得のオーバヘッドがない場合と比較して性能劣化は 1% 程度に抑えられると推測される．

5. ま と め

本論文では，特に大規模なマルチユーザシステムにおいて利用者単位でのアクセス制御を効率的に実現する手法を提案した．また提案手法に基づいたアクセス制御システムを試作し，試作システムの性能評価実験を通して提案手法の有効性を確認した．今後の課題としては，利用者の所属グループに基づいたアクセス制御や ICMP（たとえば一部の利用者にのみ ICMP ECHO 送出を許可する）などの他のプロトコルへの対応，適応的なルールの導入による更に柔軟なネットワークアクセス制御の実現などがあげられる．

参 考 文 献

1) 山井成良，山外芳伸，宮下卓也，松浦敏雄：WWW クライアントを対象とした利用者単位のアクセス制御機構，情報処理学会論文誌，Vol.43, No.11, pp.3489–3499 (2002).
2) Reed, D.: IP Filter — TCP/IP Firewall/NAT Software. http://coombs.anu.edu.au/ipfilter/
3) Antsilevich, U.J.S., Kamp, P.-H., Nash, A., Cobbs, A. and Rizzo, L.: ipfw — IP firewall and traffic shaper control program, FreeBSD System Manager's Manual (2005).
4) Cobbs, A.: divert — kernel packet diversion mechanism, FreeBSD Kernel Interfaces Manual (2004).
5) St. Johns, M.C.: Identification Protocol, RFC1413, IETF (1993).
6) Eriksson, P.: Pidentd. http://sf.www.lysator.liu.se/~pen/pidentd/
7) Sollins, K.R.: THE TFTP PROTOCOL (REVISION 2), RFC1350, IETF (1992).

(平成 17 年 7 月 12 日受付)
(平成 18 年 2 月 1 日採録)

山井 成良（正会員）

昭和 59 年大阪大学工学部電子工学科卒業．昭和 61 年同大学大学院博士前期課程修了．昭和 63 年同大学院基礎工学研究科（物理系専攻情報工学分野）博士後期課程退学．同年奈良工業高等専門学校情報工学科助手．同講師，大阪大学情報処理教育センター助手，同大学大型計算機センター講師を経て，現在，岡山大学総合情報基盤センター助教授．分散システム，マルチメディアシステム，マルチメディアネットワークの研究に従事．IEEE，電子情報通信学会各会員．博士（工学）．

眞鍋 宏隆

平成 15 年岡山大学工学部情報工学科卒業．平成 17 年同大学大学院自然科学研究科博士前期課程修了．同年キヤノンシステムソリューション株式会社入社．通信セキュリティ等に興味を持つ．

岡山 聖彦（正会員）

平成 2 年大阪大学基礎工学部情報工学科卒業．平成 4 年同大学大学院基礎工学研究科博士前期課程修了．同年同大学院基礎工学研究科博士後期課程を退学し，同大学工学部助手．平成 6 年奈良先端科学技術大学院大学情報科学研究科助手．平成 10 年岡山大学工学部助手．平成 17 年同大学総合情報基盤センター助手．博士（工学）．インタネットアーキテクチャ，ネットワーク管理，ネットワークセキュリティの研究に従事．電子情報通信学会会員．

宮下 卓也（正会員）

平成 3 年岡山大学工学部電気電子工学科卒業．平成 5 年同大学大学院工学研究科（電気電子工学専攻）修了．平成 8 年同大学院自然科学研究科（知能開発科学専攻）修了．平成 9 年東京農工大学ベンチャービジネスラボラトリー博士研究員．平成 10 年岡山大学総合情報処理センター助手．平成 16 年同大学総合情報基盤センター助手．平成 17 年津山工業高等専門学校情報工学科助教授．ディジタル機器からの放射電磁雑音の計測・予測・抑制，分散システム，ネットワークセキュリティの研究に従事．博士（工学）．IEEE，電子情報通信学会，エレクトロニクス実装学会各会員．

松浦 敏雄（正会員）

昭和 50 年大阪大学基礎工学部情報工学科卒業．昭和 54 年同大学大学院基礎工学研究科（情報工学専攻）博士後期課程退学後，同年大阪大学基礎工学部情報工学科助手．平成 4 年同大学情報処理教育センター助教授．平成 7 年大阪市立大学生活科学部教授．平成 8 年同大学学術情報総合センター教授．平成 15 年同大学院創造都市研究科教授，現在に至る．工学博士．ソフトウェア開発環境，ユーザインターフェイス，マルチメディア，情報教育等に興味を持つ．ACM，IEEE，電子情報通信学会各会員．