Meteor: 大規模ネットワーク実験環境における無線ネットワークエミュレータの設計と実装*

明石　邦夫[†a)]　井上　朋哉[††]　ラズバン ベウラン[†††]　篠田　陽一[††††]

Meteor: Design and Implementation of a Wireless Network Emulator for Large Scale Experimental Networks*

Kunio AKASHI[†a)], Tomoya INOUE[††], Razvan BEURAN[†††], and Yoichi SHINODA[††††]

あらまし　ネットワークテストベッドは，多数の計算機とネットワーク機器を接続し，大規模な検証・実験が可能な検証環境であるが，無線環境を想定した検証を行うためには計算機間に無線空間を擬似的に作り出す必要がある．我々はこの目的のために有線ネットワーク上に無線空間を擬似的に作り出すネットワークエミュレータ QOMET を提案した．しかしながら，QOMET を含む既存のネットワークエミュレータはネットワーク層での動作を前提としており，IEEE 802.11s や B.A.T.M.A.N のようなデータリンク層で動作するアプリケーションソフトウェアに対応できない問題がある．本論文では，データリンク層で動作するアプリケーションソフトウェアにも対応可能なネットワークエミュレータである Meteor の設計・実装を説明する．Meteor は，有線ネットワーク上の計算機間で無線伝送に伴う遅延時間や帯域幅とそれらの変動を忠実に再現する．そして，プロトコルに依存しない無線空間を作り出すことにより，これまで困難であったデータリンク層で動作するアプリケーションソフトウェアの検証を可能とした．

キーワード　ネットワークテストベッド，仮想化技術，ネットワークエミュレーション，無線ネットワーク

1. まえがき

ネットワークテストベッドは，新たに開発したアプリケーションソフトウェアの検証を行うために構築された実験環境として利用できる．このネットワークテストベッドには，PlanetLab [1] のようにインターネット上にオーバレイネットワークを構築するものと StarBED のように施設内に閉じた環境で閉鎖した実験ネットワークを構築するものがある．オーバレイネットワーク上で実験を行う場合，インターネット上に計算機を展開し検証が可能であるため，他のユーザのトラヒックや遠方の計算機との間で発生する遅延や帯域幅などの制限があるネットワーク環境での検証が可能である．これに対し，StarBED のような閉鎖した施設内で実験を行う場合は，他のユーザのトラヒックの影響を受けず，低遅延・広帯域な環境で検証が可能である．また，施設内に用意されている多数の計算機とネットワークスイッチからなるクラスタ環境で様々なアプリケーションの検証を繰り返し容易に行うことが可能である．多数の計算機を用いることにより大規模な実験環境が構築可能であるが，無線ネットワークのような遅延や帯域幅，パケットロスが時間推移で変動するような通信環境を構築することが困難であった．Wi-Fi，3G のような無線を利用したモバイル端末が増えてきている中，ネットワークテストベッド上で擬似的な無線空間を作り出す手法が求められている．

そこで，有線環境上に無線環境における挙動を擬似

† 北陸先端科学技術大学院大学情報科学研究科，能美市
JAIST Japan Advanced Institute of Science and Technology Infomation Sceience, Nomi-shi, 923–1211 Japan
†† 北陸先端科学技術大学院大学高信頼ネットワークイノベーションセンター，能美市
JAIST Japan Advanced Institute of Science and Technology Dependable Network Innovation Center, Nomi-shi, 923–1211 Japan
†††† 北陸先端科学技術大学院大学情報社会基盤研究センター，能美市
JAIST Japan Advanced Institute of Science and Technology Research Center for Advanced Computing Infrastructure, Nomi-shi, 923–1211 Japan
††† 情報通信研究機構北陸 StarBED 技術センター，能美市
NICT Hokuriku StarBED Technology Center, Nomi-shi, 923–1211 Japan
a) E-mail: k_akashi@jaist.ac.jp
* 本論文は，システム開発・ソフトウェア開発論文である．

的に作り出すネットワークエミュレータが数多く提案されている．ネットワークエミュレータを用いて擬似的な無線環境を作り出すことにより，無線環境での利用を想定しているアプリケーションソフトウェアの検証を繰り返し容易に行うことが可能となる．

我々は，ネットワークテストベッド上で擬似的な無線空間を構築するネットワークエミュレータとしてQOMET [2] を提案した．QOMET は，ネットワーク層で無線環境の振る舞いを作り出し，アプリケーションソフトウェアの検証を可能とするネットワークエミュレータである．これまで無線ネットワークでの利用を想定しているアプリケーションソフトウェアの検証は，QOMET により作られた擬似的な無線空間で十分対応できていた．しかし，IEEE 802.11s [3] や B.A.T.M.A.N [4] のようなデータリンク層で利用される無線技術が提案されたことにより対応が難しい場面が出てきている．

そこで，本論文では IEEE 802.11s や B.A.T.M.A.N のようなイーサネットフレームを扱うプロトコルにも対応可能とし，AODV のようなネットワーク層で動作するルーティングプロトコルで用いたマルチホップ通信時にもパケット転送ごとにエミュレーションが適用可能なネットワークエミュレーション手法の提案を行う．提案手法は，既存研究の手法であるネットワーク層ではなく，データリンク層で擬似的な無線空間を作り出すことにより，これまで検証が困難なものであった無線技術の検証を可能とする．そして，提案手法を実現するためアプリケーション及びルーティングソフトウェアが扱うプロトコルに依存しないネットワークエミュレータである Meteor [5] の設計・実装を行う．本論文では，以降アプリケーション及びルーティングソフトウェアのことをアプリケーションソフトウェアと呼ぶ．提案手法により，データリンク層の上で動作するアプリケーションソフトウェアの検証が可能な擬似無線環境を作り出すことが可能となる．

2. ネットワークエミュレータ

2.1 ネットワークテストベッドで利用されるネットワークエミュレータ

ネットワークテストベッドで提供している環境は低遅延・広帯域な有線ネットワークであり，アプリケーションソフトウェアの負荷検証や規模耐性の検証に利用できる．また，安定した環境で検証できるため，同じ検証を同じ条件で何度も繰り返し行えるといった特徴がある．しかし，無線ネットワークでは，遅延時間や帯域幅，パケットロス率が大きく変動する．ネットワークテストベッドで無線ネットワークを想定した検証を行うには，無線空間で観測される遅延時間や帯域幅，パケットロスなどを有線環境上に作り出す必要がある．

無線ネットワークを対象としたテストベッドとして ORBIT [6] がある．ORBIT は，屋内に設置した 802.11 radio node を制御することで無線グリッドを構築する．ORBIT では実際に無線伝送を行うため，現実の無線通信による検証が可能である．一方で，大規模な検証を行うためには広大な施設が必要となる．また，ORBIT の 802.11 radio node は移動端末として利用することはできないため，モビリティの検証に利用することは難しい．

そこで，ネットワーク上を流れるパケットを一時的に滞留させたり，破棄するようなトラヒック制御を行うことにより無線空間における通信品質を作り出すネットワークエミュレータが数多く設計・実装されている．ネットワークエミュレータを用いることでアプリケーションソフトウェアを動作させている計算機が無線空間に存在しているかのように見せることができる．また，有線環境上で擬似的に無線空間を作り出すため，同じ検証を繰り返し行えるというネットワークテストベッドの一つの特徴が活用できる．

2.2 擬似的な無線空間

有線ネットワークで構築されたネットワークテストベッドで無線空間の検証を行うには，無線ネットワークの特徴を再現する必要がある．有線ネットワーク上で実際に無線空間を作り出すことは不可能であるため，既存研究では物理層より上位層で無線ネットワークの特徴を再現することにより擬似的に無線空間を再現する手法がとられている．有線ネットワーク上に無線環境を擬似的に作り出すには，汎用計算機上でネットワークエミュレータ若しくは，専用ハードウェアを用いる方法が存在する．専用ハードウェアを用いたネットワークエミュレータには，SHUNRA VE-STN [7] や Anue Systems Anue 3500 [8] などがある．専用ハードウェアは，遅延挿入や帯域制限の精度が高く，ワイヤースピードでの動作が保証されているが，機器の価格が高くスケールアウトも難しい．ソフトウェアによるエミュレーションは，ハードウェアと比較して精度が劣るが，汎用計算機で動作するため導入コストが低くスケールアウトが容易である．擬似的な無線空間を

作り出すネットワークエミュレータに関する既存研究には QOMET や MobiNet [9] などがある．

2.3 QOMET

QOMET は我々が提案した，Wi-Fi や Zigbee などで構成される無線ネットワークを再現するネットワークエミュレータである．QOMET は，以下に述べるようにユーザが定義した空間をシミュレーションする deltaQ とシミュレーション結果から有線環境上に擬似的な無線環境を作り出す wireconf から構成される．

2.3.1 deltaQ

deltaQ は，フィールドとなる環境や計算機の位置関係などを記述したシナリオをもとに，計算機間の通信品質を算出する．シナリオには，計算機の位置関係や移動情報，障害物情報の他に IEEE 802.11b のような通信メディア，環境に応じた減衰パラメータ α，分散パラメータである σ が XML で記述されている．deltaQ は，これらの情報から log-distance path loss model を用いて計算機の間の信号減衰を算出する．算出した信号減衰値から通信品質を示す Frame Error Rate (FER) を算出する．

deltaQ の出力には，計算機間の遅延時間や帯域幅，パケットロス率などが時間ごとに記述される．QOMET では，シミュレーションを行う deltaQ とエミュレーションを行う wireconf に分離しているため，新たな無線メディアが提案されたとしても，deltaQ の改変を行うのみでエミュレーションが行える設計としている．

2.3.2 wireconf

wireconf は deltaQ の出力から無線環境を有線環境上に構築するエミュレータである．wireconf は deltaQ に記述されている送信元 ID と送信先 ID のペアとそのペア間の遅延時間や帯域幅などを deltaQ の出力から読み込み，これに基づいてノードが送受信するパケットをフィルタリングし遅延挿入，帯域制限などのトラヒック制御を行う．

wireconf は，図 1 で示すようにネットワークスタック上のネットワーク層で動作する．ネットワーク上を流れるパケットからは deltaQ で管理している ID が判断できないため，ネットワーク層の識別子である計算機の IP アドレスと deltaQ のノード ID を対応付けるテーブルをもつ．しかし，各計算機が送信するパケットには必ずしも一意な IP アドレスがついているわけではない．計算機間の通信がユニキャストであれば，送信元アドレスと送信先アドレスは一意なものであるが，ブロードキャストの場合，送信先アドレスは

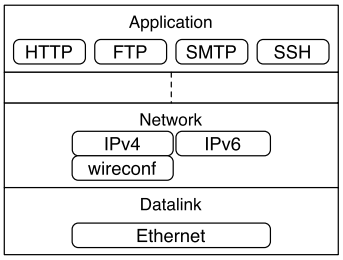

図 1 OSI 参照モデルにおける wireconf の動作場所
Fig. 1 Position of wireconf in the OSI reference model.

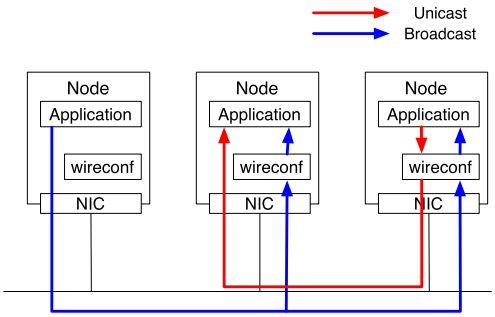

図 2 wireconf のユニキャストとブロードキャストの扱い
Fig. 2 Unicast and broadcast handling in wireconf.

ネットワーク全体で共通なものとなる．この問題に対して wireconf では，ユニキャストとブロードキャストを図 2 のように扱っている．ユニキャストに対しては，自身から送信するパケットに送信先アドレスでフィルタリングし，トラヒック制御を行う．ブロードキャストに対しては，自身宛のパケットを送信元アドレスでフィルタリングし，これにより，wireconf は有線環境上に無線環境を擬似的に作り出すが，計算機の識別に IP アドレスを用いているため，IP ヘッダをもたないイーサネットフレームに対してはトラヒック制御が行えない．例えば，IEEE 802.11s は，図 1 のデータリンク層で動作するプロトコルである．そのため，ネットワーク層で動作するネットワークエミュレータを通過せず，トラヒック制御が行われない．また，ネットワーク層で動作するルーティングプロトコルにおいても，同一ネットワーク内でマルチホップさせた場合はデータリンク層の識別子である MAC アドレスを変更しながら転送処理を行う．既存研究では，この問題に対しルーティングテーブルを参照しつつ近隣計算機への遅延や帯域幅の制限を変更する処理を行っている．しかし，この方法ではネットワーク層のプロトコルに

依存してしまうため，ネットワークエミュレータ実装後に新たに提案されたネットワーク層のプロトコルに対応するためには，ネットワークエミュレータに大幅な変更が必要となる．また，ネットワーク層のプロトコルを用いないデータリンク層のみで動作するプロトコルに対応できない問題がある．

2.4 ネットワークエミュレータの規模拡張性

既存研究で提案されているネットワークエミュレータは数十台から 200 台程度の環境での利用を想定している．QOMET は，実際に 100 台の計算機を用いてアドホックネットワークを構築している実績がある [10], [11]．MobiNet は，40 台での検証にとどまっている．

AODV では，RFC にてマルチホップ数の上限が 35 であると決められている．これを 100 台の計算機で行おうとすると計算機の配置を密な状態としなければならない．また，携帯電話の 3G 網には一つのセルに数百人のユーザが収容される．これらの環境を想定すると，ネットワークエミュレータは 1000 台以上の計算機を扱える必要がある．

3. データリンク層で動作するネットワークエミュレータの検討

ネットワークエミュレータは，有線環境上に擬似的な無線空間を構築する．これにより，状況の再現が難しい無線環境での実験が繰り返し行える．しかし，既存研究である QOMET や MobiNet は，ネットワーク層で動作し，IPv4 パケットに対してフィルタリングを適用するため，IPv4 ヘッダをもたないイーサネットフレームには対応できない．

QOMET は，IPv4 パケットに対して遅延挿入や帯域制限を行うことで，自ノードと他ノード間における無線空間を作り出す．そのため，IPv4 ヘッダをもたないイーサネットフレームのみの情報を用いる IEEE 802.11s や AODV の場合，フィルタリングできず有線ネットワーク上に擬似的な無線空間を作り出せない．

MobiNet は，MAC レイヤをエミュレーションしノード間の無線空間を作り出している．しかし，MobiNet も，フィルタリングをネットワーク層で行っており，イーサネットフレームに対してのフィルタリングができない．また，ネットワークエミュレータ自体が，アドホックルーティングプロトコルの機能をもっているため，アプリケーションソフトウェアを動作させる計算機がルーティングプロトコルを使用した際に正しい評価ができない．

既存研究におけるネットワーク層での遅延挿入や帯域制限では，データリンク層で動作するアプリケーションソフトウェアの検証環境に対して制限をかける結果となってしまっている．ネットワークエミュレータは，擬似的に無線環境を作り出すソフトウェアであるため，その上で利用するアプリケーションソフトウェアを制限することは避けるべきである．新たな技術の提案がされる中，ネットワークエミュレータもそれに対応していくことが必要となっている．

この問題は，新たなアプリケーションソフトウェアの検証に対応するためネットワーク層ではなく，データリンク層で擬似的な無線空間を作り出すことで解決可能であると考えられる．しかし，そのためにネットワークエミュレータの操作性を著しく変えてしまうことや，過去のデータが使えなくなってしまうことにも問題がある．ネットワーク実験において，過去に使用したシナリオを用いて再実験を行うことは珍しくない．その際，ネットワークエミュレータに新たな機能が追加されたためにシナリオの作り直しが発生してしまうことはネットワークテストベッドの特徴である再現性を損ねてしまう．データリンク層に対応したネットワークエミュレータを使用するとしても，過去のシナリオが利用可能であるべきである．

本論文で提案するのは，イーサネットフレームに対するフィルタリングを行い，アプリケーションソフトウェアを制限しない擬似的な無線空間のエミュレーション手法である．そして，提案手法を実現するためのネットワークエミュレータである Meteor の設計と実装を行った．本提案手法では遅延挿入や帯域制限をネットワーク層ではなくデータリンク層で行うことにより，IPv4 ヘッダを含まないイーサネットフレームへの対応が可能となる．また，本提案手法は既存研究である QOMET の deltaQ で生成されたシナリオで動作する設計となっているため，過去に QOMET で行った実験も本提案手法で再実験可能としている．これにより，既存のネットワークエミュレータを用いての検証が困難であったプロトコルやアプリケーションソフトウェアの検証を可能とする．

4. Meteor の設計と実装

ネットワークエミュレータの設計と実装に関する既存研究は多数存在するが，その多くがリンクエミュレータと呼ばれる機構を利用している．例えば QOMET

では，遅延挿入や帯域制限の実現に Dummynet [12] を用いている．リンクエミュレータには，ソフトウェアとハードウェアが存在する．本章では，既存研究で多く使用されている代表的なリンクエミュレータ及び実装手法について解説を行う．

4.1 要　　件

既存研究で提案されているネットワークエミュレータは，ネットワーク層で動作するため，計算機の識別子に IPv4 アドレスを使用している．ネットワーク層で定義されている IPv4 アドレスは，ネットワーク内でユニークな識別子であるため，パケットからは送信元と送信先の計算機しか判別できない．そのため，パケットをルーティングする実験では，次の転送先の情報をパケット以外から取得しエミュレーションの設定を変更する必要がある．既存研究で提案されているネットワークエミュレータは，計算機の経路表を参照・変更し，その情報を元に動作する手法を取っている．しかし，計算機の経路表からはネットワーク層の情報しか得られず，既存研究で提案されているネットワークエミュレータは IPv4 にしか対応していないものが多い．本来，次の転送先を決定した後に書き換えられる情報はデータリンク層の識別子である MAC アドレスであり，ネットワーク層の識別子である IP アドレスは変更されない．

ネットワーク層に依存しないネットワークエミュレータとするためには，各計算機間の遅延挿入や帯域制限を行うリンクエミュレータがネットワーク層よりも下位層で動作する必要がある．ネットワークテストベッドの有線ネットワークを変更することは物理構成を変更することとなるため，既存の構成を崩す事となる．そのため，Meteor で利用するリンクエミュレータはネットワーク層と物理層の間に位置するデータリンク層で遅延挿入や帯域制限が行えることが条件となる．そして，リンクエミュレータがデータリンク層で動作するためには，計算機の判別にデータリンク層の識別子である MAC アドレスが使用できることも条件となる．また，大規模なネットワークを構築するために，1 回の遅延時間や帯域幅に対する変更処理がノード数によって増加しないことが重要となる．

4.2 ユーザ空間でのリンクエミュレータ

遅延や帯域の制限を行うトラヒック制御は，計算機が送受信するトラヒックをネットワークバッファに一時的に滞留，破棄を行うことで実現している．ユーザ空間でリンクエミュレータを実装するためには，カーネル空間からユーザ空間へパケットを横取りする必要がある．これは Netfilter nfqueue [13] や Divert Socket などを用いることにより実現できる．ユーザ空間へパケットを横取りした場合，カーネル空間で実装することと比較して容易に実装が可能である．ユーザ空間へパケットを横取りした場合，カーネル空間からユーザ空間へのメモリコピーが発生するため，オーバーヘッドが生じる．これらのパケット横取り手法を用いたネットワークエミュレータとして GINE [14], [15] が提案されている．GINE は，複数の仮想ルータをノード内に作成し，仮想ルータ間のトラヒック制御を行うエミュレータであるが，広帯域な回線をエミュレーションした際にスループットの低下が見られている．

4.3 カーネル空間でのリンクエミュレータ

カーネル空間で実装されているリンクエミュレータでは，ユーザ空間へのメモリコピーが発生しないため，オーバーヘッドは小さくなる．その反面，実装が難しくなりカーネル実装が変更された際に動作しなくなる可能性がある．

Linux や FreeBSD，MacOS X では，OS 標準の機能としてリンクエミュレータが実装されている．カーネル空間で動作する代表的なリンクエミュレータとして TC/Netem [16] や NISTNet [17]，Dummynet が実装されている．これらのリンクエミュレータは無線空間を再現する機能をもっていないが，多くのネットワークエミュレータはこれらの機能を柔軟に制御することにより無線空間の再現を行っている．これらは，各ディストリビューションでメンテナンスされており，カーネルの実装が変わったとしても制御可能である．本節では，これらのリンクエミュレータについて解説を行う．

4.3.1 NISTNet

NISTNet は National Institute of Standards and Technology (NIST) が設計・実装したリンクエミュレータである．NISTNet はネットワーク層の入出力パケットをカーネルモジュールで横取りし，トラヒック制御を行う．

4.3.2 TC/Netem

Linux にはトラヒック制御機構として TC/Netem が採用されている．Netem は後述する遅延制御を行うモジュールであるが，Linux のトラヒック制御機構を総称して指すことが多い．以降，本論文では TC/Netem をトラヒック制御機構そのものを指し，Netem は遅延制御モジュールを指すものとする．TC/Netem の制

御用ユーティリティとして tc [18] が提供されている．TC/Netem はインタフェースごとの送信キューである qdisc でフレームを滞留，破棄することでトラヒック制御を実現している．Qdisc はネットワークインターフェースの送信キューであるためトラヒック制御を行う単位はパケットではなくフレームとなる．また，基本的に送信フレームのみ制御可能であり，受信パケットを扱うためには受信用のモジュールと Intermediate Functional Block device (IFB) インタフェースを使用する．

Qdisc は木構造で管理するため qdisc を連結することが可能であり，様々なモジュールを組み合わせることができる．Qdisc のモジュールには遅延挿入，パケットロスのための Netem，帯域制限を行うための Token Bucket Filter (TBF) や Class Base Queuing (CBQ)，Hierarchical Token Bucket (HTB) などがある．また，qdisc にはクラスフルとクラスレスの概念があり，クラスフルなモジュールはトラヒックをクラス別にフィルタリングし各クラスでトラヒック制御を行う．Netem や TBF，ingress はクラスレスであり，CBQ と HTB はクラスフルとなる．クラスフルな Qdisc をフィルタリングするには，MAC アドレスや IP アドレス，ポート番号などを指定し転送先の qdisc を指定する．

4.3.3 Dummynet

FreeBSD や MacOS X では，Dummynet を用いた遅延挿入や帯域制限が可能である．Dummynet は IPFW のユーティリティとして実装されている．IPFW は，プロトコルスタック内でデータリンク層の入出力を行う ether_demux，ether_output_frame，ブリッジ処理を行う bdg_forward，ネットワーク層の入出力を行う ip_input，ip_output のいずれかで IPFW の処理に渡される．Dummynet は TC/Netem とは異なり，単体で遅延生成と帯域制限の機能をもっているが，遅延の値や帯域の制限値に固定値しか指定できない．これをランダムに変更したい場合，複数のフィルタルールを設定し，確率的にマッチさせることでジッタの再現を実現する必要がある．また，フィルタリングの識別子として，IP アドレスのみの指定となっている．

4.4 Meteor のリンクエミュレータ

無線空間では遅延時間や帯域幅が刻一刻と変化するため，短い時間で遅延時間や帯域幅を変更し続けることが求められる．ユーザ空間におけるリンクエミュレータの実装は，メモリコピーによるオーバーヘッドが精度に大きく影響すると考え，カーネル空間における実装を使用する．カーネル空間で動作可能なリンクエミュレータには TC/Netem と Dummynet が存在するが，Dummynet はネットワーク層で動作するリンクエミュレータである．そのため，イーサネットフレームに対する遅延挿入や帯域制限ができない．本論文で提案している Meteor はデータリンク層で動作することが重要であるため，Meteor のリンクエミュレータとして利用できない．TC/Netem は，各ネットワークインターフェースの送信キューを利用するため，データリンク層で動作しイーサネットフレームに対する制御が可能である．また，Dummynet は遅延時間にミリ秒単位での指定であるのに対し，TC/Netem はマイクロ秒単位の指定が可能である．以上より Meteor においてはリンクエミュレータとして TC/Netem を使用する．

4.5 エミュレーション時のトラヒック制御

ネットワークエミュレータを使用するユーザが環境を自由に構築できるべきであるということは **3.** で述べた．本節では，無線空間を作り出す上で考慮しつつ Meteor の動作時のトラヒック制御方法について検討する．

ネットワーク上での通信は，ユニキャストやブロードキャスト，マルチキャストに分類される．ユニキャストは 1 対 1 の通信であるため，送信元アドレスと送信先アドレスを指定することでフィルタリングの適用が可能である．しかし，ブロードキャストやマルチキャストのような 1 対多の通信では，送信先アドレスによる計算機の識別が不可能となる．TC/Netem は送信時にフィルタリングを行うため，1 対多の通信はネットワークエミュレータ上を通過するフレームの向きを考慮する必要がある．

図 3 は，自ノード以外の送信元からの通信を全てフィルタリングし，他の計算機からのトラヒックに対し遅延時間や帯域制限を行った際の処理を示している．図中の赤線は，NIC でイーサネットフレームを受信してからアプリケーションソフトウェアが受け取るまでの処理の流れとアプリケーションソフトウェアから送信されたイーサネットフレームが NIC から送信されるまでの処理の流れを示している．先に述べたように，計算機から送信するフレームに対して遅延挿入や帯域制限を行うとブロードキャストトラヒックをフィルタリングできない．そのため，受信時に送信元 MAC アドレスでフィルタリングすることにより，ブロード

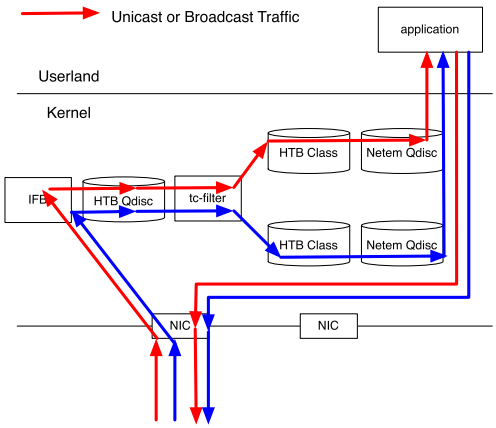

図 3　計算機上で動作時のトラヒック制御
Fig. 3　Traffic control in an end node.

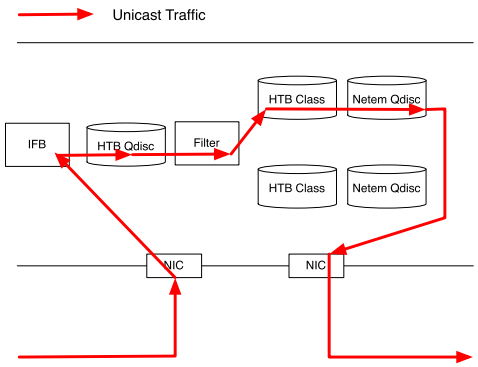

図 4　中間計算機でのトラヒック制御
Fig. 4　Traffic control in a bridge node.

キャストに対応可能とする．

TC/Netem は，**4.3.2** で述べたように送信フレームにのみトラヒック制御が可能であるため，全ての受信フレームを一旦 IFB インタフェースへ転送する．そして，IFB インタフェースからイーサネットフレームを再送信することで，受信フレームに対してトラヒック制御を行う．IFB インタフェースから再送信されるイーサネットフレームは送信処理時に HTB Class への振り分けのため tc-filter を通過する．フィルタリングの識別子には MAC アドレスや IP アドレス，任意のデータ識別子が指定可能であり，tc-filter にマッチしたフレームは対応する HTB Class へと転送される．HTB Class では，帯域制限が行われ，デキューしたフレームは次に Netem Qdisc へと転送される．Netem Qdisc では，エンキュー時にパケット破棄の判定が行われる．破棄されずに Qdisc にエンキューされたフレームは指定時刻までキュー内に滞留し，その後アプリケーションへ送信される．

ここまで，ユニキャスト，ブロードキャストフレームを送信するアプリケーションソフトウェアを対象にエミュレーション手法を述べてきた．しかし，無線空間を通信する計算機が決まっているのであれば，全ての計算機上で Meteor を動作する必要はない．Meteor を動作させる専用の計算機を用意し，1 対 1 のリンクを多数作り出すことでネットワークエミュレータの処理量を削減可能である．そこで Meteor では，ネットワークエミュレータを動作させる計算機を専用に設置し，アプリケーションソフトウェアが動作している計算機からのトラヒックをブリッジする動作の実装も行った．

図 4 はブリッジでのトラヒック制御を行う際の動作を示している．受信したフレームは，ネットワークインタフェースより IFB インタフェースへ転送される．IFB は受信したフレームを自身の Qdisc (HTB Qdisc) へエンキューし，送信する時刻まで待つ．HTB Qdisc からデキューされたフレームは送信元アドレスと送信先アドレスでフィルタリングされ，次の HTB クラスへ転送される．HTB Class は計算機間の帯域幅を制限する class モジュールであり，ここで帯域計算が行われる．Netem Qdisc では，エンキュー時に確率的にフレームが破棄され，破棄されなかったフレームのみエンキューされる．フレームの送信時刻は Netem へのエンキュー時に埋め込まれる．Linux のスケジューラにより，フレームが Qdisc からデキューされた後，埋め込まれた送信時刻を確認し，送信時刻に達していなければ Netem Qdisc へリキューし，送信時刻を過ぎていれば送信処理に移る．IFB インタフェースからのフレーム送信時に，自身の Forwarding DataBase (FDB) を確認し，送信する物理インタフェースの検索を行う．

4.6　フィルタリング

ネットワークエミュレータが扱うことのできる送信元計算機と送信先計算機の識別子として MAC アドレスや IP アドレスが挙げられる．Meteor は，データリンク層で動作する必要があることから，Meteor には MAC アドレスでのフィルタリング機能を実装している．しかし，MAC アドレスのみのフィルタリングでは IP アドレスでのフィルタリングで十分な検証でも計算機ごとの MAC アドレスを調べる必要がある．また，ハードウェアを変更した際に識別子を指定し直す

必要が出てしまい，環境の可搬性を損なってしまう．そのため，Meteor の実装は IP アドレスによるフィルタリング機能も必要であると考え，Meteor ではフィルタリングの識別子に MAC アドレスと IP アドレスのどちらでも指定可能とした．

このような MAC アドレスや IP アドレスでのフィルタリングは，汎用計算機上でアプリケーションソフトウェアが動作することが前提となっている．しかし，組み込み機器などを動作させるために，Cooja/MSPSim [19] や mac80211_hwsim [20] などを使用する場合が考えられる．これらの技術は，汎用計算機上で組み込み機器を動作させ，無線フレームを送受信可能とする．無線フレームは有線ネットワーク上に送信できないため，計算機から有線ネットワークへ送信する際に，トランスポート層でカプセル化を行う．MAC アドレスや IP アドレスでフィルタリングを行うと無線フレームを送信したインタフェースの情報はカプセル化により隠蔽されてしまうため，送信元，送信先計算機を判断することができない．このようなカプセル化を行う通信では，カプセル化の形式やデータのペイロードは実装依存となるため，フィルタリング機能としてユーザが自由に識別子を指定できる必要がある．Meteor では，カプセル化された通信に対応するため，データリンク層のヘッダ情報からのオフセット値と 16 進数の値を指定することで，MAC アドレスや IP アドレスの他に自由な形でフィルタリングルールを記述可能とした．

4.7 タイマ制御

本論文で提案している Meteor は汎用計算機，汎用 OS での動作を想定しているため，リアルタイム性が保証されない．遅延時間や帯域幅を変更する時刻から実際に変更処理が始まるまでの時間はマイクロ秒オーダであることが望ましい．ネットワークエミュレータにおけるタイマ制御は，構築可能なネットワークの規模に影響するものではないものの，指定時刻から大幅に遅れて設定の変更が行われてはエミュレータの精度に影響が出てしまう．無線空間の遅延時間や帯域幅の変動を忠実に作り出すには，高精度なタイマが求められる．

タイマ制御には，タイムスタンプカウンタ (TSC)，sleep 関数，イベント駆動型などの方法が考えられる．sleep 関数やイベント駆動型は，変更時刻までの待機時間に CPU は使われない．sleep 関数は，カーネルの動的タイマとスケジューラを使用して実装されているため，ハードウェアタイマが Programable Interval Timer (PIT) の場合はソフトウェアのタイマ割り込み周期よりも短い時間での復帰はできない．例えば，タイマの割り込み周期が 250Hz であった場合，sleep からの復帰には 4ms 以上の時間が必要となる．より高精度なハードウェアタイマである High Precision Event Timer (HPET) を使用することでマイクロ秒オーダでの割り込みが可能である．しかし，HPET を使用するとハードウェア割り込みの回数が増加するため，CPU 負荷が高くなる．また，sleep 関数による待機方法は，指定時間プログラムの実行を遅延させるものであるため，シナリオに記述されている時間で sleep を実行するとエミュレータの動作時間が長くなってしまう．例えば，Meteor の設定変更に必要な時間が 1ms，次の設定変更時間が 10ms 後の場合，1ms の設定変更が行われた後に 10ms 待機するため，実際には 11ms 後に次の設定変更が行われる．sleep 関数によるタイマ制御を行うためには，Meteor 自身の処理時間を計測した上で待機時間を計算する必要がある．

TSC によるタイマ制御は，CPU のクロックカウンタを調べながら指定したクロックカウンタまで待機する方法である．TSC の読み出しには RDTSC と呼ばれる CPU 命令を使用する．TSC によるタイマ制御は CPU のクロックカウンタをベースに動作するため最も精度が高い．また，設定変更時間をクロックカウンタの値で指定するため Meteor の処理時間に関係なく動作することができる．一方で，SpeedStep や TurboBoost のような CPU 周波数を動的に変更する機能が有効になっていると，クロックカウンタの増加値が変化するため正確な時間管理ができなくなる．そのため，TSC によるタイマ制御を行う際には，SpeedStep や TurboBoost を無効にしておく必要がある．

HPET を用いた sleep 処理で必要となる Meteor 自身の処理時間は gettimeofday で計測可能であるが，gettimeofday はシステムコールで重い部類の処理となる．そして，Meteor の処理時間は設定を変更するたびに計測する必要があるため，計測処理のみでシステムコールの呼び出し回数が増大してしまう．我々は，gettimeofday による負荷が大きいと考え，待機時間の指定が容易な TSC によるタイマ制御を使用することとした．

4.8 Meteor の実装

これまでの設計に基づき Meteor の実装を行った．Meteor は，C 言語で実装されておりタイマ制御を行

うライブラリである libtimer とシナリオパーサ，トラヒック制御ライブラリである libtc から構成される．libtimer は，TSC を用いて，シナリオに記述されているイベント時刻まで待機するライブラリである．シナリオパーサは，XML で記述された QOMET のシナリオファイルとの互換性ともたせるため，XML パーサライブラリである libexpat を用いている．既存の TC/Netem を利用したトラヒック制御は，libnetlink や libnl [21] を用いることでユーザランドから制御可能であるが，libnetlink は netlink socket を扱うのみであり，Qdisc を制御するためのプログラムは自身で実装する必要がある．一方で，libnl はトラヒック制御に関して未実装な部分があり，Meteor を実装するにあたり不十分であった．そこで本研究では，Meteor の実装にあたりトラヒック制御用ライブラリとして libtc の実装を行った．libtc は C 言語で実装されており，netlink socket を使用するため libnetlink を使用している．Ubuntu 12.04，14.04，Gentoo Linux (Kernel 2.6.32) にて動作確認を行った．なお，libtc や libtimer を含む Meteor は github にて公開している．

5. Meteor の評価

5.1 機能面の評価

Meteor の評価として，QOMET と MobiNet を比較し，機能面での評価を行った．評価項目は，ネットワークエミュレータが動作するレイヤや扱うことのできる計算機の識別子とした．表 1 に機能評価の結果を示す．

ネットワークエミュレータが扱うことのできるレイヤは，Meteor がデータリンク層であるのに対し，QOMET はネットワーク層で動作する．MobiNet は遅延挿入や帯域制限はネットワーク層で行うが，計算機の間で MAC レイヤのエミュレーションを行うため，動作するレイヤをデータリンク層とネットワーク層の二つとした．しかし，MobiNet の MAC エミュレーションは無線チャネルを模倣する機構であり，計算機間の遅延や帯域幅はネットワーク層で行っているため，イーサネットフレームや IPv6 パケットに対する遅延挿入や帯域制限はできない．ネットワークエミュレータが扱う計算機の識別子として，Meteor は MAC アドレスと IP アドレスを計算機の識別子として用いることができる．MAC アドレスを識別子としてイーサネットフレームに対して遅延挿入や帯域制限を行うことにより，Meteor は IP ヘッダをもたないイーサネットフレームや IPv6 パケットに対応可能としている．対して，QOMET と MobiNet では IP アドレスのみを識別子として用いている．そのため，イーサネットフレームや IPv6 パケットに対する遅延挿入や帯域制限を行う機能はもっていない．以降では，まずデータリンク層のプロトコルとして，IEEE802.11s と B.A.T.M.A.N が検証可能であることを示す．次に，OLSR(HNA6) 等，IPv4 ヘッダを持たないフレームを利用する場合も検証可能であることを述べる．

表 1　ネットワークエミュレータの機能評価
Table 1　Functional evaluation of network emulator.

	Meteor	QOMET	MobiNet
レイヤ	データリンク層	ネットワーク層	データリンク層 ネットワーク層
識別子	MAC アドレス IP アドレス	IP アドレス	IP アドレス

5.2 Meteor の処理時間

本節では，Meteor がパラメータの変更を行うのに必要な処理時間について評価を行う．Meteor は，大規模な実験ネットワークで動作可能なネットワークエミュレータとして実装を行った．大規模な実験ネットワークで動作可能とするためには，ネットワークエミュレータが遅延時間や帯域制限の設定を変更する際の処理時間が重要となる．設定変更の処理時間が長いと無線空間を忠実に再現することが難しくなるためである．図 5 に 10 ノードから 5000 ノードまで 100 ノード単位でノード数を増加させた無線環境をそれぞれエミュレーションした際の設定変更に必要な時間を示す．1 台のノードが設定変更する数は自ノードを省いた N − 1 である．縦軸がパラメータ変更開始から終了までの処理時間，横軸がリンク数である．計測には gettimeofday を使用し，設定前と設定後の時刻差分を算出している．計算機は北陸 StarBED 技術センターのグループ I を使用した．グループ I の構成は表 2 のとおりである．エミュレータノードは，OS に Ubuntu 14.04 をインストールし二つのインタフェース間で Meteor を動作させた．

Meteor が一度 TC/Netem へ設定するまでに必要な時間は約 17 μs であった．これは，1 対 1 のノード間での通信を設定するために必要な時間である．複数の対向計算機を扱う場合，Meteor は連続して TC/Netem の制御を行うため，ノード数の増加に対して処理時間は線形に増加する．そのため，1 台の Meteor が 4999 台の計算機への接続をエミュレーションするにはパラ

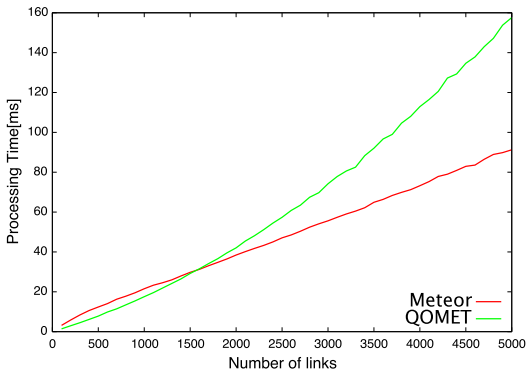

図 5　Meteor の設定時間
Fig. 5　Meteor's processing time.

表 2　実験ノードのスペック
Table 2　Experiment node spec.

Model	Cisco UCS C200 M2
Chipset	Intel 5520 (Tylersburg) chipset
CPU	Intel 6-Core Xeon X5670 x 2
Memory	48GB
NIC	Broadcom 5709 Quad Port

メータ変更時間は約 90ms であることがわかる．これに対し，QOMET は 1500 台以下の計算機への変更であれば Meteor と比較し短い時間で動作している．しかし，計算機の台数が増加すると徐々に処理時間が増加していき，1500 台以降は Meteor より処理時間が長くなっている．これは，Meteor で使用している TC/Netem が構成情報を木構造で管理しているのに対し，QOMET で使用している Dummynet はリストで管理しているためであると考えられる．

TC/Netem と Dummynet を比較した場合，カーネルモジュールに送信するまでの処理は TC/Netem のほうが複雑である．そのため，1 台の計算機への設定時間のみを計測すると TC/Netem を利用する Meteor の負荷が高くなる．今回使用した計算機では，1500 台までの環境で QOMET のほうが高速に動作している．しかし，Dummynet が構成情報をリストで管理しているため，扱う計算機の数が増加するとカーネルモジュールの負荷が高くなる．結果，Meteor では計算機の数に対して線形に増加しているのに対して，QOMET では徐々に処理時間が増加している．そのため，1500 台以下であれば QOMET が高速に動作するが，1500 台以上であれば Meteor が有利となる．

5.3　単純な遅延変化での模倣性能

本論文で設計・実装した Meteor の精度を計測するため評価実験を行った．評価環境は，図 6 に示すように，トラヒックジェネレータである Spirent Communications SmartBits 600B とエミュレータノードを 2 本の UTP ケーブルで接続した．実験には，IPv4 ヘッダを含めた状態で SmartBits 600B が送信可能な最小サイズである 70Bytes とし，1 秒間の送信フレーム数は 802.11g の最大帯域となる 54Mbps を送信した場合の 96kfps とした．送信フレーム数やフレームのサイズを変更した実験も行ったが，結果に差異が見られなかったため，本検証結果には最小フレームサイズで最大送信数となる 96kfps の結果のみとした．SmartBits 600B から送信されたイーサネットフレームはエミュレータノードで遅延挿入，帯域制限が行われ SmartBits 600B に折り返す構成となる．SmartBits 600B から送信されるイーサネットフレームには，送信時刻が埋め込まれており，受信時に受信時刻との差分を出すことで遅延時間を算出した．

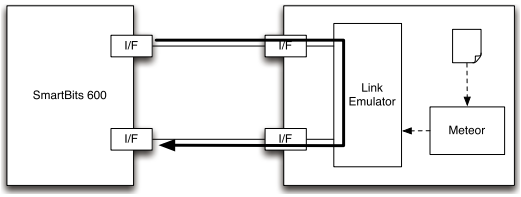

図 6　検証環境
Fig. 6　Experiment environment.

Meteor の設計から，遅延時間の変動幅によってネットワークエミュレータとしての精度に影響があると考えられる．例えば，遅延時間の変動が小さければ変更時にフレーム送信処理の負荷も小さくなると考えられるため精度は高くなる．一方で遅延時間が大幅に短くなれば，キューに滞留しているフレームを一斉に送信する必要があり精度が低下すると考えられる．本節では遅延時間を緩やかに変動させたシナリオでの精度を検証した．シナリオは遅延時間が 0ms から始まり 25 秒後に最大値となる 1000ms に達し，その後 0ms まで遅延時間が減少していく正弦曲線をシナリオとした．

図 7 は，実験結果の一部を抜き出したグラフである．縦軸が遅延時間 [ms]，横軸がイーサネットフレームの送信時刻 [s] である．シナリオのグラフに対し Meteor の結果が，約 100us 程度遅延時間が大きくなっている．これは，Meteor が挿入する遅延時間の他にイーサネットフレームを転送するための処理時間とエミュレータノードと SmartBits 600B との間の伝送時間が別途含

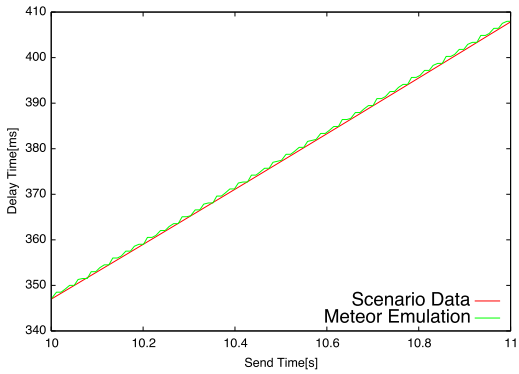

図 7　シナリオデータと計測結果 “正弦曲線”
Fig. 7　Result of SmartBits 600B and scenarios delay time. Sine curve.

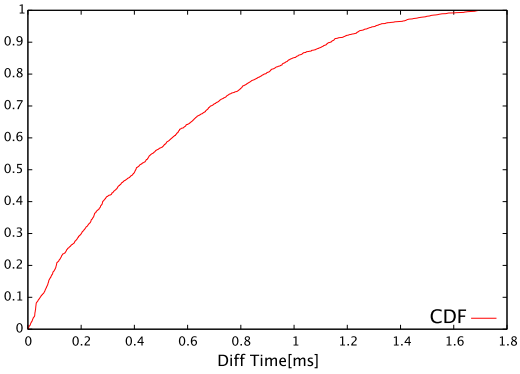

図 8　シナリオデータと計測結果の差分 (CDF) “正弦曲線”
Fig. 8　Difference of delay time (CDF). Scenario data and result of a measurement. Sine curve.

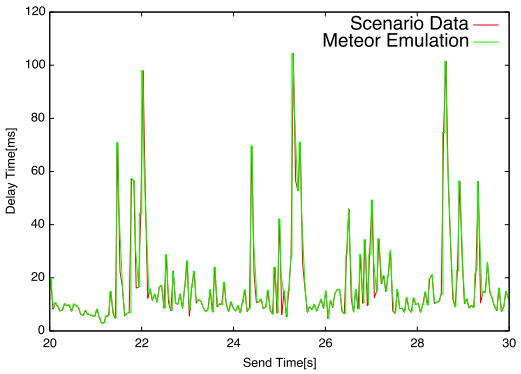

図 9　シナリオデータと計測結果 (20s - 30s) “LTE での通信時”
Fig. 9　Result of SmartBits 600B and scenarios delay time (20s - 30s). LTE communicaiton.

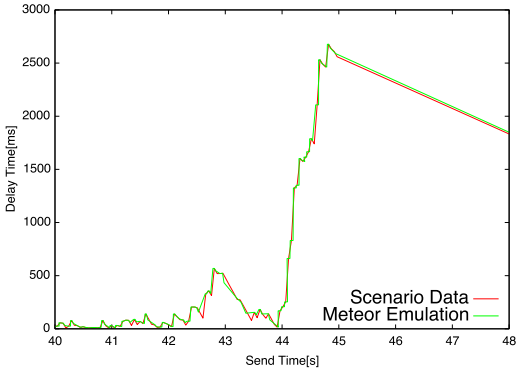

図 10　シナリオデータと計測結果 (40s - 48s) “LTE から 3G へのハンドオーバ”
Fig. 10　Result of SmartBits 600B and scenarios delay time (40 - 48s). Handover from LTE to 3G.

まれるためだと考えられる．また，Meteor の結果が段階的に増加しているのは，Meteor が 25ms 間隔で遅延時間を増加させているためである．

図 8 は遅延時間を緩やかに変動させたシナリオと Meteor の結果の差分を表した CDF グラフである．縦軸が累積確率密度，横軸がシナリオと実験結果の遅延時間 [ms] の差分である．図 7 のシナリオは，遅延時間の変動が緩やかであるため，遅延時間の差分は 1.8ms 以下に収まっており，最大遅延時間から見ると 0.18%の差分となる．アプリケーションソフトウェアの検証を行うにあたり，0.18%の差分は影響がない範囲であると考えられる．

5.4　実データに基づく模倣性能

本節では，遅延時間の変動幅が大きなシナリオを用いて計測を行った．検証環境は，**5.3** と同様の環境を用いた．図 9，図 10 は，縦軸が遅延時間，横軸がパケットの送信時刻とする実地計測データを用いて Meteor を動作させた上での Meteor の結果と重ねたグラフである．前者は，LTE で通信時のシナリオであり，数 ms から 100ms 程度まで遅延時間が連続で変動している．後者は，LTE から 3G へのハンドオーバが起こるシナリオであり，遅延変動の振幅が大きく，シナリオ後半ではハンドオーバにより 45s 以降通信切断が起きている．

シナリオデータは，電波状況により遅延時間が大きく変動しているが，Meteor によるエミュレーションの計測結果も追従して遅延時間が変動していることがわかる．しかし，遅延時間が大きく減少するタイミングにおいて一部追従できていない部分がある．これ

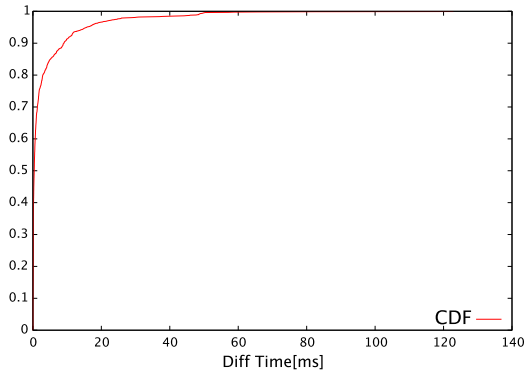

図 11　シナリオデータと計測結果の差分 (CDF) “LTE から 3G へのハンドオーバ”
Fig. 11　Difference of delay time (CDF). Scenario data and result of a measurement. Handover from LTE to 3G.

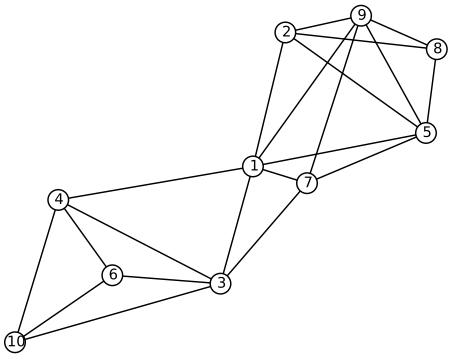

図 12　10 台でのメッシュネットワーク
Fig. 12　10 nodes mesh network.

表 3　マルチホップ通信時の遅延時間
Table 3　Delay of multi-hop communicaiton.

Protocol	1 Hop	2 Hop	3 Hop	4 Hop
AODV	32.1	97.4	140.2	153.7
802.11s	32.3	98.4	142.0	158.0
B.A.T.M.A.N	32.3	150.0	229.0	294.0
OLSR (HNA6)	32.4	98.0	141.0	154.9

は，長い遅延時間から短い遅延時間へ変更した際に，キューの中には送信時刻まで待機しているパケットが滞留しているため，後からエンキューされた短い遅延時間のパケットがデキューできないため，遅延時間の減少に追従できないものと考えられる．

次に，LTE デバイスを用いた実地計測のシナリオデータと Meteor によるエミュレーションの計測結果の差分を CDF で表したグラフを図 11 に示す．縦軸と横軸は図 8 と同じである．図 10 のシナリオは，ハンドオーバが起こる直前に大きな遅延変動が起こっているため，図 8 と比較して遅延時間の差分が大きくなっている．計測結果の中で 1ms 以下の差分が 65%，10ms 以下の差分時間か 90%となっている．Meteor 動作時に遅延時間の減少に追従できていない部分が 90%以上に表れていると考えられる．

5.5　Meteor を用いたメッシュネットワークの構築

本節では，Meteor を用いた構築したネットワーク上で 802.11s，AODV，B.A.T.M.A.N，OLSR のルーティングプロトコルを動作させ，アドホックネットワーク上でのマルチホップ時に遅延時間の変化の計測を行った．図 12 は，10 台の計算機を用いて構築したメッシュネットワークである．

802.11s には，open80211s [22] を使用した．open80211s を使用するためには無線インタフェースが必要となるため，仮想インタフェースを作成できる mac80211_hwsim も用いている．

表 3 は，各ルーティングプロトコル動作時のホップ数ごとの遅延時間である．各ルーティングプロトコルでホップ数が増加するごとに遅延時間も増加していることがわかる．B.A.T.M.A.N のみ遅延時間の増加が大きくなっているのは，データの転送処理が他のルーティングプロトコルと比較して大きいためであると考えられる．

OLSR (HNA6) は，IPv6 を用いて動作させた結果である．Meteor は，ネットワーク層のプロトコルに依存しないネットワークエミュレータとして実装を行ったため，IPv6 や B.A.T.M.A.N のような，IPv4 ヘッダをもたないフレームにも対応できている．

5.6　大規模ネットワークの構築

本節では，仮想計算機 1024 台を用いて格子状のネットワークを構築し，Meteor の動作検証を行う．図 13 は，1024 台の仮想計算機を格子状に配置したネットワーク図である．各計算機間の距離は 30m とし，このときの一回の転送遅延は 30ms とした．

検証には，OLSR (HNA6) を使用した．AODV は，RFC にて最大ホップ数が 35 までと決められている．そのため，1024 台の計算機を格子状に配置した場合，最大ホップ数は 63 となり，AODV では到達できない規模のネットワークとなる．Meteor のみを動作させたときの CPU 使用率は 8%であったが，OLSR を動作させると CPU 使用率が 100%となり，パケットロスや想定より大きな遅延が発生した．OLSR の HELLO メッセージは標準の設定では 6 秒間に 1 回であるた

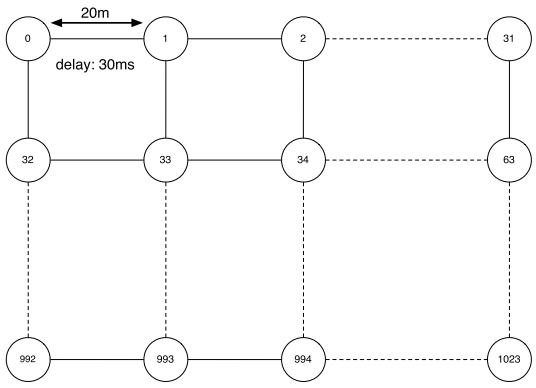

図 13　1000 台でのメッシュネットワーク
Fig. 13　1000 nodes mesh network.

め，HELLO メッセージのみであればトラヒック負荷は高くない．しかし，OLSR を 1000 台規模で動作させると大量の Topology Control (TC) メッセージが送信される．そして，TC メッセージを受信するたびに経路表の更新が行われるため，各計算機の負荷が高くなっていると考えられる．

6. 議　　論

6.1　スケーラビリティ

Meteor は複数の計算機間のリンクを無線通信のように遅延挿入や帯域制限を行うが，制御可能なリンク数によって生成可能な無線空間の規模が決まる．Meteor が使用する Qdisc の ID 空間は 16bit であるため，論理的に扱えるリンク数は 65535 となる．これは，Meteor の本質的な制限ではないものの，TC/Netem を扱っている上でスケーラビリティの限界となる．また，リンク数の増加に従い TC/Netem へ遅延時間や帯域制限の変更にかかる時間も線形に増加するため，変更間隔とリンク数はトレードオフの関係にある．そのため，非常に短い時間で変更を行う場合は，扱うリンク数を少なくして Meteor を動作させる必要がある．実際の無線ネットワークの再現は，**5.4** にて LTE での通信時における実地計測データを用いて検証を行った．この検証では，25ms 間隔で設定を変更しているが，シナリオデータと SmartBits 600B の結果から遅延時間の再現ができていると言える．また，**5.2** での結果より，25ms の変更間隔であれば，約 1000 台の計算機を扱うことが可能である．したがって，全ての計算機が常に移動する状況においても，約 1000 台までのエミュレーションが可能である．また，アプリケーションソフトウェアを動作させる計算機間の遅延時間やパケットロス値は常に変動するという訳ではない．例えば，パケットロス率が 100%となる計算機間の設定を変更し続ける必要はない．そのため，計算機間の通信品質が変化したもののみ Qdisc の変更を行うことにより Meteor の処理時間を削減することが可能であると考えられる．また，Meteor が論理的に扱えるリンク数に関しても，通信が行えるもののみを扱えば処理時間の削減が可能であると考えられる．複数の計算機上で Meteor を協調動作させる場合，シナリオの同期や制御方法などを考慮する必要があると考えられるが，複数のネットワークエミュレータの制御方法は今後の課題である．

6.2　エミュレータの精度

Meteor による遅延制御はシナリオに記述された遅延時間の変動値により大きく影響を受ける．特に遅延時間が減少するタイミングで追従性が低下している．遅延時間が追従できていない部分では，次の遅延時間まで低下している．これは，遅延時間の変動が大きいほど追従性の低下が起きている傾向がある．この追従性の低下は，キューに滞留しているイーサネットフレームが送信される前に次の遅延時間に変更された結果，短い遅延時間で設定されたイーサネットフレームが送信できないものと考えられる．Meteor の実装では，イーサネットフレームのリオーダは許容しないため，このような動作となっている．これは，ネットワークエミュレータ側でリオーダが発生する現象がアプリケーションソフトウェアに対して大きな影響を与えると考えたためである．

ネットワークエミュレータとして遅延時間の追従性を重視するのであれば，リオーダ若しくはキューに滞留しているイーサネットフレームを破棄し，次のイーサネットフレームを送信する動作にしなければならない．遅延時間の追従性やリオーダの許容，イーサネットフレームの破棄のような動作に関しては想定してる環境によるため，Meteor の動作として拡張の余地がある．

6.3　フィルタリング機能

Meteor は，IEEE 802.11s や AODV のようなデータリンク層で動作するプロトコルに対応できるネットワークエミュレータとして設計･実装を行った．これにより，これまで困難であったメッシュネットワークの検証も可能となる．しかし，Internet of Things (IoT) で使用するデバイスには汎用計算機上で動作しないも

のも存在する．また，802.11 イーサネットフレームなどは有線ネットワーク上には送信できないため，完全な無線空間を作り出しているとは言えない．

IoT で使用するデバイスを汎用計算機上で動作させるためのエミュレータは幾つか実装されている．Cooja/MSPSim [19] は汎用計算機上で様々なセンサーノードを動作させるエミュレータである．このセンサーノードから送信されたパケットは，計算機上でカプセル化され有線ネットワークへ送信される．

無線ネットワークで使われる radiotap を作り出す実装は，mac80211_hwsim がある．mac80211_hwsim は，無線インタフェースをもたない計算機に IEEE 802.11 フレームの送信・受信を行う radiotap インタフェースと仮想無線インタフェースを作成する．この仮想インタフェースを使用することで，有線ネットワーク上でも無線インタフェースを用いることが可能となる．しかし，mac80211_hwsim も有線ネットワーク上に IEEE 802.11 フレームを送信するために計算機上で IEEE 802.11 フレームに対しカプセル化を行う．これらのようなカプセル化されたフレームは，有線ネットワーク上に送信されたときにペイロード部分に隠蔽されてしまうため，MAC アドレスや IP アドレスでは識別できない．

Meteor は，これらの IEEE 802.11 フレームに対しても対応できるようフィルタリングルールに 16 進数でデータの値を指定できるようにしている．しかし，フレーム内のデータの位置の値を直接指定するためユーザがヘッダやペイロードの内容を詳細に知っている必要がある．このようなカプセル化されたイーサネットフレームに対するフィルタリングルールの指定方法を今後検討していく必要があると考えている．

7. むすび

インターネットへの接続端末が多様化する中，ネットワークテストベッド上に擬似的な無線空間を作り出すネットワークエミュレータが数多く提案されている．しかし，新たな無線技術の提案によりネットワーク層で動作しているネットワークエミュレータでは対応が難しくなってきている．

そこで我々は，無線空間における遅延時間や帯域制限を忠実に再現するとともに，アプリケーションソフトウェアのプロトコルに依存しない無線空間のエミュレーション手法の提案を行った．また，提案手法を実現するネットワークエミュレータである Meteor の設計・実装を行った．Meteor は，既存研究の QOMET や MobiNet と比較しネットワーク層での動作からデータリンク層での動作にしたため，これまで検証が困難であった IEEE802.11s や B.A.T.M.A.N のようなアプリケーションソフトウェアの検証が可能となった．

既存のネットワークエミュレータは，計算機の識別にネットワーク層の識別子である IP アドレスが用いられているが，Meteor ではユーザがパケットの中のデータを自由に指定できる形でアプリケーションの実装依存となる特殊なデータ形式においても対応可能としている．識別子としての指定方法は，16 進数の値で指定するため実験者が容易に設定できるとは言い難いが，Cooja/MSPSim や mac80211_hwsim のようなカプセル化されたパケットにおいてもエンドノードの識別が可能である．

Meteor を用いたネットワークエミュレータの精度は，遅延時間が大きく減少する際に追従性が低下する傾向が見られるものの，ハンドオーバが発生した際の遅延時間の増加に対しても再現できている結果が得られた．本提案を用いることにより，これまで困難であったデータリンク層のプロトコルを使用するアプリケーションソフトウェアの検証が可能な擬似的な無線空間を作り出すことが可能となる．

謝辞　本論文を執筆する上で，有益な助言を頂いた北陸先端科学技術大学院大学知念賢一准教授，NICT 北陸 StarBED 技術センターの三輪信介博士，安田真悟博士，三浦良介氏，パナソニック株式会社村本衛一氏に感謝します．

文　献

[1] B. Chun, D. Culler, T. Roscoe, A. Bavier, L. Peterson, M. Wawrzoniak, and M. Bowman, “Planetlab: An overlay testbed for broad-coverage services,” ACM SIGCOMM Computer Communication, vol.33, pp.2–13, July 2003.

[2] R. Beuran, L.T. Nguyen, K.T. Latt, J. Nakata, and Y. Shinoda, “Qomet: A versatile wlan emulator,” IEEE International Conference on Advanced Information Networking and Applications (AINA-07), pp.21–23, 2007.

[3] G. Hiertz, D. Denteneer, S. Max, R. Taori, J. Cardona, L. Berlemann, and B. Walke, “IEEE 802.11s: The wlan mesh standard,” IEEE Wireless Communications, pp.104–111, Feb. 2010. http://www.comnets.rwth-aachen.de

[4] Open-Mesh.net, “B.a.t.m.a.n. (better approach to mobile ad-hoc networking),” http://www.open-mesh.net/

[5] K. Akashi, “Meteor,” https://github.com/k-akashi/meteor, 2011.
[6] M.-M. Moazzami, “Orbit: A smartphone-based system platform for embedded sensing applications,” Technical Report MSU-CSE-13-11, Department of Computer Science, Michigan State University, East Lansing, Michigan, Dec. 2013.
[7] N.A. Technology, http://ngw.ntt-at.co.jp/product/tester/products/Shunra.html
[8] A. Systems, http://www.ixiacom.jp/products/applications/anue-3500, 2014.
[9] P. Mahadevan, A. Rodriguez, D. Becker, and A. Vahdat, “Mobinet: A scalable emulation infrastructure for ad hoc and wireless networks,” SIGMOBILE Mob. Comput. Commun. Rev., vol.10, no.2, pp.26–37, April 2006. http://doi.acm.org/10.1145/1137975.1137979
[10] S. Yasuda, K. Akashi, T. Miyachi, R. Beuran, Y. Makino, T. Inoue, S. Miwa, and Y. Shinoda, “Emulation-based ict system resiliency verification for disaster situations,” International Workshop on Resilient Internet based Systems (REIS 2013) in SITIS 2013, pp.875–882, Dec. 2013.
[11] R. Beuran, S. Yasuda, T. Inoue, S. Miwa, and Y. Shinoda, “Using emulation to validate post-disaster network recovery solutions,” 7th International ICST Conference on Simulation Tools and Techniques, pp.92–97, March 2014.
[12] L. Rizzo, “Dummynet: A simple approach to the evaluation of network protocols,” ACM SIGCOMM Computer Communication Review, vol.27, no.1, pp.31–41, Jan. 1997.
[13] H. Welte and P.N. Ayuso, “libnetfilter_queue project,” http://www.netfilter.org/projects/libnetfilter_queue/index.html, June 2008.
[14] A. Ihara, S. Murase, and K. Goto, “Ipv4/v6 network emulator using divert socket,” 18th International Conference on Systems Engineering (ICSE2006), pp.159–166, 2006.
[15] Y. Sugiyama and K. Goto, “Design and implementation of a network emulator using virtual network stack,” 7th International Symposium on Operations Research and Its Applications (ISORA ’08), pp.351–358, Nov. 2008.
[16] S. Hemminger, “Network emulation with netem,” Linux Conference Australia 2005, April 2005, http://lca2005.linux.org.au/abstract2e37.html?id=163
[17] M. Carson and D. Santay, “Nist net - a linux-based network emulation tool,” ACM SIGCOMM Computer Communications Review, vol.33, no.3, pp.111–126, July 2003.
[18] “Linux advanced routing & traffic control,” http://lartc.org/, 2002.
[19] J. Eriksson, F. Österlind, N. Finne, N. Tsiftes, A. Dunkels, and T. Voigt, “Cooja/mspsim: Interoperability testing for wireless sensor networks,” Proc. 2nd International Conference on Simulation Tools and Techniques, pp.27:1–27:7, 2009. http://dx.doi.org/10.4108/ICST.SIMUTOOLS2009.5637
[20] J. Malinen, “mac8011_hwsim,” http://wireless.kernel.org/en/users/Drivers/mac80211_hwsim, 2008.
[21] libnlProject, “netlink library,” http://people.suug.ch/~tgr/libnl/
[22] open80211s, http://open80211s.org/open80211s/

（平成 26 年 7 月 22 日受付，11 月 17 日再受付）

明石　邦夫　（正員）

2008 年大阪電気通信大学通信工学科修了．2010 年北陸先端科学技術大学院大学情報科学研究科博士前期課程修了．現在同大学院情報科学研究科博士後期課程に所属．ネットワークテストベッドの研究に従事．

井上　朋哉

2003 年石川工業高等専門学校専攻科電子機械工学専攻修了．同年株式会社 PFU 勤務．2006 年北陸先端科学技術大学院大学情報科学研究科博士前期課程修了．株式会社 Clwit 勤務を経て，2012 年同大学博士 (情報科学)．同年北陸先端科学技術大学院大学高信頼ネットワークイノベーションセンター研究員．2014 年より同特任助教．P2P ネットワーク，分散ネットワークの研究開発に従事．

Razvan Beuran

received the B.Sc., M.Sc. and Ph.D. degrees from “Politehnica” University, Bucharest, Romania in 1999, 2000 and 2004, respectively (the Ph.D. degree was delivered jointly with “Jean Monnet” University, Saint Etienne, France). From 2001 to 2005 he was with CERN, Geneva, Switzerland as a researcher. Since 2006 he is researcher, and since 2012 senior researcher with the National Institute of Information and Communications Technology, Hokuriku StarBED Technology Center, Ishikawa, Japan. Since 2007 he is also project researcher with the Japan Advanced Institute of Science and Technology, Ishikawa, Japan. His research topics include network dependability studies in wired and wireless networks, in particular through the use of network emulation. He is a member of IEEE.

篠田　陽一

1988 年東京工業大学理工学研究科博士課程修了．同年同大学工学部助手．1991 年北陸先端科学技術大学院大学情報科学研究科助教授．2001 年同大学情報科学センター教授．2006 年情報通信研究機構北陸リサーチセンタープロジェクトリーダー．2007 年内閣官房情報セキュリティセンター情報セキュリティ補佐官．情報環境，ネットワーク分散情報システム，ソフトウェア開発環境の研究に従事．工学博士．