IPv6 における TCP パケットを利用した輻輳検出

S073085　村澤太郎

島根大学総合理工学部 数理・情報システム学科

計算機科学講座　　田中研究室

2012 年 3 月 10 日

目次

第 1 章　はじめに　4

第 2 章　輻輳検出とその必要性　5

2.1　輻輳とは　5
2.2　輻輳検出の必要性　5
2.3　従来の方法　6

第 3 章　TCP 制御フラグによる輻輳検出　8

3.1　トランスポート層について　9
3.2　TCP プロトコル　9
3.3　TCP 制御フラグによる輻輳検出　9
3.4　ACK を用いた検出原理　10
3.5　重複 ACK　11
3.6　重複 ACK 検出方法　12

第 4 章　IPv6　14

4.1　インターネット層について　14
4.2　IPv4 プロトコル　15
4.3　IPv6 プロトコル　15
4.4　IPv6 での重複 ACK　16

第 5 章　実証実験　18

5.1　実験内容　18
5.2　実験環境　19
5.2.1　IPv4 ネットワーク　19
5.2.2　IPv6 ネットワーク　20
5.2.3　NAT-PT を利用した IPv4/IPv6 ネットワーク　21
5.3　実験方法　22
5.3.1　平常時の測定方法　22
5.3.2　輻輳発生時の測定方法　23
5.4　Iperf の使用方法　25
5.5　輻輳検出ソフトウェア概要　29

第 6 章　実験結果と考察　31

6.1　実験結果　31

6.1.1　IPv4 ネットワークの平常時・負荷時の結果　31

6.1.2　IPv6 ネットワークの平常時・負荷時の結果　34

6.1.3　トランスレータネットワークの平常時・負荷時の結果　37

6.2　実験結果まとめ　39

6.3　考察　39

6.4　まとめと展望　40

6.5　謝辞

参考文献　42

付録　43

第 1 章　　はじめに

現在最も普及しているプロトコルスタックである TCP/IP ネットワークでは、輻輳制御は TCP プロトコルによって自動的に行われており、一般的にユーザーがかかわることは無い。また、ユーザーから輻輳状況を確認する方法も確立されていなかった。

しかし、ネットワーク上の輻輳状態という情報は、ネットワークアプリケーションからは非常に有益な情報である。なぜなら、輻輳状況に応じて転送量を増大または減少させるなど柔軟に対応することができたり、ネットワーク上のトラブルを早期に発見することができるからである。

当研究室では以上のような理由から、以前からTCPパケットをキャプチャし、その制御フラグの変化を見ることによって輻輳状態を把握する研究を行ってきた。この方法を利用すれば、ネットワーク上の輻輳状態を、一般的には把握することができないユーザーが、簡単に知ることができる。

先行研究では一般的な TCP/IP ネットワークである IPv4 ネットワークと、その上位層であるアプリケーション層のプロトコル別に実験が行われた。本研究では、先行研究をもとに、今後普及が見込まれる IPv6 ネットワークにおいて TCP パケットを利用した輻輳検出が可能であるか、実験を行い検証した。

第 2 章　　輻輳検出とその必要性

2.1　　輻輳とは

通信における輻輳とは、アクセスが特定の宛先に集中することにより、通常行えるはずの通信ができなくなる状況のことを指す。道路上での交通渋滞がイメージしやすい。一般的な例として人気のコンサートチケットの予約の電話が一斉にかけられ、電話がつながりにくくなったりすることや、インターネット上などで一つのファイルのダウンロード要求が多数発生し、ダウンロードに時間がかかったりすることなどが挙げられる。

TCP/IP ネットワークでの輻輳は、サーバーやクライアントのコンピュータの性能や負荷の状態、中継するルーターのパケット転送性能などによって変化するため、事前に輻輳を予測することは難しい。

2.2　　輻輳検出の必要性

大勢の人が同時に回線を利用するなどで、ネットワーク上のトラフィックが増大し輻輳状態になった場合、早期に発見する必要がある。なぜならば、一度輻輳が発生すると、それらが引き金となって輻輳が悪化する可能性があるからである。

トラフィックを自動車と例えると玉突き事故のようなものであり、1 つの事故によって後続の自動車が次々と事故を誘発させ、収集がつかなくなることと類似している。

ネットワーク上の輻輳によってトラフィックの遅延が発生し、パケットの再送要求が大量に出され、それによってさらにネットワークの状態が悪くなるという悪循環が起きる可能性がある。そのために輻輳が発生したとしてもそれを早期に検出し、通信量を制御することによって輻輳状態を解消していくことが大切となってくる。そのために現在はさまざまな輻輳の検出手法が考えられている。

図 2.1　輻輳検出の必要性

2.3　従来の方法

ネットワーク上の混雑に対応するため、輻輳検出の一例を紹介する。(参考文献[1][2][3])

[ネットワークの状態推測機構]

ネットワークの情報は直接手に入らないのでエンド間の通信から得られる情報で推察する。単純なアルゴリズムとしてはパケットの受信数が落ちないなら転送量をあげそうでないと下げるというものがある。

[スロースタートアルゴリズム]

通信を開始するときに控えめに転送をはじめ、徐々に転送速度を上げていく。長所としてはネットワークの突発的なトラフィック流入を避けることができるが、その反面短所として、転送速度が収束するのに時間がかかる。

輻輳が起きる直前のウィンドウサイズを A、輻輳が起きたときのウィンドウサイズを B とすると、最適なウィンドウサイズ X は

A ＜ X ＜ B

と推測される。ここではウィンドウサイズをいったん下げ、再度増加させながら輻輳が発生しない最適な X を探る。

[RTTの相関関係を利用]

RTT(Round Trip Time)か分かることは、輻輳の度合いである。パケットロスの指標であり、RTT から得られる指標を平滑化することによって RTT を評価し(RTT 表価値)輻輳を検出する。RTT はネットワークのトラフィック量に従って時間とともに変化するので常に適切な値に補正しなければならない。ここでは ACK の送信から受信までにかかる時間を測定する式を示す。

RTT 評価式(RFC793 準拠)

$R = \alpha R + (1 - \alpha) M$

R:RTT 評価値

M:新しい測定値

α:平滑化係数(推奨値は 0.9)

[回線利用率からの推測]

ネットワーク間における、ネットワークトラフィックから利用率を測定し輻輳状態の推測を行う。回線利用率は

回線利用率 = 1 時間に発生する総データ量(Bit) / 回線容量(Bit)*時間

で計算される。

これらの手法は、TCP プロトコル上で行われる手法であるため、ユーザーが目に触れることは無い。そのため、本研究での検出方法は、本来ユーザーからは検出することが難しい輻輳状態を、ソフトウェアによって検出するためのものであり、その実用化を試行していきたい。

第 3 章　　TCP 制御フラグによる輻輳検出

3.1　　トランスポート層について

インターネットにおける TCP/IP プロトコルスタックは、4 つの層に分かれており(図 3.1)、いちばんユーザーに近い層からアプリケーション層、トランスポート層、ネットワーク層(インターネット層)、リンク層となっている。ネットワーク上で通信を行う場合、それぞれの層において通信を行うプロトコルを選択し、各ノード間で通信を行うのである。(参考文献[3][4])

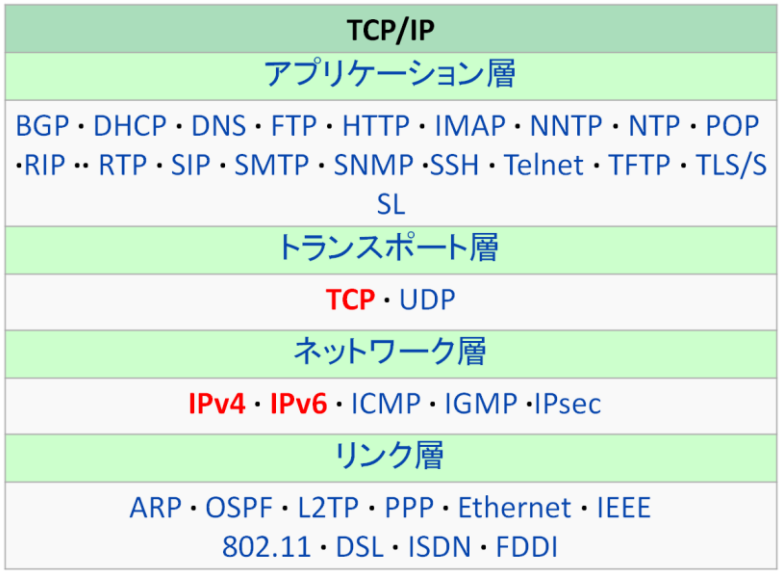


図 3.1　TCP/IP プロトコルスタックの例

4 つの層の 1 つであるトランスポート層では、TCP/IP モデルにおいて根底を成す「TCP プロトコル」と「UDP プロトコル」が存在する。

特にトランスポート層において輻輳制御を担うのが TCP プロトコルであり、エラー発生時の再送制御機能やデータの順序制御、データを効率よく転送するためのウインドウ制御、ネットワーク混雑度合いに応じて送信するデータ量を調整するフロー制御など、TCP/IP モデルの信頼を高めるための重要なサービスを行っている。

また UDP プロトコルは、TCP プロトコルのような輻輳制御などの複雑な仕組みを持っていないが、輻輳によるパケット破棄などがあまり問題にならない動画や音声のストリーミング通信や、常時通信し続ける必要のあるオンラインゲームなどで使用されるプロトコルである。

次からは、本研究の内容である TCP プロトコルについて説明を行っていく

3.2 TCP プロトコル

TCP プロトコルはコネクション型プロトコルであり、信頼性のあるデータ転送を行い適切なアプリケーションへデータを送り届けることができる。コネクション型プロトコルとはデータ転送を行うにあたって、3 ウェイハンドシェイクと呼ばれるコネクションを確立する手順を行うことを指す。

トランスポート層プロトコルに TCP を利用するかどうかは、アプリケーション層のプロトコルが決定する。TCP をトランスポート層プロトコルに利用するのは、HTTP や SMTP、FTP などがある。これらのプロトコルが取り扱うデータは、途中で失われてしまったりすると、送信先でデータが再構築できなくなってしまう。また、サイズが大きくなることがある。途中で失われたりしないようにするため、そして、サイズが大きいデータでも正確に転送するために、信頼性のある転送プロトコルである TCP を利用するのである。(参考文献[3][4])

3.3 TCP 制御フラグによる輻輳検出

今回の研究ではトランスポート層に位置する TCP を使用し、そのプロトコルに含まれる制御フラグの情報をもとにネットワークの輻輳の検出を行う。(以下 3.6 まで参考文献[1][2])

TCP プロトコルは図 3.2 のようなヘッダで構成されている。

図 3.2　TCP ヘッダ

中でも注目すべきは、コードビットと呼ばれる 6bit のフラグである。これらは制御フラグと呼ばれており、ネットワークの接続の確立や切断、応答を示すための情報を含み、それらのフラグの情報を元にしてネットワーク接続の信頼性を確立している。

制御フラグはネットワーク中にさまざまな形で存在しているので、TCP トラフィック中に TCP 制御フラグがどのような割合で含まれており、また輻輳状態になったときにどのように変化をしていくのかをキャプチャリングする。

輻輳検出手法はいくつか存在するが、この制御フラグを使った輻輳検出手法のメリットとしては、従来に検出手法にあるような計算式は使わずに、単純に単位時間あたりの制御フラグの個数を数え上げることにある。そして、制御フラグの出現比率などを測定することによって、輻輳を導き出す。検出する制御フラグは以下のとおりである。

URG	ビットが 1 の場合、緊急データを含んでいることを示す
ACK	確認応答を行うセグメントであることを示す
PSH	受け取ったデータをすぐにアプリケーションに渡すことを示す
PST	コネクションの強制終了を要求する
SYN	コネクションの確立のときに使用される
FIN	コネクションの正常終了を要求する

表 3.1　制御フラグの解説

制御フラグの中の、ACK フラグとそれに付随される応答確認番号を利用して輻輳検出を行った。

3.4　ACK を用いた検出原理

ACK とは肯定確認応答のことであり、送信されたパケットが確実に送信されているかということを確認するためのフラグである。このフラグの特徴として送信側から送られる最初のコネクション開始要求以外のすべてのパケットに付属するということである。また、送信された ACK が正しいのかどうかを判断する基準としては TCP ヘッダに付属している応答確認番号を利用する。

ACK フラグが立っているということは、応答確認番号が有効であるということである。そして、応答確認番号とは次に受信すべきシーケンス番号を表している。つまりパケットをやり取りするたびに応答確認番号は変化するのである。

このメカニズムは次のように説明される。

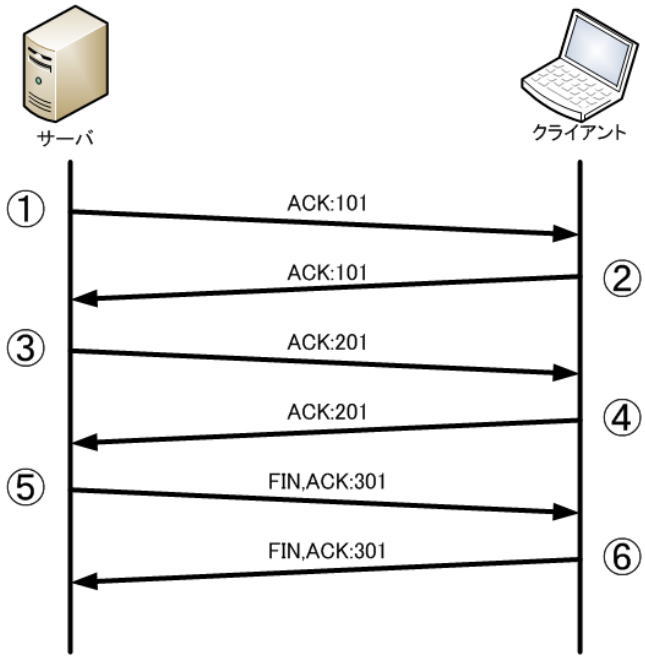


図 3.3　ACK フラグ

① まず、送信側から ACK101 が送られる。101 は応答確認番号である
② 受信側が ACK を受け取ると送信側に受け取った旨として同じ番号を送り返す
③ 送信側はつづけて番号を更新し ACK201 を送り出す。
④ 受信側も受け取った番号を返す。
⑤ パケットの最後になると終了の合図である FIN と同時に ACK301 を送る
⑥ 同じように FIN と ACK301 を返す。

3.5　重複 ACK

TCP プロトコルによる通信の場合、ほぼすべてのパケットの ACK フラグが付属し、確認応答番号は受け取ったパケットによって増加していく。しかし、受信側から受け取る確認応答番号が通常と違う場合がある。それは輻輳などによるパケットの損失の場合だ。

パケットの損失が発生し、通常受け取るはずのパケットが受信できなかった場合は、受信側から受け取ったということを意味する ACK を送信することはない。その状態でも送信側は次々とパケットを送信し続けるため、応答確認番号が正常な順序にならなくなる。そして、受信側はまだ受け取っていないパケットが把握できるため、受け取っていない応答確認番号のパケットを送信するのである。

このとき送信されるまだ受け取っていない応答確認番号のパケットは、送信側が送信するまで生成され続けるため、同じ内容のパケットが複数確認できる。このため、このパケットを本研究では特別に**重複 ACK** と呼ぶ。

重複 ACK が発行されると、それだけパケットの損失が発生していることであるため、失われたパケットの指標となりうると考えられる。またそのパケット損失率が高ければ高いほどネットワークが混雑していると考えられ、エンドノードの輻輳を計るひとつの要因になるのではないかと考えられる。

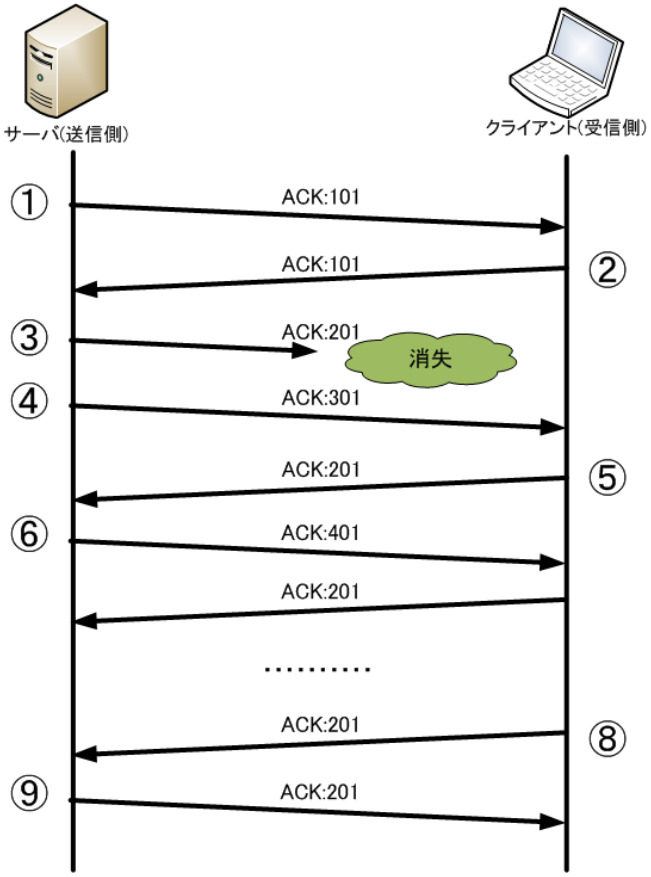


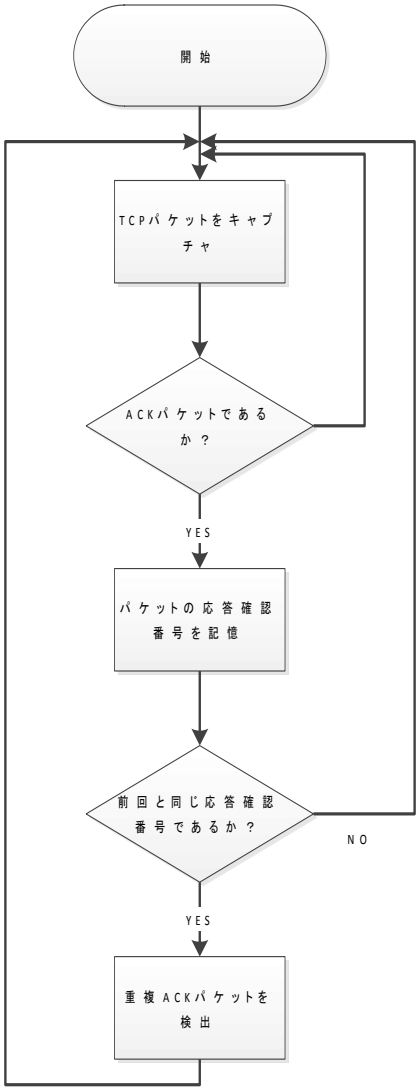
図 3.4　重複 ACK

① まず、送信側から ACK101 が送られる(101 は応答確認番号)
② 受信側が ACK を受け取ると送信側に受け取った旨として同じ番号を送り返す
③ 送信側はつづけて番号を更新し ACK201 を送り出すがパケットが消失。
④ 続けて送信側が ACK301 を送信
⑤ このとき受信側では ACK301 を先に受信するので ACK201 が受信できていないこと分かる。そのため ACK201 を再送信するように再送要求を出す。
⑥ 受信側が ACK 番号を間違っている可能性もあるので様子を見るために次の番号を送る。
⑦ ・⑧届いていない番号を送りつづける。
⑧ 何度送っても同じ答えが返ってくるので ACK201 を再送する。

3.6　　重複 ACK 検出方法

輻輳などが原因で重複 ACK が生成される場合があるが、この重複 ACK を計測できれば、ネットワークの混雑状態を把握することができると考えられる。

重複 ACK の検出方法のアルゴリズムを以下に記す。



1.アルゴリズム開始。

2.重複 ACK であるか確認するために TCP パケットをキャプチャ。

3.ACK フラグ付のパケットであるか分岐する。YES なら次へ、NO なら TCP パケットキャプチャまで戻る

4.パケットの応答確認番号を記憶する。なおここで記憶される応答確認番号は最新のもののみであり、順次更新される。

5.新しく取得された応答確認番号と前回取得した応答確認番号を比較する。同じものであるなら YES へ、違うものならパケットキャプチャまで戻る。

6.取得したパケットは重複 ACK であることが分かり、輻輳状態であると考えられる。そして TCP パケットキャプチャまで戻る。

図 3.5　　アルゴリズム

このアルゴリズムは、前回と同じ応答確認番号のパケットを受信した場合輻輳であるとみなしている。再送要求の応答確認番号を比較するだけの単純なアルゴリズムのため、動作は比較的軽いと思われる

この方法は先行研究(樋上智彦 2004)によって有効性が示された輻輳検出方法である。

第 4 章　　IPv6

前章までは主に TCP プロトコルでの重複 ACK 検出方法を説明してきた。この章では、本研究の最大のテーマである IPv6 での重複 ACK の振る舞いについて説明していく。(以下参考文献[3][4])

4.1　　インターネット層について

3.1 では TCP/IP プロトコルスタックの 4 層について説明したが、ここではトランスポート層の下位層であるインターネット層について説明する。

インターネット層は TCP/IP の 4 階層モデルで、下から 2 番目に位置する層である。インターネット層は、ホストからホストの通信を行う階層であるため、その通信のために「IP アドレス」が使われる。インターネット内で固有の IP アドレスをホストに割り当てることで、ネットワークの種類に関係なく、世界中のどの場所からでも通信したい相手ホストを特定できる。インターネットで用いる IP アドレスはグローバル IP と呼ばれ、インターネット上で同じ IP アドレスを持った端末は存在しない。

また、IP ではデータを「パケット」という単位に分割する。これは、ネットワークに回線が 1 本しかなかった場合でも、複数のホストによる通信が同時にできるようにするための仕組みである。データをパケットに分割することで、多数のホストのパケットが回線を共有できるようになる。

IP には大きく分けて現在主流である IPv4 プロトコルと、IPv6 をベースにアドレス空間を増大した IPv6 が存在する。

図 4.1　インターネット層について

4.2 IPv4 プロトコル

インターネット層では主に IPv4 が使用されている。TCP/IP プロトコルスタックにおいて、またインターネットの発展に重要な役割を担ってきた。IP と呼ばれるのは一般的にこの IPv4 のことを指す。

IPv4 には IP アドレスと呼ばれる 32 ビットで表現されたアドレスを持っており、このアドレスを用いて通信を行う。インターネット上で用いられるのはグローバル IP アドレスと呼ばれ、LAN 内で使用されるものはプライベート IP アドレスと呼ばれる。このうちグローバル IP アドレスは IPv4 アドレス枯渇問題に直面しており、新しく発行できる IP アドレスが残り少ないのである。そのため、IPv4 アドレスをベースにアドレス空間を広げた IPv6 に順次切り替えられていく予定である。

IP アドレスは 32 ビットであるため約 43 億個のアドレス空間となる。かなりの量に見えるが世界中のコンピュータがそれぞれに IP アドレスを利用しており、また個人では一人で何台もコンピュータや携帯電話、スマートフォンを利用している場合もあるため足りないのである。

一方プライベート IP アドレスは、インターネットの接続には使用できず、LAN 内で使用される。それぞれの LAN はクラスに応じたプライベートアドレスを利用しており、インターネットに接続するためには NAT などを利用してプライベートアドレスをグローバルアドレスに変換しなければならない。

クラス	範囲	サブネット	アドレス数
クラス A	10.0.0.0-10.255.255.255	255.0.0.0	16,777,216
クラス B	172.16.0.0-172.31.255.255	255.240.0.0	1,048,576
クラス C	192.168.0.0-192.168.255.255	255.255.0.0	65,536

表 4.1　プライベートアドレス

4.3 IPv6 プロトコル

本研究での最大のポイントである IPv6 アドレスについて説明する。

前述の IPv4 はグローバル IP アドレスの枯渇問題があると述べた。これはアドレス空間が 32 ビットと少ないからである。そのアドレス空間を 128 ビットに増やし、約 340 澗(340 兆の 1 兆倍の 1 兆倍)という事実上無限大のアドレスが利用できるようにしたものが IPv6 である。

他にも、管理者に負担をかけない IP アドレスの自動設定やアドレスの集約による基幹ルーターでの経路表サイズの抑制、固定長ヘッダなどの特徴がある。

今後インターネットでは順次 IPv6 アドレスに切り替わっていく予定である。すでに IPv6 を、グローバル IP アドレスとして払いだしている ISP も存在している。

なお、IPv6 にはプライベートアドレス・グローバルアドレスという違いは無い。

	IPv4	IPv6
アドレス長	32 ビット	128 ビット
アドレス空間	約 43 億個	約 340 澗個
ヘッダサイズ	20～60 バイト 可変長	40 バイト 固定長
最小 MTU 値	規定なし	1280 バイト
フラグメント	ホスト ルーター	ホスト
L2 アドレス解決	ARP	ICMPv6
ブロードキャスト	あり	なし
設定	手動 DHCP	手動 自動 DHCPv6

表 4　IPv4 と IPv6 の比較

IPv4 は人間が扱いやすいように 10 進数の 12 ケタで表現される。IPv6 ではアドレス空間が増えたため 10 進数では桁数が多くなりすぎるので、16 進数の 32 ケタで表現される。

一般的な IPv6 アドレス

[例 1]2001:0db8:bd05:01d2:288a:1fc0:0001:10ee

各セクションが 0 で始まる場合、省略できる

[例 2]2001:0db8:0020:0003:1000:0100:0020:0003 = 2001:db8:20:3:1000:100:20:3

0 が連続する場合も省略可能

[例 3] 2001:0db8:0000:0000:1234:0000:0000:9abc = 2001:db8::1234:0:0:9abc
[例 4] 2001:0db8:0000:0000:0000:0000:0000:9abc = 2001:db8::9abc

4.4　IPv6 での重複 ACK

ここまでは TCP や IP の説明を行ってきたが、次は IPv6 を使用した場合の TCP での重複 ACK の振る舞いを考えていきたいと思う。

先行研究では、一般的なネットワークを想定した実験を想定してインターネット層には IPv4 を使用している。本研究では、先行研究をもとに今後普及が予想される IPv6 での重複 ACK を用いた輻輳検出について行う。

理由として IPv4 アドレス枯渇問題が挙げられる。今後 IPv4 アドレスは数が足りないためインターネット上で順次利用できなくなり、IPv6 へ切り替わっていくからである。先行研究で有効性が認められた重複 ACK を用いた輻輳検出は、重複 ACK を計測するだけで容易に輻輳状態が判別できる優れた輻輳検出法であるが、IPv6 での振る舞いしだいによっては今後インターネットでの利用ができなくなるかもしれない。それを検証するために本研究では IPv6 上での振る舞いを検証していく。

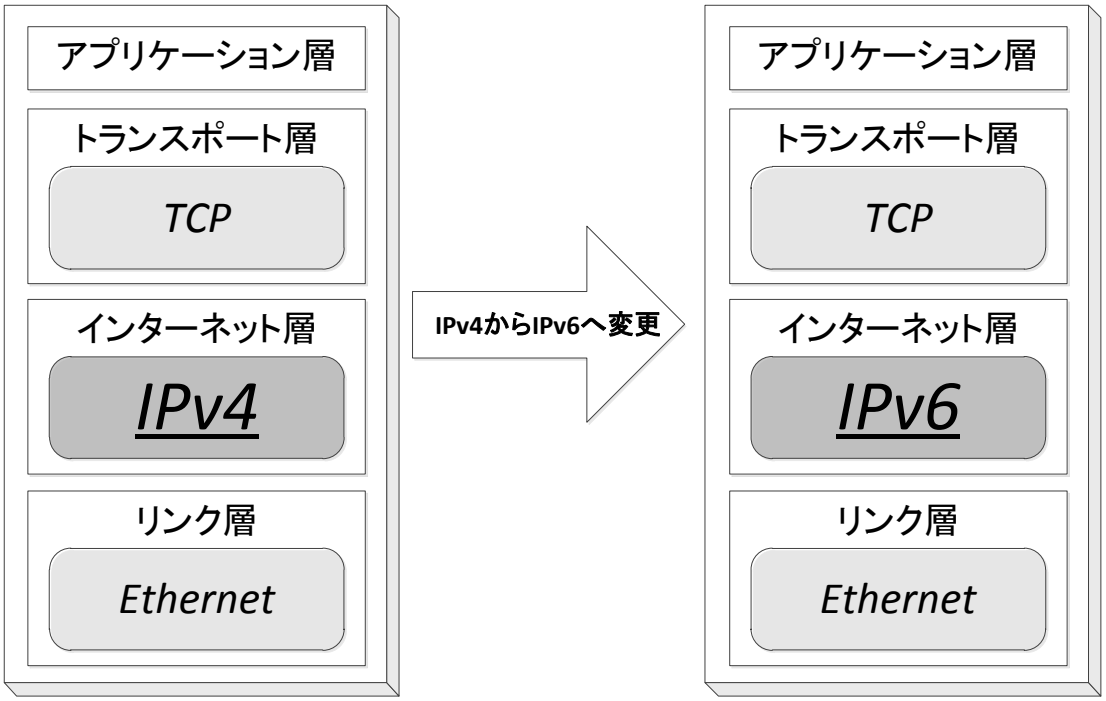


図 4.2　IPv4 から IPv6 への変更

インターネット層を IPv4 から IPv6 に変更して検証していくが、重複 ACK パケットが発生する TCP はトランスポート層であるため、基本的には影響しないと考えられる。TCP/IP は 3.1 で説明したように、それぞれの層に分かれ、たとえ上位、下位のプロトコルが違ったとしてもトラフィックを転送できるからだ。インターネット層が変更されたとしても TCP プロトコルの振る舞いは変化することはないと予想されるため、輻輳状態になったとしても生成される重複 ACK は変わらず検出されると思われる。

つまり、IPv4 から IPv6 にプロトコルを変更し、輻輳発生時に重複 ACK が検出できればこの実験は成功であるといえる。

次の章では、輻輳状態において重複 ACK パケットを検出するための実験方法について解説していく。

第 5 章　　実証実験

5.1　　実験内容

ここでは IPv6 で輻輳発生時に重複 ACK を検出するための実験の解説をする。

実験内容は IPv4 と IPv6 を使用したネットワークを作成し、平常時、輻輳発生時に重複 ACK が検出されるか実験を行う。

作成するネットワークは 3 種類である。

①IPv4 プロトコルを用いたネットワーク
②IPv6 プロトコルを用いたネットワーク
③IPv4 と IPv6 を NAT-PT でトランスレートさせるネットワーク

これらのネットワークの作成するための使用機器は表 5.1 のとおりである

名前	ルーター	Server	Client	パケットキャプチャマシン
OS	IOS 15.1	Windows Server 2008 R2	Windows7 Professional	Windows Server 2008 R2
モデル	Cisco 1812J	NEC Express 5800 110Ge	DELL Vostro 1520	NEC Express 5800 GT110a
CPU	/	Celeron 430 1.8GHz	Celeron 575 2GHz	Celeron 430 1.8GHz
メモリー	256MB	2GB	4GB	1GB
リンク速度	100Mbps			

表 5.1　使用機器一覧

このネットワークにはサーバーとクライアント間で通信した場合のパケットの変化を調べる目的があり、実際にパケットを調査するのはパケットキャプチャマシンと呼ばれる専用のコンピュータである。ネットワークは二つ存在し、そのネットワーク同士をルーターが経路選択を行うことによって通信が可能となっている。

作成したネットワークそれぞれ平常時と輻輳発生時の TCP パケットの変化を記録する。擬似的に輻輳を発生させるために、スループット計測ソフトによってネットワークに負荷をかけている。平常時と負荷発生時のパケットの変化を計測し、輻輳時で重複 ACK パケットの増加が見られれば、重複 ACK パケットを利用した輻輳検出が可能であるといえる。

5.2　　実験環境

5.1 で 3 種類のネットワークを構築すると述べたが、ここではそれぞれのネットワークの解説を行う。それぞれのネットワークのコンフィグは付録に載せた。

5.2.1　　IPv4 ネットワーク

IPv4 ネットワークはインターネット層に IPv4 を使用した一般的なネットワークである。サーバーやクライアント、ルーターの各 NIC には IPv4 アドレスを固定 IP アドレスとして付与している。パケットキャプチャマシンからはパケットを送信することはなく、サーバーの送受信するパケットをすべてミラーリングさせ、モニタリングのみに使用している。パケットキャプチャマシンで重複 ACK パケットを計測する。なおこのネットワークは先行研究によって有用性が確認されたネットワークとほぼ同じである。

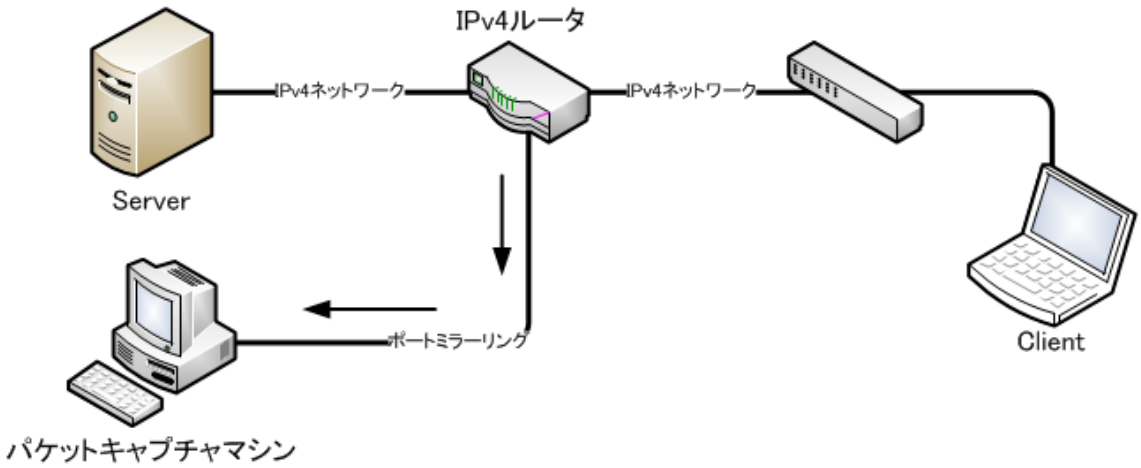


図 5.1　IPv4 ネットワーク

それぞれのマシンの IP アドレスは以下のように設定している。

マシン	IP アドレス/サブネットマスク	ゲートウェイ
ルーター(Server 側)	192.168.10.1/24	/
ルーター(Client 側)	192.168.20.1/24	/
Server	192.168.10.101/24	192.168.10.1
パケットキャプチャマシン	192.168.10.201/24	192.168.10.1
Client	192.168.20.64/24	192.168.20.1

表 5.2　IP アドレス一覧

5.2.2　IPv6 ネットワーク

IPv6 ネットワークではインターネット層に IPv6 を使用し、LAN を構築している。このネットワークは前述の IPv4 ネットワークのインターネット層を IPv4 から IPv6 に変更したのみで、そのほかの構成は変化していない。IPv6 ネットワークでもルーターによる DHCPv6 での自動設定やリンクローカル IP アドレスは使用せず、固定 IP アドレスを使用している。図 10 のようなネットワークを構築し、サーバーの送受信するパケットをすべてパケットキャプチャマシンへミラーリングさせ、輻輳時の重複 ACK の変化を計測する。

このネットワークで輻輳発生時に重複 ACK パケットが検出できれば、重複 ACK フラグを利用した輻輳検出が可能といえる。

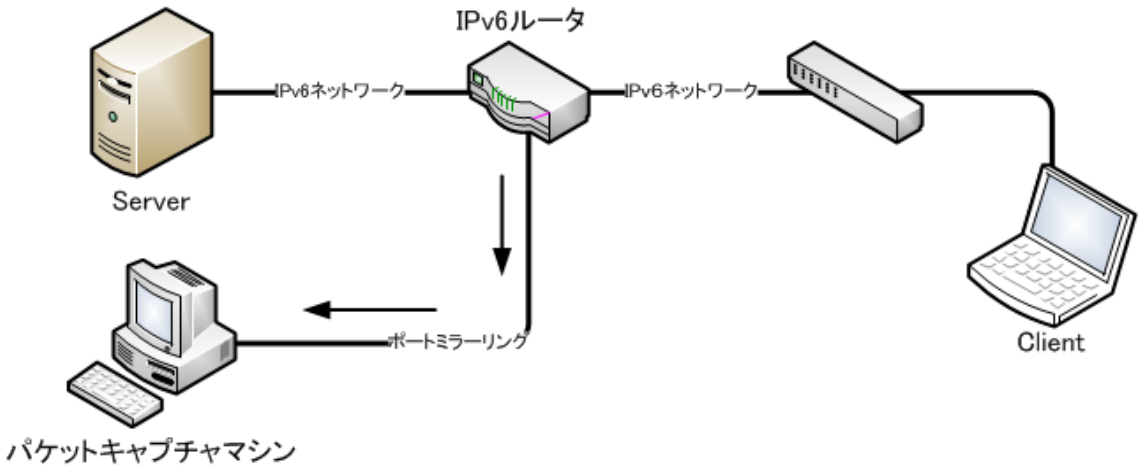


図 5.2　IPv6 ネットワーク

IPv6 アドレスそれぞれの表 5.3 のように設定した。

マシン	IP アドレス/サブネットマスク	ゲートウェイ
ルーター(Server 側)	2001:a:b:c::1/64	/
ルーター(Client 側)	2001:d:e:f::1/64	/
Server	2001:a:b:c::101/64	2001:a:b:c::1/64
パケットキャプチャマシン	2001:a:b:c::201/64	2001:a:b:c::1/64
Client	2001:d:e:f::64/64	2001:d:e:f::1/64

表 5.3　IP アドレス一覧

5.2.3　NAT-PT を利用した IPv4/IPv6 ネットワーク

三番目のネットワークは IPv4 と IPv6 それぞれのネットワークを、NAT-PT と呼ばれるトランスレータとして動作するプロトコルを使用し、通常では通信することができない IPv4 と IPv6 の通信することを可能としたネットワークである。IPv6 ネットワークでの重複 ACK の振る舞いを検証するのであれば 5.2.2 のネットワークでの実験で可能であるが、今回はトランスレータネットワークでも実験を行う。

なぜこのようなネットワークで実験を行うのかというと、今後普及が見込まれる IPv6 ではあるが、移行期にはどうしても IPv4 も併用しなければならない可能性があり、そのような場合通常は IPv4 と IPv6 は通信できないという事態が発生する。しかし、NAT-PT のようなトランスレータプロトコルを使用すれば IPv4 と IPv6 で通信することができ、安心して移行することが可能となる。

NAT-PT とは Network address translation - Protocol translation のことであり、アドレス変換とプロトコル変換を行うものである。これは IPv4 と IPv6 のような異種のプロトコル同士を通信させるための仕組みである。NAT(Network address translation)の動作と似ているが、アドレス自体を全く別のプロトコルのアドレスに変換している。トランスレートした場合は IPv4 側から IPv6 のネットワークは IPv4 ネットワークとして認識され、IPv6 側から IPv4 ネットワークは、IPv6 ネットワークとして認識される。NAT-PT には 4 種類の方法があるが今回はもっとも基本的なスタティック NAT-PT でネットワークを構築する。

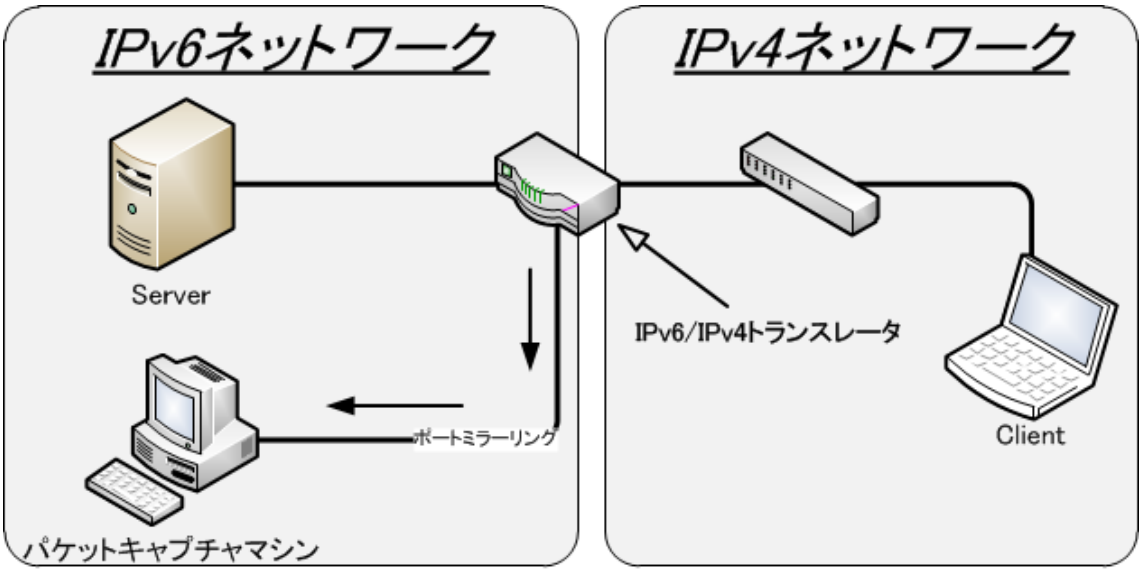


図 5.3　　NAT-PT を利用した IPv4/IPv6 ネットワーク

IP アドレス設定を表 5.4 に記す。

マシン	IP アドレス/サブネットマスク	ゲートウェイ	トランスレート後の IP アドレス
ルーター(Server 側)	2001:a:b:c::1/64	/	/
ルーター(Client 側)	192.168.20.1/24	/	/
Server	2001:a:b:c::101/64	2001:a:b:c::1/64	192.168.20.101
パケットキャプチャマシン	2001:a:b:c::201/64	2001:a:b:c::1/64	/
Client	192.168.20.64/24	192.168.20.1	2001::C0A8:1440

表 5.4　IP アドレス一覧

5.3　実験方法

ここではどうやって輻輳状態での重複 ACK パケットを検出するか解説を行う。

5.2 で説明した 3 つのネットワークにはサーバー、クライアント、パケットキャプチャマシンの 3 つのコンピュータが存在するが、これらのコンピュータを使用して実際に通信を行い、その時のパケットの変化を検出させる。具体的な手順は以下の通りとなる。

5.3.1　平常時の測定方法

平常時はネットワークに負荷をかけずに通信を行い、輻輳が発生しない場合のネットワークをシミュレーションする。計測時間は 10 分間行い、トータルパケット数、ACK パケット数と重複 ACK パケット数を計測する。このネットワークは一般的な LAN⇔LAN の通信をシミュレートした環境であり平常時は輻輳が発生しないようにしている。輻輳検出ソフトとスループット測定ソフト「Iperf」については次の章で説明する。サーバーでパケットキャプチャを行わず、別にパケットキャプチャマシンを準備している理由は、パケットキャプチャプログラムよってサーバに負荷をかけないためである。

① サーバー、クライアント、ルーターを使用しネットワークを構築する。
② パケットキャプチャマシンで重複 ACK 検出ソフトを実行する。ルーターにポートミラーリングの設定を行っているため、サーバーの送受信するパケットがパケットキャプチャマシンにも転送される。
③ サーバー、クライアント間でスループット測定ソフト「Iperf」を実行する。この時のオプションは、あらかじめ調査済みの輻輳を起こさない設定にされている。
④ 10 分間計測を行い、トータルパケット数、ACK パケット数、重複 ACK パケット数、を記録する。

⑤ この方法で IPv4、IPv6、トランスレータネットワークの 3 つのネットワークで計測を行い、輻輳時と比較していく。

平常時の計測方法の図を記す。

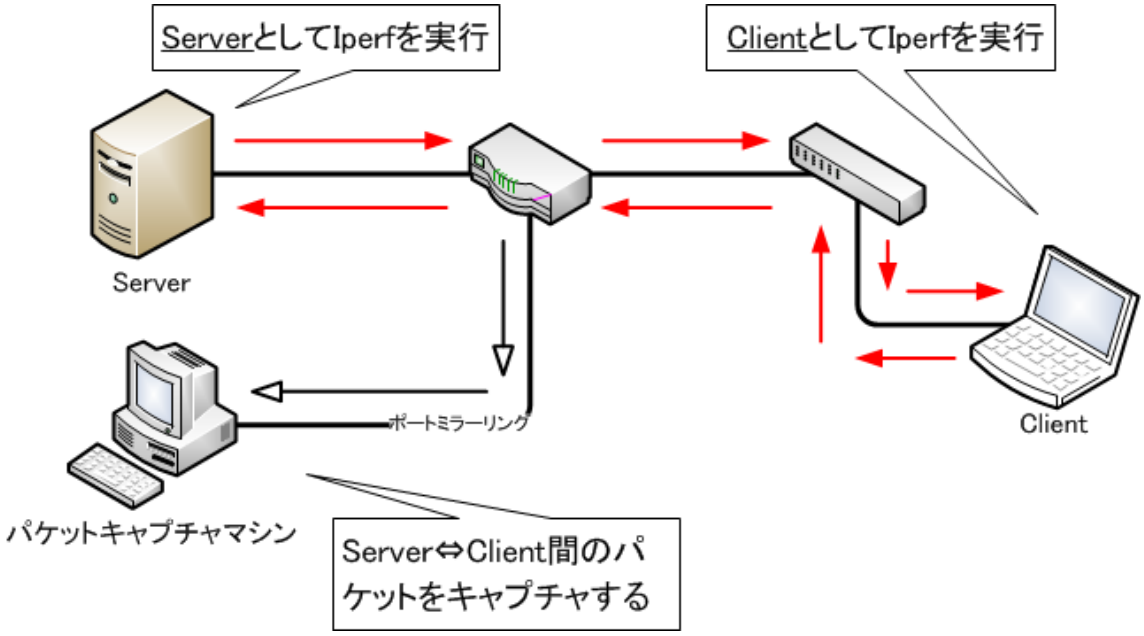


図 5.5　測定方法

5.3.2　輻輳発生時の測定方法

輻輳状態のシミュレーションは、平常時の計測方法とほぼ一緒であるが、擬似で気にネットワークに負荷をかけるために負荷発生ソフトを利用する。負荷発生ソフトには平常時の通信をシミュレートしているものと同じ Iperf を別プロセスで実行させているが、この設定はネットワークに負荷をかけるための特別な設定で動作させている(5.4 参照)。そのため平常時の動作とは全く別物であり、ネットワークにかなりの負荷をかけ、輻輳を引き起こすようにしている。負荷をかけている間も平常時のシミュレートをしている Iperf は実行させ続けている。パケットキャプチャマシンにはサーバーの送受信するすべてのパケットを転送させているので、サーバーの重複 ACK パケットが発生した場合でも検出することができる。

計測時間も平常時と同じ 10 分間であるが、負荷をかけるタイミングは開始 4 分目から 1 分間と開始 8 分から 1 分間である。これは平常時から負荷時へネットワークが変化した際に重複 ACK の変化を確認しやすくするためである。**負荷をかけ輻輳が発生した際に、重複 ACK フラグが確認されれば実験は成功といえる。**輻輳検出ソフトとスループット測定ソフト「Iperf」については次の章で説明する。

測定手順は以下のとおりである。

① サーバー、クライアント、ルーターを使用しネットワークを構築する。
② パケットキャプチャマシンで重複 ACK 検出ソフトを実行する。ルーターにポートミラーリングの設定を行っているため、サーバーの送受信するパケットがパケットキャプチャマシンにも転送される。
③ サーバー、クライアント間でスループット測定ソフト Iperf を実行する。この時のオプションは、あらかじめ調査済みの輻輳を起こさない設定にされている。
④ 10 分間計測を行い、特定の時間帯に負荷をかける。開始 4 分目から 1 分間と開始 8 分から 1 分間で負荷発生用の Iperf を別プロセス実行させる。その間 ACK パケット数、重複 ACK パケット数を記録する。
⑤ この方法で IPv4、IPv6、トランスレータネットワークの 3 つのネットワークで計測を行い、平常時と比較していく。

負荷発生時の重複 ACK フラグの計測方法の図を記す。

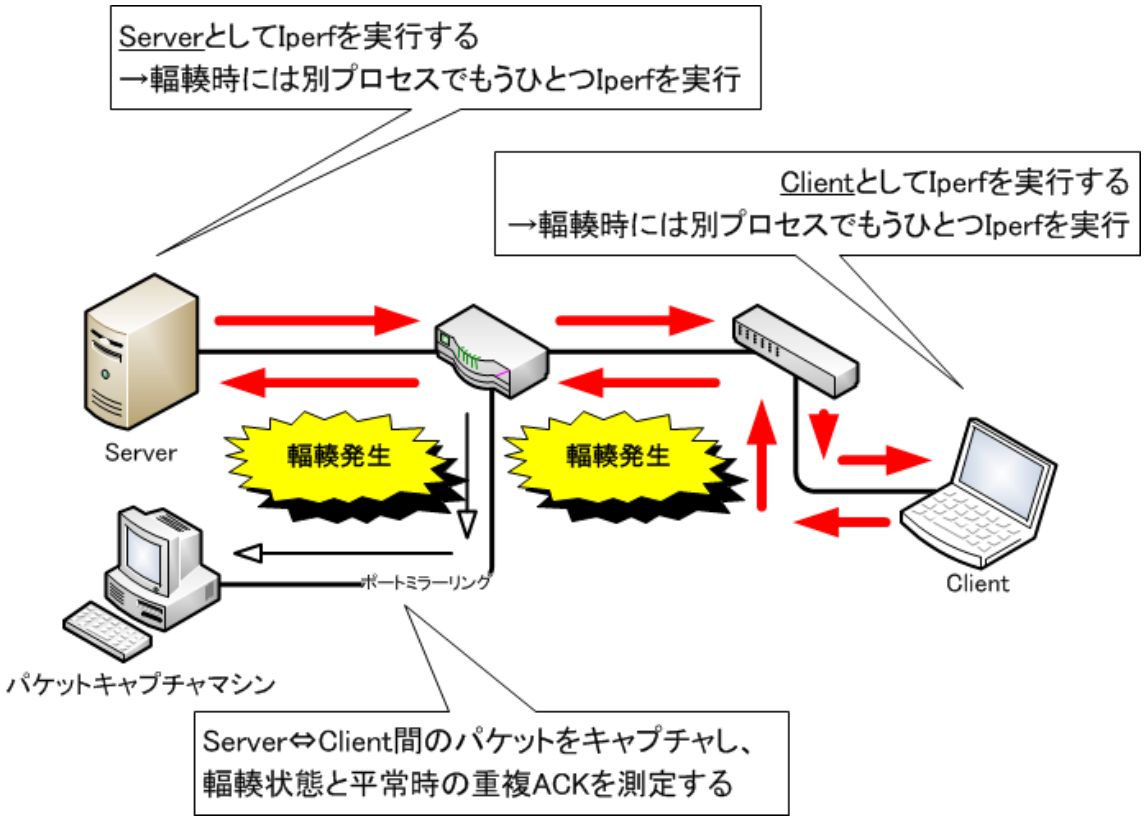


図 5.6　測定方法

5.4　Iperf の使用方法

平常時と輻輳状態のシミュレーションのために Iperf を使用するが、ここでは Iperf について説明していく。

Iperf はサーバークライアント間のスループットを計測するソフトウェアである。しかし設定次第では一般的な通信状態のシミュレートや、擬似的な輻輳を発生させることもできる。サーバーとなるコンピュータとクライアントとなるコンピュータ両方に Iperf を実行させ、TCP プロトコルを利用した通信させる。これによって実験目的の TCP パケットによる重複 ACK を利用した輻輳検出が可能になり、実験を行うことができる。Iperf は便利なソフトウェアだが出力に関する機能が弱いため、Iperf のフロントエンドである Jperf を実験では使用している。これは Iperf に GUI 機能やグラフ出力機能を追加した Windows 上で動作するプログラムである。実際にスループットを計測しているのは Iperf であり、Jperf は補助的なものである。

ソフトウェアは以下のリンクからダウンロードした。

リンク先　http://sourceforge.net/projects/iperf/files/　(sourceforge.net)

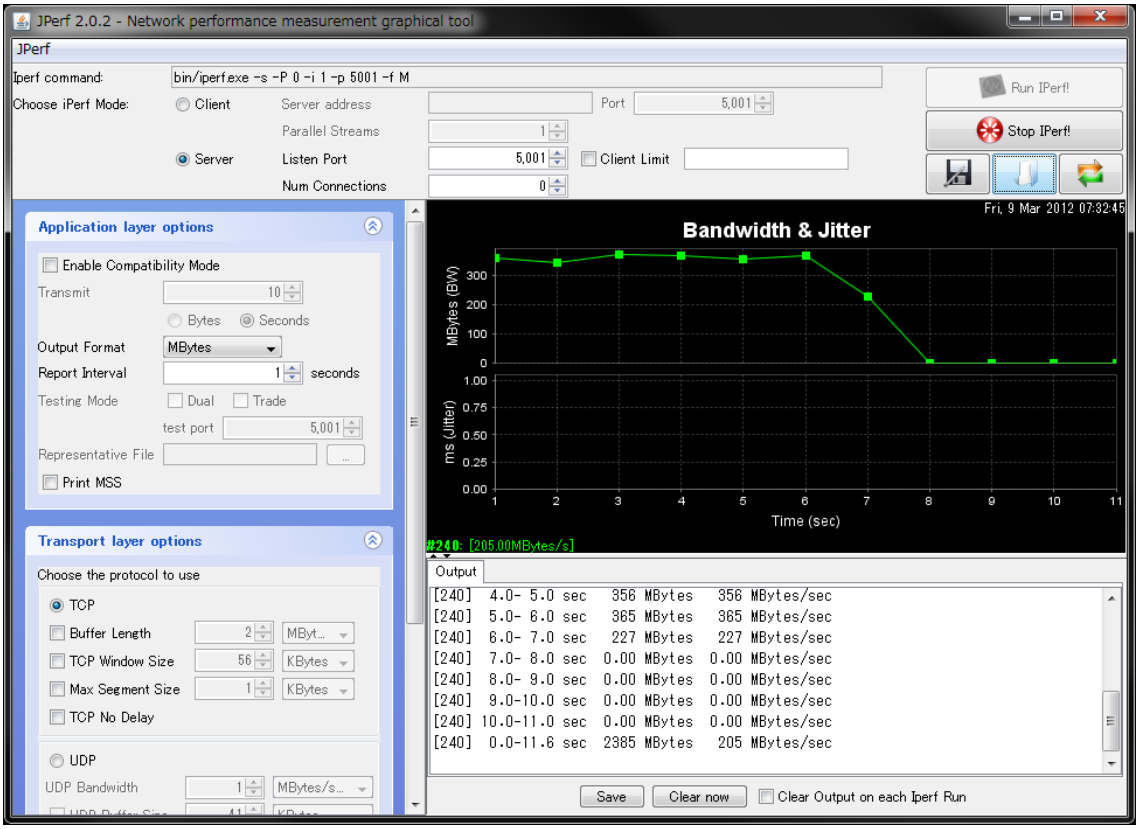


図 5.7　Jperf 実行画面

以下に Jperf を用いた平常時・輻輳状態の模式図を記す。

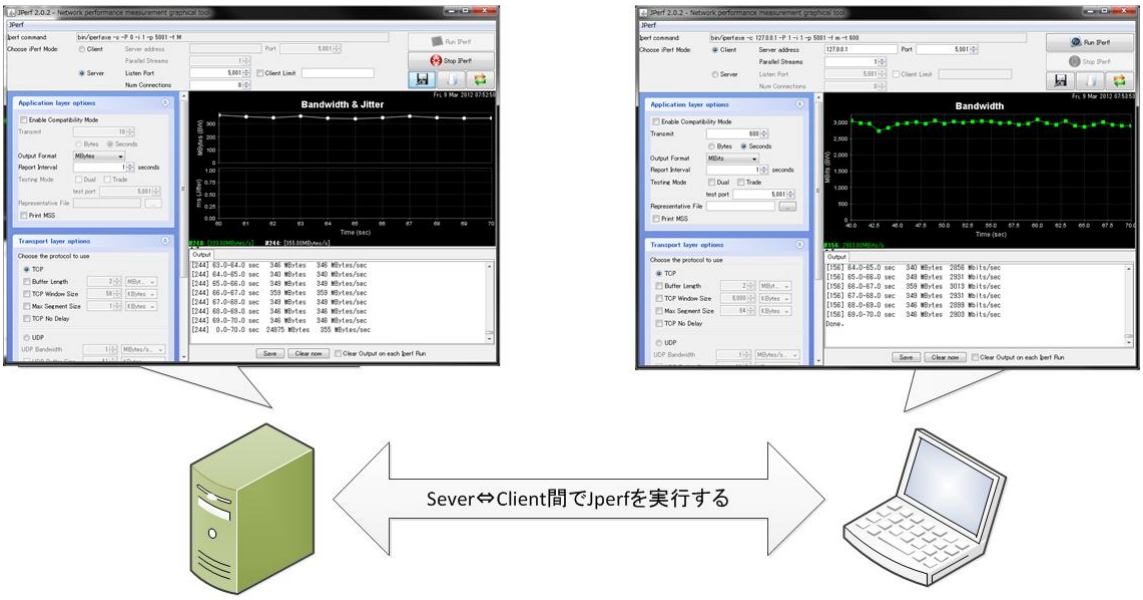


図 5.8　平常時の実験環境

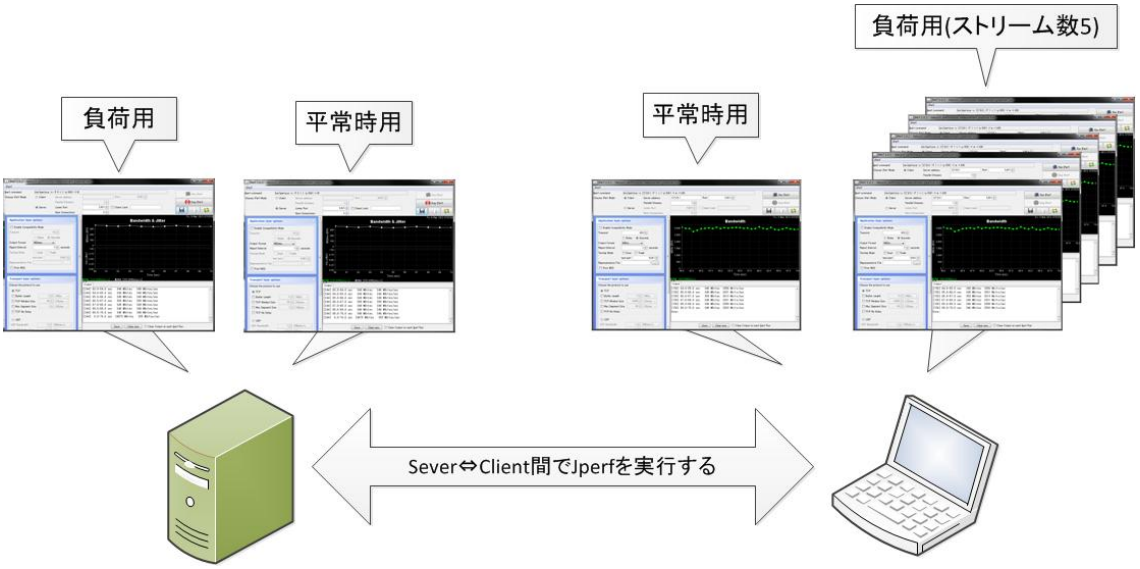


図 5.9　負荷状態の実験環境

平常時・輻輳発生時の Iperf の設定(コマンド・オプション)を表 5.5～5.8 に記す。

項目	値	コマンド オプション
モード	サーバーモード	-s
待ち受けポート	5001	-p 5001
プロトコル	TCP	標準設定
パケットサイズ(MSS)	1Kbyte	標準設定

表 5.5　平常時 サーバー設定

項目	値	コマンド オプション
モード	クライアントモード	-c
送信先ポート	5001	-p 5001
プロトコル	TCP	標準設定
ストリーム数	1	-P 1
転送時間	10 分	-t 600

表 5.6　平常時 クライアント設定

項目	値	コマンド オプション
モード	サーバーモード	-s
待ち受けポート	5002	-p 5002
プロトコル	TCP	標準設定
パケットサイズ(MSS)	1Kbyte	標準設定

表 5.7　負荷時 サーバー設定

項目	値	コマンドオプション
モード	クライアントモード	-c
送信先ポート	5002	-p 5002
プロトコル	TCP	標準設定
ストリーム数	**5**	-P 5
転送時間	1 分	-t 60

表 5.8　負荷時 クライアント設定

ストリーム数の解説は次のページで行う。

設定が必要である項目以外は初期設定(ウィンドウサイズ:5KByte バッファ:2KByte)のままである。IPv4 と IPv6 のネットワークに応じて適宜 IP アドレスは変更して実験を行っている。今回構築したネットワークはリンクスピードが 100Mbps であり、平常時の Iperf ではスループットが約 40～50Mbps ほど速度が出ている。

負荷時のクライアントのストリーム数とは並列に通信させる個数である。ストリーム数 5 ということは同時に同じ内容の通信を 5 ポート分行うという意味である。ストリーム数だけクライアントのスレッドを生成し、クライアントサーバの通信をシミュレーションできる。1 つの WEB サーバーに対して複数のクライアントのブラウザーが接続している状態がイメージしやすい。1 つの Iperf でスループットが 40～50Mbps 程度速度出ているため 100Mbps のリンク速度に対してあきらかな帯域不足である。しかしこれは確実に輻輳を発生させるために故意にストリーム数を増やしているのであるため問題は無い。負荷は連続して 1 分間行い、負荷をかけるタイミングは 4 分経過時と 8 分経過時である。

以下に定期的に負荷をかけた場合の、負荷の強度予想をグラフ化した。

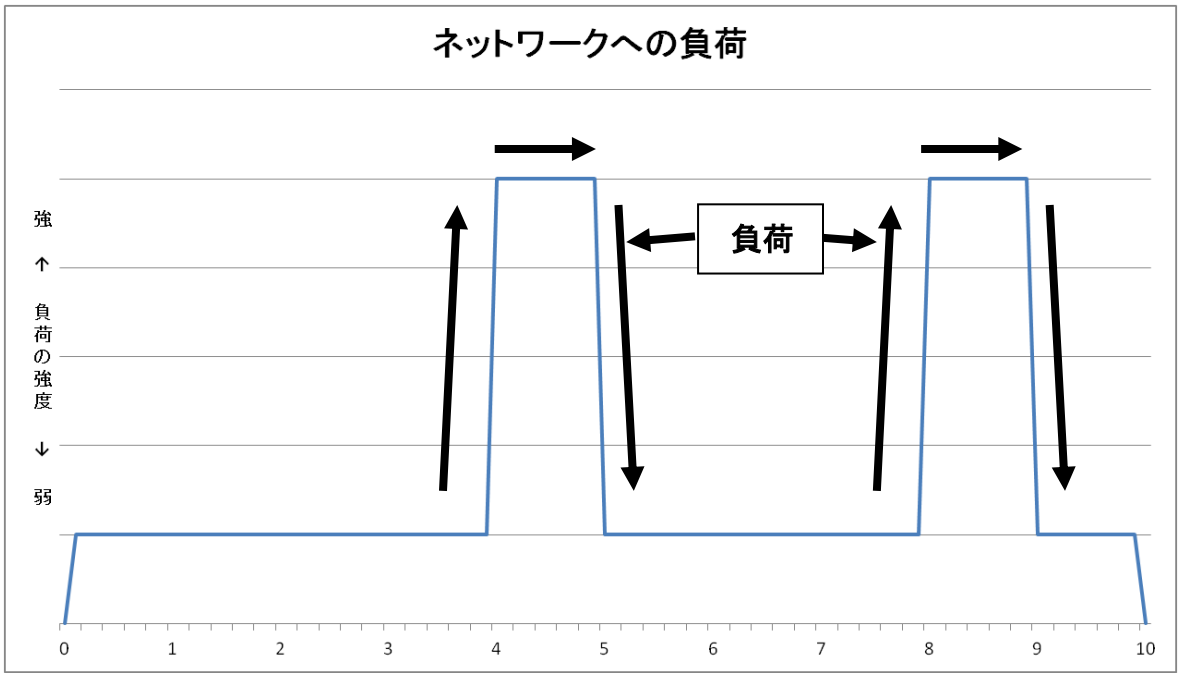


図 5.10 輻輳時の負荷の強度

このグラフは縦軸がネットワークへの負荷の強度、横軸が時間(分)である。山が 2 つできているが、これは、山ができているタイミングで Iperf を用いてネットワークへ負荷をかけているからである。負荷をかける方法は前述のとおりである。このタイミングは確実に輻輳状態となっているため、本研究の目的である輻輳時の重複 ACK パケットが検出できるか計測し、検出されれば実験は成功と言える。

5.5　輻輳検出ソフトウェア概要

ここでは、重複 ACK フラグを検出するために作成したプログラムについて説明する。本プログラムは標準的なネットワークプログラムを作成するために使用される Socket を用いて作成した。実験環境は OS として Windows を使用しているため、Socket の Windows 版である Winsock2 を用いている。今回作成した輻輳検出ソフトウェアの機能は①パケットキャプチャ②プロトコル解析機能③重複 ACK 検出機能の 3 つの機能である。(参考文献[1][2])

パケットキャプチャ機能は、ネットワーク中に流れてくるパケットをコンピュータに取り込む機能のことである。この機能によって本来は OS レベルで処理されてしまうパケットを見ることができるようになる。

次にプロトコル解析機能であるが、一般にパケットを取り込んだパケットが何のプロトコルを使用しているか判別する機能である。この機能によって必要なプロトコルのパケットをキャプチャし、実験に必要でないプロトコルを使ったパケットをフィルタリングすることが可能である。

最後に重複 ACK 判別機能については前述(3.5)されているが、ここで少し触れておくことにする。一般に重複 ACK はパケットが失われた場合などに受信コンピュータで発行されるフラグのことである。このフラグを数え上げることはそのままパケット損失率につながる。それは輻輳とも関係のあることなので、この機能が本プログラムで一番重要視される。この機能を使用し、通常の ACK とこの重複 ACK の割合をとることで輻輳の検知に役立てている。

パケットキャプチャプログラムのフラグ検出からその解析までの手順は以下のとおりである。

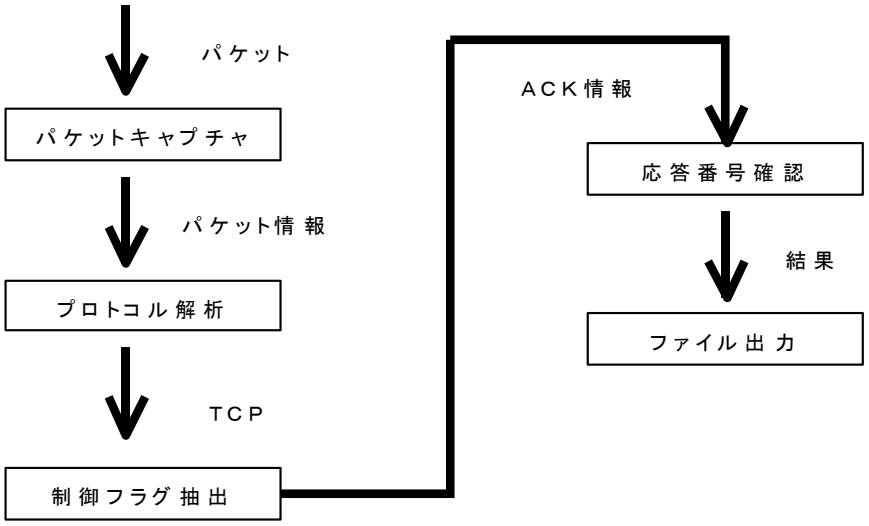


図 5.11　検出の流れ

①パケットキャプチャ
最初に行うことは流れているパケットを検出するパケットキャプチャである。今回作成したモニタでは IP パケットに反応するようにしている。
②プロトコル解析
パケットをキャプチャリングすると次にそのパケットのプロトコル解析を行う。この機能を使ってプロトコルを判別し、プロトコルのフィルタリングを行う。今回では必要としているプロトコルはTCPであり、それ以外のプロトコルを使ったパケットはフィルタリングされることになる。
③制御フラグ抽出
プロトコルによって分けられたパケットは、その中に含まれる 6 つの制御フラグの検出を行うことになる。
④応答番号確認
制御フラグに分けられた後は、パケットにＡＣＫが付加されていたときにその応答番号の確認を行う。ここで、以前送信側から送られてきた番号と、今回送信側から送られてきた番号が同じであったとき、そのＡＣＫは重複ＡＣＫとみなされカウントされる。
⑤結果出力
モニタが設定した時間(1 分)によって、フラグの状態とACKの割合を出力する。ここで出力されたものを結果として、比較・考察をおこなう。

パケットキャプチャプログラムを「Pcap」と名付け開発した。開発環境は Microsoft 社製 Visual C++ 2010 Express を使用し、ライブラリには Winsock2 ライブラリを、コンパイラには付属の VC++2010 コンパイラを使用している。

このプログラムをパケットキャプチャマシンで実行させ、サーバーのパケットをすべて計測していった。

第 6 章　　実験結果と考察

6.1　　実験結果

IPv4 ネットワーク、IPv6 ネットワーク、トランスレータネットワークの 3 種類のネットワークの平常時・負荷時の計 6 種類のネットワーク中の、重複 ACK フラグの状態を確認する実験を行った。

6.1.1　　IPv4 ネットワークの平常時・負荷時の結果

最初に IPv4 ネットワークの平常時でのキャプチャリングの実験を行った。平常時に使用する Iperf は輻輳が発生しない設定を前もって確認済みである。この状態でキャプチャ専用マシンによって全パケット数、ACK フラグ数、重複 ACK フラグ数を計測した。

時間(分)	パケット数	ACK パケット	重複 ACK
0～1	311989	273020	0
1～2	315604	276189	0
2～3	314468	275179	0
3～4	318509	278727	0
4～5	313011	273098	0
5～6	314111	274857	0
6～7	319208	279325	0
7～8	311673	272739	0
8～9	312654	273594	0
9～10	314995	275629	0
合計	3146222	2752357	0

表 6.1　IPv4 ネットワークの平常時のパケット

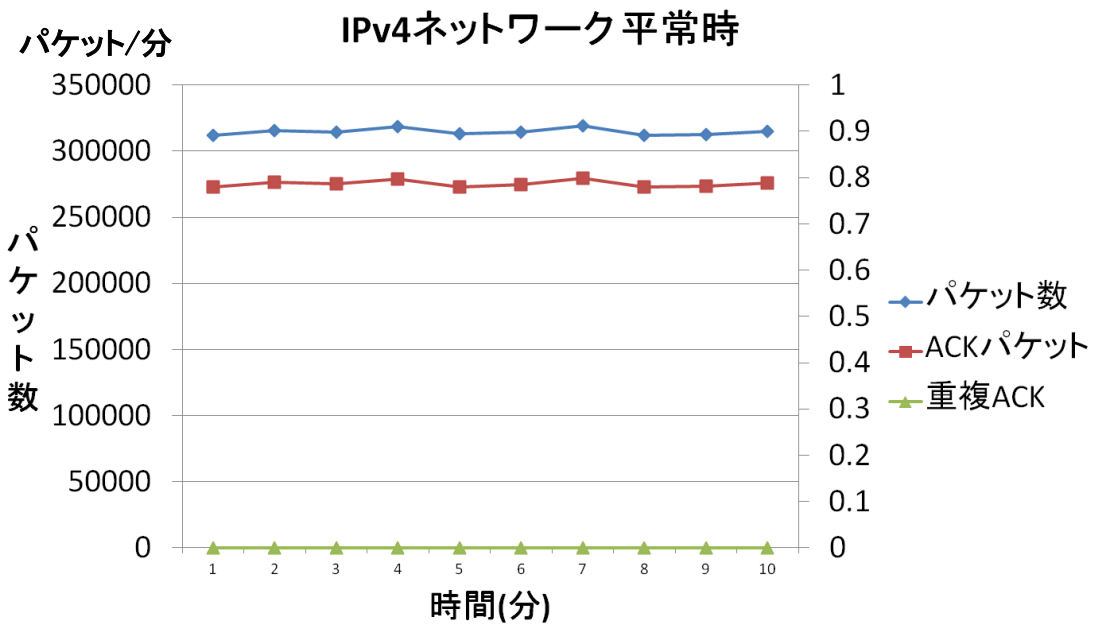


図 6.1　IPv4 ネットワークの平常時のパケット推移

このグラフは、縦軸がパケット数、横軸が時間(分)であり、左の縦軸の数値がパケット数と ACK パケットの個数、右側の縦軸の数値が重複 ACK パケットの個数となっている。以後のグラフはすべてこのスタイルで統一している。

結果は、重複 ACK フラグは確認されず比較的穏やかなグラフとなった。負荷はかけず輻輳は発生していないため重複 ACK フラグが検出されていない。一分当たりのパケット数はどの時間でもほぼ変化はみられない。

次に IPv4 ネットワークで負荷をかけた際のパケットの変化を計測した。特定の時間帯に負荷をかけることによって輻輳状態を作り出し、重複 ACK の変化を測る。

時間(分)	パケット数	ACK パケット	重複 ACK
0～1	312323	273331	0
1～2	312643	273627	0
2～3	315613	276198	0
3～4	318509	278727	0
4～5	530657	469216	3070
5～6	314052	274833	0
6～7	318443	278668	0
7～8	314513	275234	0
8～9	558000	493340	3262
9～10	315530	276100	0
合計	3610283	3169274	6332

表 6.2　IPv4 ネットワークの負荷時のパケット

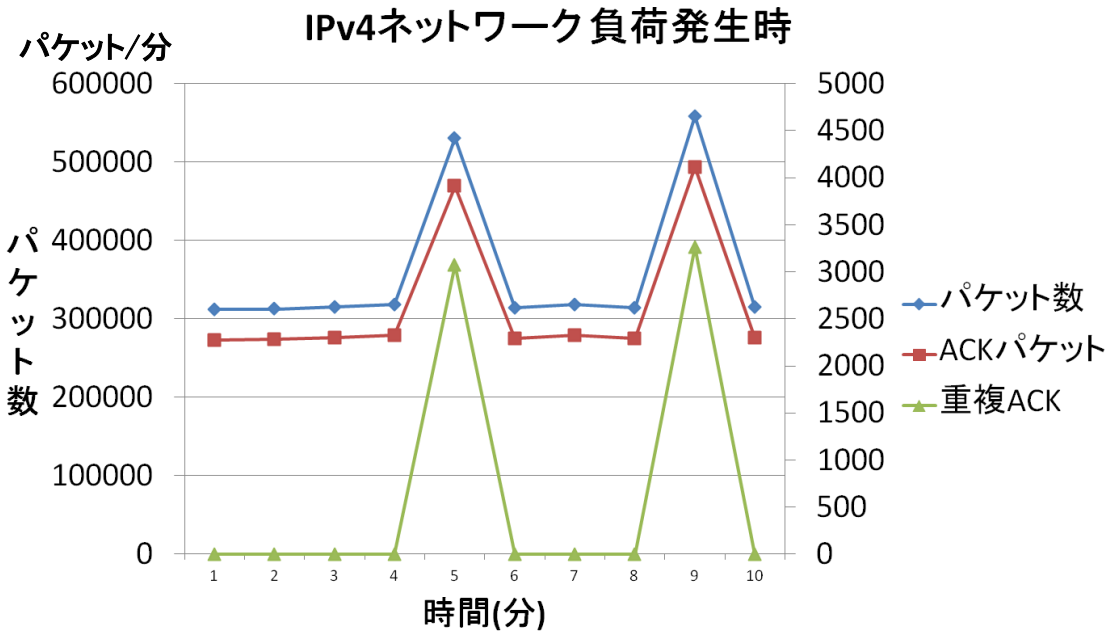


図 6.2　IPv4 ネットワークの負荷時のパケット推移

グラフにそれぞれのパケットの山ができており、平常時とは違ったグラフとなっている。特に重複 ACK が検出できている点が大きな違いだ。平常時の山ができている時間帯に負荷をかけ輻輳状態を発生させているため、影響を受けているとみられる。負荷時はネットワークで通信されるパケット量が増えるためパケット数や ACK パケットの数値が増加している。そして負荷をかけた際に重要な重複 ACK の増加も確認できる。

つまり、輻輳の影響を受け重複 ACK が検出されているのである。なおこの結果は先行研究で樋上が計測した結果とほぼ同じ内容である。

6.1.2 IPv6 ネットワークの平常時・負荷時の結果

今度は本研究最大の目的である IPv6 ネットワークでの実験を行った。こちらも先述の IPv4 と同じように平常時・負荷時のパケット推移を計測している。

ネットワーク構成は 5.2.2 で述べたものとなっている。まず、このネットワークでの平常時の TCP パケットをパケットキャプチャマシンで計測した。基本的にネットワーク構成は 6.1.1 で使用した IPv4 から、IPv6 に変更したのみである。それに伴って Iperf のアドレスが変更している。

時間(分)	パケット数	ACK パケット	重複 ACK
0～1	315124	275841	0
1～2	313250	274203	0
2～3	312240	273330	0
3～4	313039	274022	0
4～5	315811	276410	0
5～6	313081	274003	0
6～7	312990	273970	0
7～8	313623	274532	0
8～9	310593	271874	0
9～10	306645	268412	0
合計	3126396	2736597	0

表 6.3 IPv6 ネットワークの平常時のパケット

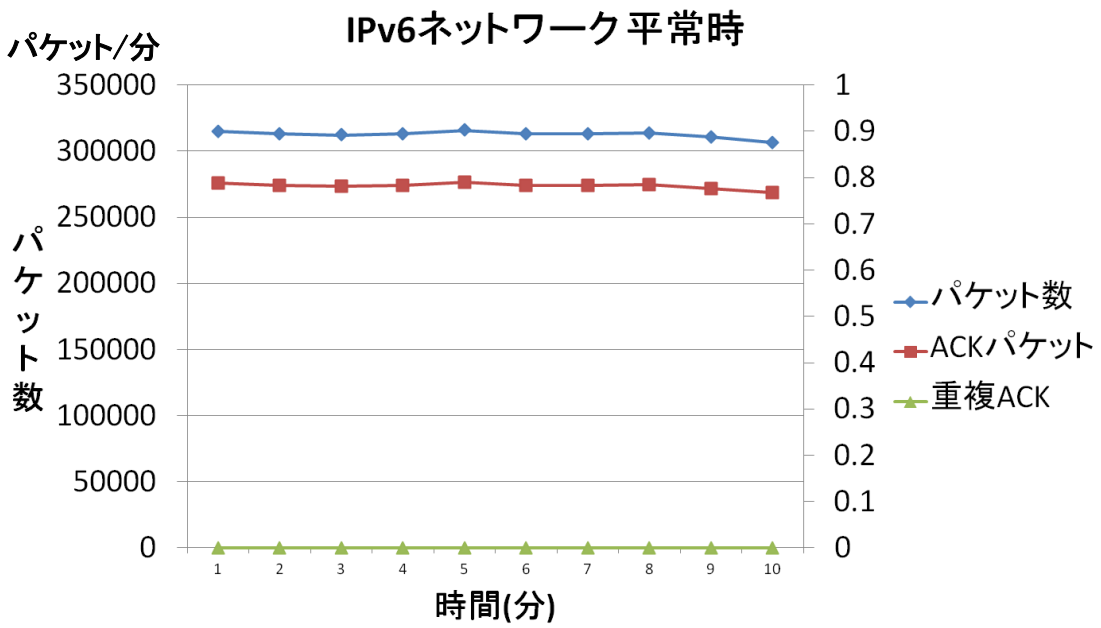


図 6.3　IPv6 ネットワークの平常時のパケット推移

IPv6 での平常時の各パケット推移は図 6.3 のような結果となった。この結果は IPv4 の結果とほぼ同じグラフの形をしている。負荷をかけていないため重複 ACK は確認できない。

次にネットワークに負荷をかけ、重複 ACK が確認できるか計測する。

時間(分)	パケット数	ACK パケット	重複 ACK
0～1	314175	274950	0
1～2	315120	275741	0
2～3	318147	278392	0
3～4	315142	275768	0
4～5	545389	481630	1750
5～6	314723	275400	0
6～7	316715	277132	0
7～8	314924	275564	0
8～9	614477	543309	2854
9～10	314532	275230	0
合計	3683344	3233116	4604

表 6.4　IPv6 ネットワークの負荷時のパケット推移

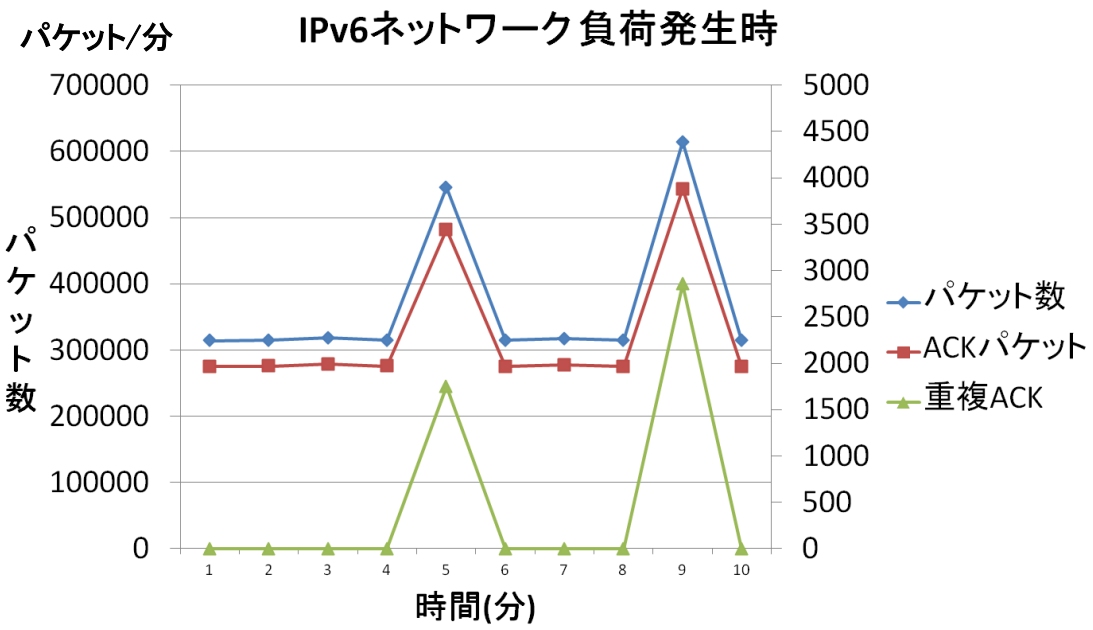


図 6.4　IPv6 ネットワークの負荷時のパケット推移

IPv6 でも負荷時にはグラフに山ができ、重複 ACK が確認された。平常時には重複 ACK は確認できない。この結果は IPv4 の負荷時のグラフの傾向とほぼ同じ結果となっている。少し重複 ACK パケットの量が IPv4 より少ないのは、IPv6 のルーティング時の負荷が下がっているからかもしれない。

つまり、先行研究で示された重複 ACK を利用した輻輳検出は IPv6 においても利用可能であるといえる。

6.1.3　IPv6/IPv4 ネットワークの平常時・負荷時の結果

6.1.2 にて IPv6 ネットワークでの重複 ACK を利用した輻輳検出が可能であることが分かった。最後に IPv6 と IPv4 をトランスレートさせたネットワークでの動作を検証する。ここでも以前のネットワークと同じく、平常時のものから測定を開始した。

時間(分)	パケット数	ACK パケット	重複 ACK
0～1	306659	267828	0
1～2	310829	271741	0
2～3	308218	269925	0
3～4	308806	270188	0
4～5	307867	269104	0
5～6	306620	268370	0
6～7	311350	271994	0
7～8	308733	269590	0
8～9	311405	273003	0
9～10	310554	271233	0
合計	3091041	2702976	0

表 6.5　IPv6/IPv4 トランスレータネットワークの平常時のパケット

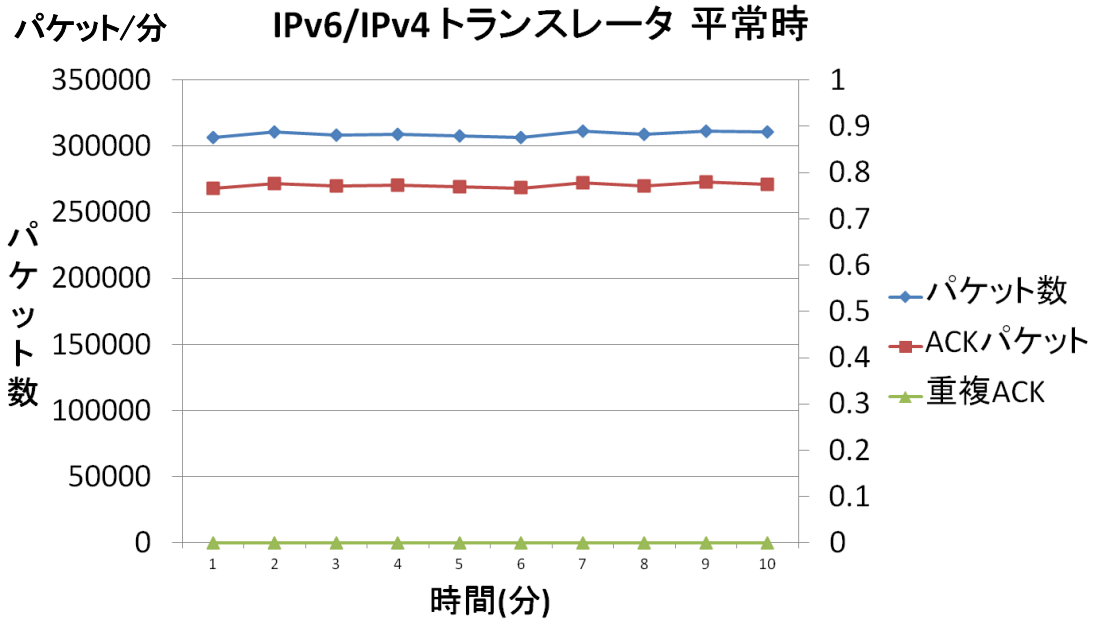


図 6.5　IPv6/IPv4 トランスレータネットワークの平常時のパケット推移

IPv4、IPv6 ネットワークと同様に、大きな変化は見られなかった。負荷もかかっていないため、重複 ACK は検出されていない。トランスレートの関係で全体的に他のネットワークと比較してパケット転送量が落ちているようだ。

最後に、トランスレータネットワークでの負荷時の測定を行った。

時間(分)	パケット数	ACK パケット	重複 ACK
0～1	311711	272564	0
1～2	308970	269218	0
2～3	310314	270758	0
3～4	309154	269767	0
4～5	516798	454627	2987
5～6	310793	271291	0
6～7	311385	271744	0
7～8	309659	269982	0
8～9	547042	484181	3020
9～10	309499	270028	0
合計	3545325	3104160	6007

表 6.6　IPv6/IPv4 トランスレータネットワークの負荷時のパケット

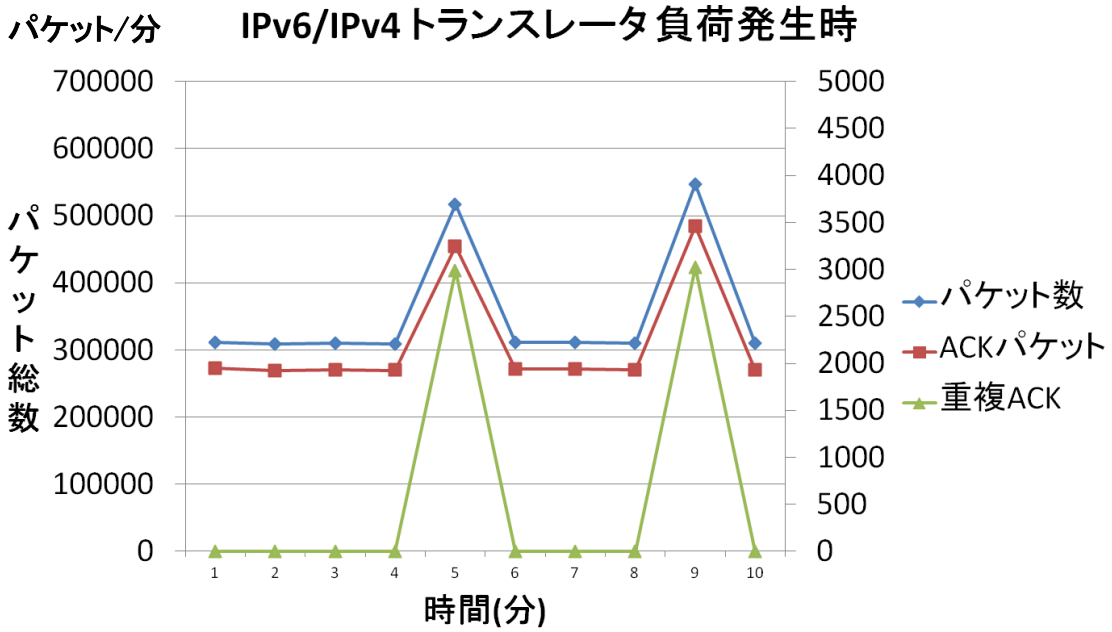


図 6.6　IPv6/IPv4 トランスレータネットワークの負荷時のパケットの推移

トランスレータネットワークでも輻輳時に、重複 ACK パケットが検出された。

6.2 実験結果まとめ

それぞれ IPv6 など 6 種類のネットワークでの実験結果を以下の表にまとめる。

ネットワーク	平常時	負荷時
IPv4 アドレスネットワーク	検出されない	**検出された**
IPv6 アドレスネットワーク	検出されない	**検出された**
トランスレータネットワーク	検出されない	**検出された**

表 6.7　重複 ACK パケット検出まとめ

実験結果をまとめると、平常時では重複 ACK パケットが検出されず、負荷時にのみ重複 ACK パケットが検出された。

6.3 考察

本研究の目的である IPv6 での重複 ACK を利用した輻輳検出は可能であることが分かった。先行研究でも示されていた IPv4 だけではなく IPv6 でもよいのである。またトランスレートされたネットワークにおいても、同様に重複 ACK を用いた輻輳検出が可能といえる。

4.4 で述べた予想のとおり、インターネット層のプロトコルの変更は、上位のトランスポート層にはほぼ影響がないとみられる。TCP プロトコルを使用していれば重複 ACK は輻輳時に検出されると思われる。

今後 IPv4 から IPv6 にシフトしていくと思われるが、そういった環境でも変わらずにこの輻輳検出法が利用可能であることが確認できた。

6.4 まとめと展望

イントラネット内での重複 ACK フラグを利用した輻輳検出が可能であることが分かったため、データセンター内などでの利用が考えられる。イントラネット環境だけではなく、インターネット上でどのように重複 ACK パケットが振る舞うかが次の研究の対象になるだろう。

本研究では試験的に構築したネットワーク内の実験であったため、重複 ACK を利用した輻輳検出を利用するのであれば、今後インターネットや大規模なネットワークでのテストが必要になると予想される。

また、この方法を利用した手軽に輻輳状態を検出できるソフトウェアの開発も可能だろう。近年スマートフォンが普及し携帯端末でも大容量の通信をすることが多くなったのでキャリアの交換機が輻輳状態に陥らないよう、ロードバランサーなどが輻輳発生を予防するためパケットの破棄を行っているとよく聞く。特に多数の通信が発生する夜などの時間帯にはかなりのパケットが破棄されてしまうようだ。こういった場合これまではネットワークのスループットを測って確認するしかなかったが本研究の方法を利用すれば破棄されたパケットに対する再送要求パケットをカウントするだけでどの程度輻輳が起きているか簡単にわかるだろう。スマートフォン向け輻輳検出ソフトウェアもよいかもしれない。

6.5 謝辞

本研究を進めるにあたり、最後まで熱心なご指導、ご助言をして頂きました田中章司郎教授には心より御礼申し上げます。また、本研究のきっかけとなりましたすばらしい研究とアイデアを提供していただいたことは感謝の言葉もありません。

同研究室の皆様にも、研究内容のみに留まらない数々の御協力と御助言を頂きました。ここに厚く御礼申し上げます。

なお、本論文，本研究で作成したプログラム及びデータ，ならびに関連する発表資料等の知的財産権を、本研究の指導教官である田中章司郎教授に譲渡致します。

参考文献

[1]
TCP 制御フラグを使った輻輳検知に関する研究　樋上智彦　2004

[2]
TCP 制御フラグによる輻輳検知方法のプロトコル別検証　関本啓介　2005

[3]
速習 TCP/IP ゼロからのネットワーク　三浦一志　2003

[4]
日経 BP 社 Cisco CCNA ICND1 テキスト　Gene 松田千賀/著　2007

付録

ここでは、構築したネットワークのルーターのコンフィグを載せている。

1　　IPv4 ネットワーク

```
1812J-IPv4#show running-config
Building configuration...

Current configuration : 1273 bytes
!
! Last configuration change at 17:14:15 UTC Thu Mar 15 2012
!
version 15.1
service timestamps debug datetime msec
service timestamps log datetime msec
no service password-encryption
!
hostname 1812J-IPv4
!
boot-start-marker
boot-end-marker
!
!
!
no aaa new-model
!
crypto pki token default removal timeout 0
!
!
dot11 syslog
no ip source-route
!
!
!
!
```

```
!
ip cef
no ipv6 cef
!
multilink bundle-name authenticated
!
!
!
license udi pid CISCO1812-J/K9 sn FHK123126FC
!
!
!
!
!
!
!
!
!
interface BRI0
 no ip address
 encapsulation hdlc
 shutdown
!
interface FastEthernet0
 ip address 192.168.20.1 255.255.255.0
 duplex auto
 speed auto
!
interface FastEthernet1
 no ip address
 shutdown
 duplex auto
 speed auto
!
interface FastEthernet2
!
```

```
interface FastEthernet3
!
interface FastEthernet4
!
interface FastEthernet5
!
interface FastEthernet6
!
interface FastEthernet7
!
interface FastEthernet8
!
interface FastEthernet9
!
interface Vlan1
  ip address 192.168.10.1 255.255.255.0
!
ip forward-protocol nd
no ip http server
no ip http secure-server
!
!
!
logging esm config
!
!
!
!
!
!
control-plane
!
!
!
line con 0
  exec-timeout 0 0
```

```
line aux 0
line vty 0 4
 login
 transport input all
!
!
monitor session 1 source interface Fa2
monitor session 1 destination interface Fa9
end

1812J-IPv4#
```

2 IPv6 ネットワーク

```
1812J-IPv6#show running-config
Building configuration...

Current configuration : 1312 bytes
!
! Last configuration change at 17:24:39 UTC Thu Mar 15 2012
!
version 15.1
service timestamps debug datetime msec
service timestamps log datetime msec
no service password-encryption
!
hostname 1812J-IPv6
!
boot-start-marker
boot-end-marker
!
!
!
no aaa new-model
!
```

```
crypto pki token default removal timeout 0
!
!
dot11 syslog
no ip source-route
!
!
!
!
!
ip cef
ipv6 unicast-routing
ipv6 cef
!
multilink bundle-name authenticated
!
!
!
license udi pid CISCO1812-J/K9 sn FHK123126FC
!
!
!
!
!
!
!
!
!
!
interface BRI0
 no ip address
 encapsulation hdlc
 shutdown
!
interface FastEthernet0
 no ip address
 duplex auto
```

```
 speed auto
 ipv6 address 2001:D:E:F::1/64
!
interface FastEthernet1
 no ip address
 shutdown
 duplex auto
 speed auto
!
interface FastEthernet2
!
interface FastEthernet3
!
interface FastEthernet4
!
interface FastEthernet5
!
interface FastEthernet6
!
interface FastEthernet7
!
interface FastEthernet8
!
interface FastEthernet9
!
interface Vlan1
 no ip address
 ipv6 address 2001:A:B:C::1/64
!
ip forward-protocol nd
no ip http server
no ip http secure-server
!
!
!
logging esm config
```

```
!
!
!
!
!
!
control-plane
!
!
!
line con 0
 exec-timeout 0 0
line aux 0
line vty 0 4
 login
 transport input all
!
!
monitor session 1 source interface Fa2
monitor session 1 destination interface Fa9
end

1812J-IPv6#
```

3 IPv6/IPv4 トランスレータネットワーク

```
1812J-NAT-PT#show running-config
Building configuration...

Current configuration : 1462 bytes
!
! Last configuration change at 17:49:11 UTC Thu Mar 15 2012
!
version 15.1
service timestamps debug datetime msec
```

```
service timestamps log datetime msec
no service password-encryption
!
hostname 1812J-NAT-PT
!
boot-start-marker
boot-end-marker
!
!
!
no aaa new-model
!
crypto pki token default removal timeout 0
!
!
dot11 syslog
no ip source-route
!
!
!
!
!
ip cef
ipv6 unicast-routing
ipv6 cef
!
multilink bundle-name authenticated
!
!
!
license udi pid CISCO1812-J/K9 sn FHK123126FC
!
!
!
!
!
```

```
!
!
!
!
interface BRI0
 no ip address
 encapsulation hdlc
 shutdown
!
interface FastEthernet0
 ip address 192.168.20.1 255.255.255.0
 duplex auto
 speed auto
 ipv6 enable
 ipv6 nat
!
interface FastEthernet1
 no ip address
 shutdown
 duplex auto
 speed auto
!
interface FastEthernet2
!
interface FastEthernet3
!
interface FastEthernet4
!
interface FastEthernet5
!
interface FastEthernet6
!
interface FastEthernet7
!
interface FastEthernet8
!
```

```
interface FastEthernet9
!
interface Vlan1
 no ip address
 ipv6 address 2001:A:B:C::1/64
 ipv6 nat
!
ip forward-protocol nd
no ip http server
no ip http secure-server
!
!
!
logging esm config
ipv6 nat v4v6 source 192.168.20.64 2001::C0A8:1440
ipv6 nat v6v4 source 2001:A:B:C::101 192.168.20.101
ipv6 nat prefix 2001::/96
!
!
!
!
!
!
control-plane
!
!
!
line con 0
 exec-timeout 0 0
line aux 0
line vty 0 4
 login
 transport input all
!
!
monitor session 1 source interface Fa2
```

```
monitor session 1 destination interface Fa9
end

1812J-NAT-PT#
```