ネットワークプロセッサを用いた 実験用ネットワークエミュレータシステムの構築

東條 晴基 † **長谷川 剛** ‡ **村田 正幸** ‡

† 大阪大学大学院基礎工学部
〒 560-8531 大阪府豊中市待兼山町 1-3
TEL: 06-6850-6616, FAX: 06-6850-6589
E-mail: kouda@ics.es.osaka-u.ac.jp

‡ 大阪大学サイバーメディアセンター
〒 560-0043 大阪府豊中市待兼山町 1-30
TEL: 06-6850-6616, FAX: 06-6850-6589
E-mail: {hasegawa, murata}@cmc.osaka-u.ac.jp

あらまし 本稿では、インテル株式会社製のネットワークプロセッサであるインテル IXP1200 を用いて実験用ネットワークエミュレータシステムを構築する。まず、インテル IXP1200 のハードウェア構成やその特徴について触れ、それらのハードウェアを用いて実現されているパケット入出力機構について述べる。次に、実験用ネットワークエミュレータシステムに期待される機能として、バッファリング機能、パケット遅延発生機能、パケットロス発生機能等を挙げ、各機能をインテル IXP1200 上に実装するための設計指針を述べる。また、実験環境設定を容易に行うための設定用シグナリングプロトコルの提案を行う。次に、提案するネットワークエミュレータシステムをインテル IXP1200 の動作のシミュレートが可能なマイクロエンジン開発環境である WorkBench 上に実装し、実装した機能の動作確認を行い、インテル IXP1200 実機に提案機能を実装した際に予想される性能に関する計測評価を行った。

キーワード ネットワークプロセッサ、インテル IXP1200、ネットワークエミュレータシステム、実験用ネットワーク、マイクロエンジン

Realizing Network Emulator System with Intel IXP1200 Network Processor

Haruki Tojo† **Go Hasegawa**‡ **Masayuki Murata**‡

Faculty of Engineering Science, Osaka University
1-3, Machikaneyama, Toyonaka, Osaka 560-8531, Japan
TEL: +81-6-6850-6616, FAX: +81-6-6850-6589
E-mail: kouda@ics.es.osaka-u.ac.jp

Cybermedia Center, Osaka University
1-30, Machikaneyama, Toyonaka, Osaka 560-0043, Japan
TEL: +81-6-6850-6616, FAX: +81-6-6850-6589
E-mail: {hasegawa, murata}@cmc.osaka-u.ac.jp

Abstract In this paper, we show the design and implementation issues of a network emulator system with Intel IXP1200 network processor. We first we introduce the architecture of Intel IXP1200 and its characteristics briefly, including the hardware I/O processing mechanisms for packet forwarding. We next describe required functions for the network emulator system such as buffering algorithms, link delay/loss emulation mechanisms and so on, and show implementation issues for those functions. We have implemented some of the functions on the software simulator of IXP1200, and confirmed that the implemented functions work correctly. We further evaluate the estimated performance of the implemented functions to discuss the processing overhead compared with the simple packet forwarding algorithm.

Keyword Network Processor, Intel IXP1200, Netowrk Emulator, Experimental Network, Microengine

1 はじめに

近年のインターネットユーザの増加に伴うネットワークトラヒックの急増に対し、輻輳を回避しネットワーク資源をより有効に利用するための研究が様々な方面から行われている。その結果、これまでに多岐に渡るアルゴリズム、アーキテクチャ等が提案され、評価、実装されている。それらの提案方式を評価し、その有効性及び現行方式に対する優位性を示すためには、数学的解析手法、およびコンピュータ上のシミュレーション技法のみでは不十分で、提案方式を実装し、実ネットワーク上における評価実験を行うことが必要不可欠である。しかし、提案方式をハードウェアを用いて実装するためには莫大な費用が必要である上に、ハードウェアの特性上、その機能拡張および改善に様々な制約が生じ、また再利用性にも欠けるため、現実的であるとはいえない。したがって、提案方式に対する性能評価の多くは、ソフトウェアを用いて提案方式の実装を行い、実験を行った結果によるものである。例えば、ルータバッファにおけるパケット処理方式の評価や、その影響に関する実験においては、ALTQ [1] 等の計算機上で動作するパケット処理システムが多く用いられている。

それらの評価実験においては、実運用されているネットワーク環境を用いることはネットワーク運用上困難であるため、小規模な実験用ネットワークを構築し、そこで提案方式の評価を行うのが一般的である。しかし近年、より大規模なネットワークを想定し、コンピュータ上のシミュレーションでは考慮することができない実装時のオーバヘッド等の事象を含めたより詳細な評価を行うために、ソフトウェア/ハードウェア処理によってネットワーク遅延、パケットロスなどの特定のネットワーク特性を実験ネットワーク内でエミュレートする、ネットワークエミュレータシステムが導入されつつある。

ネットワークエミュレータシステムは、計算機上で動作するシステム、独自のハードウェアによるシステム、及び本報告中で対象とする、主にパケット処理に特化した命令を持つネットワークプロセッサを利用するシステムの三つに分けることができる。計算機上で動作するシステムには、システムカーネルにモジュールを追加することで実現されている ALTQ [1] や NISTNet [2] 等がある。ALTQ はパケット処理機構として Class-Based Queueing (CBQ) [3]、Random Early Detection (RED) [4]、RED with In and Out (RIO) [5]、Hierarchical Fair Service Curve (HFSC) [6] 等を実現する機能を持つ。NISTNet は通過するパケットを確率的あるいは周期的に廃棄したり、伝搬遅延を発生させる機能を持つ。また、Linux カーネルモジュールとして実装されており、XWindow システムに対応し、環境設定を GUI を用いて行うことができる。これらの方式はソフトウェアで実現されているため、新機能の追加や改善を容易に行うことができるが、計算機をベースにしているために CPU の処理能力やネットワークカードの性能によって、パケット処理能力が制限されるという欠点を持つ。

独自のハードウェアによるシステムには、STORM [7]、MicroNET [8] 等がある。例えば STORM は、最大 10 までのマルチノード、マルチ WAN 環境をシミュレートすることが可能であり、パケット廃棄、伝搬遅延、リンクエラー等の発生機能、およびゲートウェイバッファのエミュレーション機能等を持つ。これらのシステムは独自のハードウェアを用いるため、高速かつ大規模な IP ネットワークをエミュレートすることができる反面、ある目的に特化された機能を実現するためにシステム全体が特殊な設計をしていることが多いため、機能拡張性に乏しいといえる。

一方、ネットワークプロセッサは、パケット処理に適したハードウェア構成、および命令セットを持つため、高速なパケット処理が可能なネットワークエミュレータシステムを構築することができる。さらに、パケット処理を行う箇所はプログラムで実現することができるため、機能拡張性に優れ、プログラム資源を再利用することが可能である。さらに、ソフトウェアネットワークシミュレータである ns [9] のようにプログラム資源を公開・共有することで、開発を促進することもできるため、安価かつ効率的に実験用ネットワークエミュレータシステムの構築が可能であると考えられる。

そこで本報告では、インテル株式会社 [10] 製のネットワークプロセッサであるインテル IXP1200 [11] を用い、実験用ネットワークエミュレータシステムの構築を行う。まず、本報告で対象としたインテル IXP1200 のハードウェアの全体構成を示し、次に各ハードウェアをマイクロエンジン、IX-Bus ユニット、メモリユニット等の部分に分けて各々のスペックや用途について説明を行い、単純なパケット転送を行うプログラムである Simplified Reference Design (SRD) の解析を通じて、インテル IXP1200 を用いたパケット処理システムの概要を示す。インテル IXP1200 によるパケット転送処理は大きく分けて、ネットワークから到着した入力ポートへのパケットのインテル IXP1200 内部への転送、転送されたパケットに対するあらかじめプログラムされたマイクロエンジンによる様々な処理、処理されたパケットの出力ポートへの移動、および出力ポートからネットワークへのパケット送出という 4 つの手順から構成される。

本報告では、このリファレンスデザイン SRD を基盤とし、ネットワークエミュレータシステムの構築を行う。まず、ネットワークエミュレータシステムに必要な機能としてバッファリング機能、パケットロス発生機能、パケット遅延 (伝搬遅延) 発生機能などを挙げ、それらの機

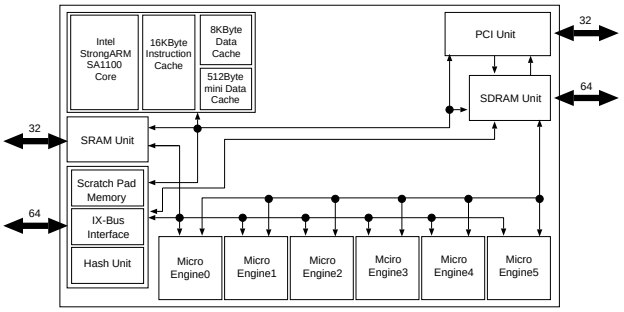

図 1: インテル IXP1200 の内部ハードウェア構成

能概略を示し、インテル IXP1200 上へ実装するための設計指針を示す。また、提案したエミュレータシステムの各機能を、インテル IXP1200 が持つマイクロエンジン上で動作するアセンブラコードによって実現し、その性能評価を行う。提案方式の評価は、マイクロエンジンコード開発環境である WorkBench [12, 13]を用いて行い、予測される処理オーバヘッド、パケット処理能力等の検証をおこなう。さらに、インテル IXP1200 の評価ボードであるインテル IXP1200EB (Evaluation Board) [14]上に提案するエミュレータシステムを実装し、データ転送実験を行うことにより、提案方式の有効性を検証する。

以下、2 章ではインテル IXP1200 のハードウェア各部の説明を行い、パケット処理の基本的な過程について述べる。3 章ではネットワークエミュレータシステムを構築するために、インテル IXP1200 上で実現すべき機能を挙げ、その設計指針について述べる。4 章ではそれらのうちいくつかの機能を実装し、シミュレータ及び実機を用いた性能評価結果を示す。最後に 5 章で本報告のまとめと今後の課題について述べる。

2 ネットワークプロセッサ インテル IXP1200

本報告で用いるインテル IXP1200 はパケット処理に適したネットワークプロセッサである。本章ではインテル IXP1200 を構成するハードウェア、および各ハードウェアを構成するユニットの特徴や用途の紹介を行う。また、インテル IXP1200 を用いて実現されている単純なパケット転送処理機構を例に挙げ、各ハードウェアの動作に関する詳細な説明を行う。

2.1 ハードウェア構成

インテル IXP1200 はパケット処理に関するプログラミングを行うことができるマイクロエンジンを 6 個、および主にマイクロエンジンの制御用途に用いられる StrongARM コアを 1 個の、合計 7 個の RISC プロセッサを持

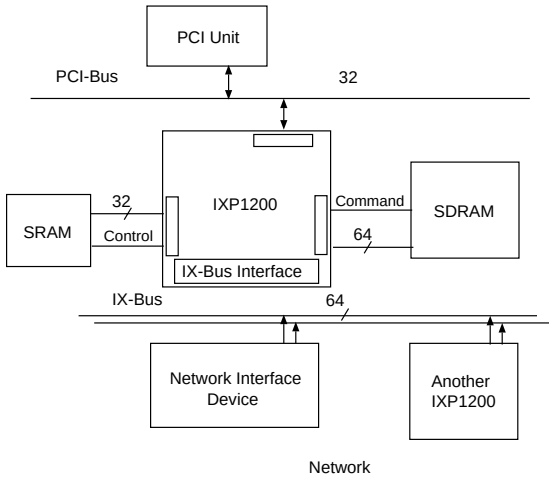

図 2: インテル IXP1200 の外部ハードウェア構成

ち、メモリ関連のハードウェアとして、SRAM ユニット、および SDRAM ユニットを持つ。さらに、パケット等のデータ転送用内部バスとして PCI ユニットや IX-Bus ユニットと呼ばれる独自のバス機構を有する。図 1 はインテル IXP1200 内における各ユニットの配置図を、図 2 はインテル IXP1200 と周辺ハードウェアの配線図をそれぞれ示す。以降では、各部についての詳細な説明を行う。

2.1.1 マイクロエンジン

マイクロエンジンはインテル IXP1200 内でパケット処理を行うために用いられる。インテル IXP1200 は 6 つのマイクロエンジンを持ち、それらは互いに独立して動作する。また、1 つのマイクロエンジンは 4 つのスレッドを動作させることができるため、全体で 24 個のスレッドがパケット処理を行うことができる。しかし、各マイクロエンジン内の 4 つのスレッドは並列に動作することはできないため、スレッドは適宜切り替えられて動作する。したがって、全体では 6 つのスレッドが並列処理を行うことが可能である。この機構はマルチスレッド機構と呼ばれ、マイクロエンジン内のスレッドの切り替えは各スレッドが用いているプログラム・カウンタと相対レジスタセットを入れ替えることにより行われる。スレッド切り替えは、主にメモリ参照のように実行に長い時間を必要とする命令を行う間の待ち時間を、他のスレッドが有効に利用するために行われる。またスワップ命令を用いて、スレッドの切り替えを意図的に行うことも可能である。

また、マイクロエンジンは 1 命令あたりの実行速度を向上させることを目的とした、パイプライン制御処理機構を持つ。これにより、外部参照やメモリ参照等の一部を除く全ての命令を 1 クロックサイクルで実行することができる。またメモリ参照等、他のハードウェアとデー

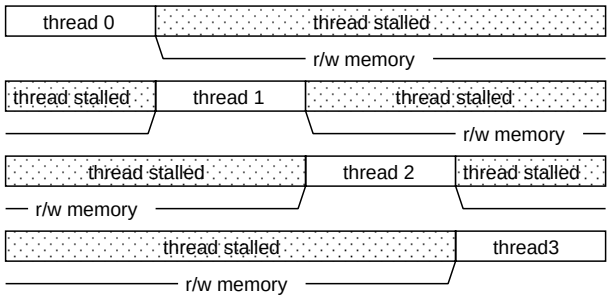

図 3: マイクロエンジンのマルチスレッド機構

タ転送を行う際に利用する転送レジスタ、および汎用的に用いる汎用レジスタを 1 スレッドあたり 32 個利用することができる。

マイクロエンジンは RISC プロセッサであり、プログラムによってその動作を記述することが可能である。また、マイクロエンジンは以下の 5 種類の命令群を持つ。

データ処理命令 算術演算、論理演算、移動命令、シフト命令等

分岐命令群 ジャンプ命令、スレッド分岐命令、サブルーチンコール等

メモリ参照命令群 Scratch Pad メモリ、SRAM、SDRAM を参照するための命令群

外部参照命令群 R-FIFO、T-FIFO を参照および利用するための命令群

CSR (Control Status Register) アクセス命令群 制御レジスタを利用するための命令群

また、移動命令およびメモリ参照命令は、パケット処理を容易にするためにバイト (8 ビット) 単位、ワード (16 ビット) 単位、ロングワード (32 ビット) 単位でのデータ処理が可能となるように設計されている。

2.1.2 StrongARM

StrongARM はインテルが開発した小さい消費電力で、高い処理能力を持つマイクロプロセッサの総称で、主にデジタルビデオカメラ等の携帯製品に利用されている。インテル IXP1200 で用いられている StrongARM は SA (StrongARM) ファミリー [15] の SA1100 RISC コアであり、動作周波数は 200 MHz である。また、16 KByte の命令キャッシュ、8 KByte のデータキャッシュ、512 Byte のミニキャッシュを持つ。

インテル IXP1200 上の StrongARM は、主にマイクロエンジンの制御に用いられるが、マイクロエンジンとの間のデータ転送機能を利用することにより、三角関数や指数関数の演算等というマイクロエンジンで行えない複雑な処理を StrongARM 上で実行することができるため、マイクロエンジンの負荷を減少させ、インテル IXP1200 全体の処理能力を向上させることができると考えられる。

2.1.3 IX-Bus ユニット

IX-Bus ユニットは IX-Bus、T-FIFO、R-FIFO、IX-Rdy-Bus、ハッシュユニット、IXP1200 CSRs (Control Status Registers: 制御レジスタ群)、および Scratch Pad メモリから構成され、ネットワークインタフェースとインテル IXP1200 内部間のインタフェースを提供している。

ネットワークインタフェースとインテル IXP1200 内部間のパケットの移動に用いられる IX-Bus は、85MHz の周波数で動作し、単方向 32 ビット、双方向 64 ビットのデータインタフェースを持ち、最大 5.44 Gbps でのデータ転送が可能である。また、IX-Bus を介して複数のインテル IXP1200 を接続することが可能で、それによりさらに高速なパケット処理を行うことができる [16]。

T-FIFO、R-FIFO はネットワークインタフェースとインテル IXP1200 内部間のパケット転送のためのバッファとして用いられる。ネットワークインタフェースを通してネットワークから受信したパケットは R-FIFO を経由してインテル IXP1200 内部へ移動し、処理を受けた後、T-FIFO、ネットワークインタフェースを経由して、ネットワークへ出力される。IX-Rdy-Bus は T-FIFO、R-FIFO 内のパケットの蓄積状態を監視するために用いられ、IX-Rdy-Bus が持つレジスタ内のビットによってパケットの受信/送信をマイクロエンジン等に通知することができる。また、R-FIFO、T-FIFO からのデータの移動許可に関する信号の送受信は、受信/送信ステートマシンと呼ばれるパケット処理機構によって行われる。

ハッシュユニットは 48/64 ビットのハッシュ操作を固定時間で行うことができる。ハッシュ操作は CSRs および SRAM 転送レジスタを用いてハッシュ・キーを生成し、別の 2 つの SRAM 転送レジスタからハッシュ処理を行う値を受け取り、ハッシュ処理後、データを取りだした 2 つの SRAM 転送レジスタに結果を格納する。また、1 個のハッシュ・キーに対して最大 6 個の SRAM 転送レジスタを用いることで、3 組の SRAM 転送レジスタに対して同時にハッシュ操作を行うことが可能である。

IXP1200 CSRs (Control Status Registers) にはマイクロエンジンから受信ステートマシンへパケットの受信要求を伝える際に用いる RCV_REQ レジスタや受信要求を受信した後に受信制御情報が書き込まれる RCV_CTL レジスタ等のマイクロエンジンおよび IX-Bus ユニットにおけるパケット送受信処理の制御を行うためのレジスタや、SRAM 転送レジスタにサイクル数を書き込む際に用いられる CYCLE_CNT 等のレジスタが含まれる。これらのレジスタは CSR アクセス命令を用いることで利用することができる。

Scratch Pad メモリは IX-Bus ユニット内に存在し、インテル IXP1200 が持つ唯一のメモリである。4 KBytes の容量を持ち、1024 個の 32 ビットエントリを持つ。その

ため Scratch Pad メモリへのアクセスは 32 ビット単位で行われる。また、Scratch Pad メモリは IX-Bus ユニット内に存在するため、アクセス時間は約 20 クロックサイクル (本報告で用いるインテル IXP1200 は 200MHz で動作するため、1 クロックサイクル 5nsec である) であり、外部メモリである SRAM (30 クロックサイクル) および SDRAM (50 クロックサイクル) に比べて高速なアクセスが可能である。さらにデータ転送を最大 3.2 Gbps で行うことができるため、主に制御信号のやりとりやマイクロエンジンが用いるプログラム変数の格納場所として利用される。

Scratch Pad メモリに対して実行可能な操作は、二つのオペランドを用いて指定する 32 ビット幅のエントリの内容を SRAM 転送レジスタ (マイクロエンジンが持つ SRAM データ転送用の 32 ビットレジスタ) へ読み出す READ 命令、SRAM 転送レジスタに格納されている値を指定するエントリに書き込む WRITE 命令、前回指定したエントリの値を 1 増加させる INCR 命令、指定するエントリに対して SRAM 転送レジスタに格納されている値を用いてビットマスク操作を行う BIT_WR 命令の 4 操作である。

2.1.4 メモリユニット

インテル IXP1200 は内部メモリである Scatch Pad メモリの他に、外部メモリである SRAM、および SDRAM を用いるためのインタフェースとなる SRAM ユニット、および SDRAM ユニットを持つ。

SRAM ユニットは 8 MByte までの SRAM 領域をサポートすることができる。32 ビットのデータインタフェースを持ち、CPU 動作周波数の 1/2 の周波数で動作するため、3.2 Gbps でのアクセスが可能である。SRAM ユニットは、インテル IXP1200 の中で唯一排他制御機構である CAM (Common Access Method) を持ち、あるスレッドが参照中のメモリ領域に他のスレッドがアクセスすることを禁止することができる。SRAM ユニットはこの CAM を実現させるためのレジスタを 8 個持つ。また、SRAM 領域に線形リスト状のスタックを生成するための Push/Pop レジスタをそれぞれ 8 エントリ持つ。Push/Pop レジスタは、SRAM 領域上に生成した各々のリストの先頭要素が格納されているメモリアドレス値を格納している。

SRAM ユニットを用いて SRAM に対して実行することができる操作は、Scratch Pad メモリに対する 4 つの操作 (READ、WRITE、INCR、BIT_WR 命令) に加え、指定したアドレスをロックしてから内容を SRAM 転送アドレスに読み出す READ_LOCK 命令、ロックされたアドレスに SRAM 転送レジスタの値を書き込み、そのアドレスを解放する WRITE_UNLOCK 命令、READ_LOCK 命令でロックされたアドレスを解放する UNLOCK 命令、Push/Pop レジスタが生成したスタックに SRAM 転送レジスタの内容を追加する PUSH 命令、およびスタックトップからデータを読み出し、その内容を SRAM 転送レジスタに格納する POP 命令の計 10 操作である。これらの操作により、SRAM に対して様々なメモリアクセスを行うことができるため、SRAM はインテル IXP1200 の出力ポートに対して用意される出力ポート・キューの先頭を指すポインタや、現在のキューサイズなど、様々なデータの格納場所として用いられる。

また、SRAM ユニットは 4 つのマイクロエンジン・コマンド・キューと呼ばれるキューを持つ。これは、マイクロエンジンからのメモリアクセス命令をその内容に応じてキューに格納し、READ/WRITE 命令を連続して実行できるようにするなどの、メモリアクセス効率の向上のためのスケジューリングを行う。

SDRAM ユニットは 256 MBytes までの SDRAM 領域をサポートすることができる。64 ビットのデータインタフェースを持ち、CPU の 1/2 の周波数で動作するため、6.4 Gbps でのアクセスが可能である。SDRAM ユニットはデータ転送用のデータバス、コマンド転送用のコマンドバスを各々一本ずつ持つ。コマンドバスは SDRAM に対してデータ転送を行う前にデータの列アドレスを指定する Row Access Strobe (RAS)、データの行アドレスを指定する Column Access Strobe (CAS)、読み出しによるデータ損失を防止するための Pre-Charge (PRE C) コマンドを実行するために用いられる。また、SDRAM ユニットは Direct Memory Access (DMA) のように、転送用のレジスタを用いずに IX-Bus ユニットとの間で直接データ転送を行うことができる。また、SRAM ユニットと同様、メモリアクセス効率を向上させるための 4 つのマイクロエンジン・コマンド・キューを持つ。

SDRAM は通常、入出力されるパケットを格納するために用いられ、二つのバンク (領域) に分けられる。この 2 つのバンクおよび SDRAM ユニットが持つ独立した 2 本のバス (データバスとコマンドバス) を用いて、あるバンクに対してメモリアクセスを行う間に、他方のバンクに対して別のメモリアクセスコマンドを実行をすることが可能になる。したがって、メモリアクセスを行うバンクを交互に切り替えることによってメモリアクセスコマンドの実行時間を短縮し、効率のいいメモリアクセスを行うことができる。

2.1.5 その他

インテル IXP1200 の評価ボードである IXP1200EB は、ネットワークインタフェースとして、全二重通信が可能である 8 個の 10/100 base-T のイーサネットポート、およ

び 2 個の 1000 base-SX のイーサネットポートを持つ。また、シリアルポートと PCI ユニットを持ち、他の計算機からシリアルポート経由でインテル IXP1200 にアクセスし、データ転送を行うことができる。PCI ユニットは 32 ビットのデータインタフェース、33 MHz または 66 MHz の周波数で動作する。また、OS と拡張カード間の入出力方法を標準化した規格である I_2O (Intelligent I/O) をサポートしており、本来計算機側の CPU が行う入出力制御を I/O コントローラで実行することが可能になる。それにより、CPU の負担を軽減することができる。

2.2 パケット転送処理機構

本節ではインテル株式会社が作成している単純なパケット転送を行うリファレンスデザイン SRD (Simplified Reference Design) を基に、インテル IXP1200 がそのハードウェアを用いてパケットの送受信処理を行う手順についての説明を行う。 2.1.1 節で述べたようにインテル IXP1200 は 6 つのマイクロエンジンを持ち、それぞれが独立して動作する。リファレンスデザイン SRD ではその特徴を利用し、各マイクロエンジンはパケット受信あるいは送信のいずれかの処理のみを行うようにプログラムされている。また、受信処理用マイクロエンジンと送信処理用マイクロエンジンの個数は 2:1 の比率とすることで、インテル IXP1200 のハードウェアが持つ能力を最も有効に利用することができる [17]。以下では、パケット転送処理を図 4 に示すように、スイッチング処理、およびデイジーチェイン処理に分割し、それぞれについて説明する。

2.2.1 入出力

外部ネットワークから到着したインテル IXP1200 へのパケットは、以下に示す手順にしたがって処理され、外部ネットワークへ送出される。

ネットワークからの到着パケット処理

ネットワークインタフェースデバイスは外部ネットワークとインテル IXP1200 とを接続するためのデバイスで、パケットはそのデバイスを経由してインテル IXP1200 に到着する。到着パケットは IX-Bus を経由し、R-FIFO へ格納される。R-FIFO 内のパケットの管理は IX-Bus 内にある受信ステートマシンが行う。

パケットのインテル IXP1200 内への転送

マイクロエンジンは IX-Rdy-Bus を監視し、受信フラグが立っている、すなわち R-FIFO にパケットが格納されていれば、受信ステートマシンにパケット転送要求を送る。受信ステートマシンはマイクロエンジンからパケット転送要求を受理すると、マイクロエンジンに受理確認信号を送信する。マイクロエンジンは受理確認信号を受信すると、R-FIFO 内のパケットを SDRAM に転送する。この後、2.2.2 節で説明するスイッチング処理が行われる。

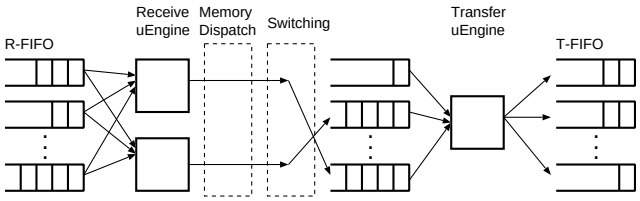

図 4: SRD におけるパケット処理手順

パケットの外部ネットワークへの送出

スイッチング処理が終わると、マイクロエンジンはパケットを送信するために T-FIFO へ移動させる。そのために、マイクロエンジンは IX-Rdy-Bus を監視し、送信フラグの状態からパケットを T-FIFO へ送信することができるか否かを判断する。送信可能であれば、パケットを SDRAM から T-FIFO へ移動し、送信ステートマシンにパケットの移動を知らせる信号を送る。送信ステートマシンはマイクロエンジンからパケット移動の信号を受けると、T-FIFO 内に格納されているパケットを IX-Bus に移動し、パケットを外部ネットワークへ送出する。

2.2.2 スイッチング

インテル IXP1200 に新しくパケットが到着すると、そのパケットを格納する SDRAM 領域へのポインタを SRAM 上に作成し、以降のパケット参照のために用いる。パケットを SDRAM に転送した後、パケットヘッダ情報から各パケットの目的アドレスを参照し、出力ポートを決定する。その後、パケットへのポインタを SRAM 上に用意されている各出力ポート毎の出力ポート・キューの末尾に追加 (ディスパッチ) することでスイッチング処理が完了する。リファレンスデザインでは、各キューは 8192 個のパケットを格納することができ、キューの末尾が先頭に接続されているサイクリック構造を持つ。格納されたパケットは図 5 に示すように SDRAM 上に不連続に格納されているが、SRAM 上の出力ポート・キューは SDRAM 上に格納されたパケットへのポインタのリストであるため、出力ポート・キューへのパケットの到着順通りにパケット転送処理を行うことができる。

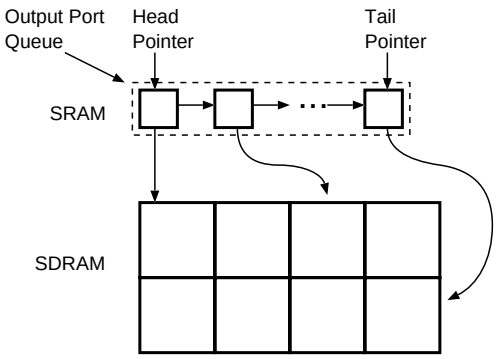

図 5: 出力ポート・キュー

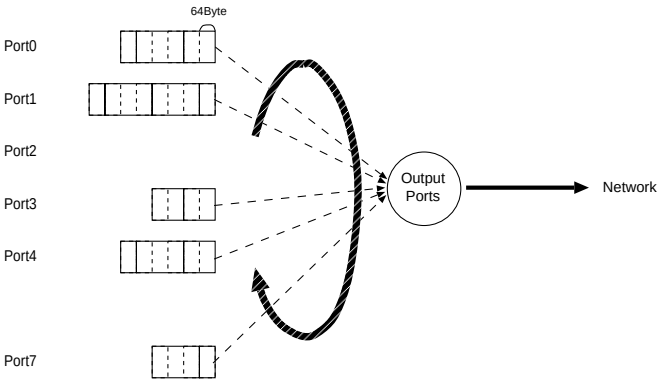

図 6: デイジーチェイン機構

2.2.3 デイジーチェイン

リファレンスデザイン SRD が用いているパケット送信プログラムは、図 6 に示すデイジーチェインと呼ばれるアルゴリズムを用いて、各出力ポート・キューに格納されたパケットを公平に外部ネットワークへ送信する。デイジーチェインは出力ポート・キューを順に参照し、キュー内に出力できるパケットが存在する場合は 64 Byte 分のビット送信を行い、次の出力ポート・キューの参照を行う。またキュー内に出力できるパケットが存在しない場合は処理を行わずに次の出力ポート・キューの参照を行う。すなわちデイジーチェインは、各出力ポート・キューのパケットを 64 Byte 単位で処理を行うラウンドロビンアルゴリズムに相当する。

3 ネットワークエミュレータシステム

本章では、本報告において構築するネットワークエミュレータシステムの特徴、およびエミュレーションを行うネットワーク機能について述べ、各機能をインテル IXP1200 上に実装する際の設計指針を示す。また、実装した機能に関しては設計指針に加え、詳細なアルゴリズムの説明を行う。

3.1 特徴

図??に、提案するネットワークエミュレータシステムの全体図を示す。システムはネットワークインタフェースとしての 8 個の 10/100 base-T のポート、バッファリング機能、伝搬遅延、パケットロス、ビットエラー等の発生機能を持つ。システムを利用するユーザは、データの送受信を行うホストをインテル IXP1200 に接続し、インテル IXP1200 がエミュレートするネットワークとして各ポートの帯域や伝搬遅延時間、出力バッファにおけるパケット処理機構等の様々な設定、およびパラメータの決定を行い、実験用ネットワークを構築する。次節以降で、本報告で提案するエミュレータシステムに必要と考えられるネットワーク機能を紹介し、その設計指針について述べる。

3.2 機能

3.2.1 ルータバッファにおけるパケット処理機構

現在多くのルータバッファで採用されているパケット処理機構は、単純な TD (Tail Drop) 機構である。これはルータバッファにおけるパケット処理機構として最も基本的な機構であるため、エミュレータシステムに必要不可欠であると考えられる。TD 機構は、パケットをその到着順にバッファに格納し、FIFO (First In First Out) 規律に従って到着順にパケット処理を行い、出力する。バッファが一杯になれば、その後到着するパケットは廃棄される。

また、近年実装が進みつつあるパケット処理機構として RED (Random Early Detection) 機構 [4] が挙げられる。これは、バッファが一杯になる前に確率的なパケット廃棄を行うことで、ルータのスループットを保ちながらキュー長 (バッファ内パケット数) を小さく抑える機構である。また、TD 機構で発生するバースト (連続) 的なパケット廃棄を防止することができる。RED 機構はパケットが到着する毎に、ローパスフィルタの一種である重み付き指数平均を用い、現在のキュー長 (バッファ内パケット数) q から平均キュー長 $\overline{q}$ を以下の式にしたがって計算する。

$$\overline{q} \longleftarrow (1 - w_q) \cdot \overline{q} + w_q \cdot q \tag{1}$$

ここで、w_q は重みを表す RED 機構の制御パラメータである。この平均キュー長 $\overline{q}$ を 2 つのしきい値 min_{th} および max_{th} と比較することにより、以下のように到着したパケットに対する処理を行う。

1. $\overline{q} < min_{th}$ ならば到着したパケットをバッファへ格納する。
2. $min_{th} \leq \overline{q} < max_{th}$ ならば以下の式を用いて決定

される確率 p_a で到着したパケットを廃棄する。

$$p_b \longleftarrow \frac{\overline{q} - min_{th}}{max_{th} - min_{th}} \cdot max_p \tag{2}$$

$$p_a \longleftarrow \frac{p_b}{1 - count \cdot p_b} \tag{3}$$

ここで、max_p はパケット廃棄確率の大きさを決定する RED 機構の制御パラメータである。

3. $max_{th} \leq \overline{q}$ ならば到着したパケットを廃棄する。

また、その他実現の必要があると考えられるパケット処理機構として、RED 機構の拡張方式であり、キュー長の安定性を向上させる SRED (Stabilized RED) 機構 [18]、フロー (コネクション) 間の公平性を向上させる FRED (Flow RED) 機構 [19]、DiffServ[20] 用いられる RIO (RED In and Out) 機構等が挙げられる。

3.2.2 リンク遅延、パケット廃棄機能

ネットワーク内のリンクをエミュレーションするために必要な機能として、以下に示す機能が必要であると考えられる。

パケット伝搬遅延発生機能 リンクの物理的距離をエミュレートする

パケットロス発生機能 輻輳等によるネットワーク内におけるパケット廃棄をエミュレートする

ビットエラー発生機能 無線回線、衛星回線に多く見られるリンクのノイズをエミュレートする

また、これらの機能の実現の際には、一様分布にしたがう確率分布のみではなく、様々な確率分布を用いることが考えられる。したがって、確率分布を指定する機能が必要であると考えられる。

3.2.3 リンク帯域設定機能、バックグランドトラヒック設定機能

インテル IXP1200EB にはネットワークインタフェースとして 10/100base-T および 1000base-SX のみを持ち、小さい帯域のインタフェースを持たない。そのためアクセスリンク等を想定した、帯域の小さなリンクを実験用ネットワーク内に実現するためには、帯域を設定する機能が必要である。また、バックグランドトラヒックを発生させる機能を用意することで、バックグランドトラヒックを発生させ、実験ネットワークに負荷をかけるための外部ホストを用意する必要がなくなる。

3.2.4 パケットフィルタリング

TCP/IP レベルの動作をエミュレートする機能として、特定の IP アドレスからのパケットまたは、特定の IP アドレスに向けてのパケットに対して、パケット廃棄または優先的なパケット転送等の処理を行うパケットフィルタリングおよびルーティング等の機能が必要であると考えられる。これにより、DiffServ ネットワーク等のエミュレートや、TCP トラヒックと UDP トラヒックが混在する環境を想定した実験用ネットワークの構築等が可能となる。

3.2.5 設定用シグナリングプロトコル

提案システムにおいて実現するエミュレーション機能の設定は、通常インテル IXP1200 内部のプログラムやパラメータの変更によって行うことができるが、ネットワークエミュレータシステムを用いるユーザが容易に設定を行うために、外部ホストからの設定が可能となるシグナリングプロトコルが必要であると考えられる。これにより、ユーザはインテル IXP1200 内部の構成やプログラムの詳細を知ることなく、用意されたネットワークエミュレート機能を用いて実験用ネットワークを構築することが可能になる。

3.3 設計指針

本節では 3.3 節で述べた機能のうち、本報告において構築したネットワークエミュレータシステムに実装した機能を挙げ、その設計指針を示す。

3.3.1 ルータバッファにおける TD 機構

TD 機構はある一定サイズのバッファを持ち、ルータバッファが一杯になるまで到着したパケットがキューに格納される。したがって、バッファサイズ (格納可能なパケット数の上限) を設定し、パケットが到着する度に現在のキュー長 (バッファ内パケット数) とバッファサイズを比較し、キュー長がバッファサイズより大きい場合には到着したパケットを廃棄する。以下にマイクロエンジンにおいて実行される受信側処理および送信側処理の手順を示す。

- 受信側処理
  1. 各出力ポート i 毎にバッファサイズ B_i を設定し、Scratch Pad メモリに変数として保持する。
  2. 出力ポート i のキュー長 q_i を格納する変数を Scratch Pad メモリに作成し、0 に初期化する。
  3. R-FIFO にパケットが到着するまで待機し、到着すればパケットを R-FIFO から SDRAM 領

域へ転送する。

4. SDRAM 領域への転送が完了すれば、到着したパケットが出力されるポート j を調べ、該当するポートのキュー長 q_j とバッファサイズ B_j を取得し、比較する。
5. $q_j < B_j$ の場合、出力ポート・キューに到着したパケットを格納し、$q_j \longleftarrow q_j + 1$ とする。
6. $q_j \geq B_j$ の場合、バッファ溢れが発生したと判断し、到着したパケットを廃棄する。パケット廃棄は、パケットが格納されている SDRAM 領域を開放し、出力ポート・キューからポインタを廃棄することで実現する。

- 送信側処理
  1. パケット転送の終了後、出力ポート i のキュー長 q_i を、$q_i \longleftarrow q_i - 1$ とする。

3.3.2 ルータバッファにおける RED 機構

RED 機構が用いる平均キュー長、およびパケット廃棄確率の導出は、節で示した式にしたがって行う。その際、制御パラメータである w_q、max_p、および変数である $\overline{q}$、p_a、p_b 等は小数値を取るが、インテル IXP1200 は小数値を取り扱うことができないため、これらの変数は擬似的に固定小数点による実数として表現する。インテル IXP1200 のマイクロエンジンが用いるレジスタは 32 ビットであるため、ここでは小数点を 15、16 ビットの間に固定し、上位 16 ビットを整数部分、下位 16 ビットを小数部分として扱う。これにより、小数は 0.0000152587890625 から 0.9999847412109375までの数値を表現することができ、整数は 0 から 65535 までの数値を表現することができる。また、平均キュー長、およびパケット廃棄確率の導出の際には、乗除算演算を行う必要があるが、任意の数に対する乗除算演算は処理時間が非常に大きくなることが考えられる。そこで、乗除数または被乗除数の一方を 2 のべき乗に設定、あるいは近似することで、ビットシフト操作のみで演算を行う。

式 (1)–(3) は以下のように変形することができる。

$$\overline{q} \longleftarrow \overline{q} + (q - \overline{q}) \cdot w_q \tag{4}$$

$$p_b \longleftarrow (\overline{q} - min_{th}) \cdot \frac{max_p}{max_{th} - min_{th}} \tag{5}$$

$$p_a \longleftarrow \frac{p_b}{(1 - count \cdot p_b)} \tag{6}$$

したがって、w_q、max_p、および $(max_{th} - min_{th})$ を 2 のべき乗に設定し、$(1 - count \cdot pb)$ を 2 のべき乗に近似することで、ビットシフト操作のみで乗除算を行うことが可能となる。以下にマイクロエンジンにおいて実行される受信側処理/送信側処理の手順を示す。

- 受信側処理
  1. 出力ポート i 毎に RED 機構のパラメータ ($max_{th,i}$、$min_{th,i}$、$max_{p,i}$、$w_{q,i}$) を設定し、Scratch Pad メモリ領域に変数として保持する。
  2. 出力ポート i 毎に平均キュー長 $\overline{q}_i$、通過パケット数 $count_i$、キュー長 q_i を格納する変数を Scratch Pad メモリに作成し、0 に初期化する。
  3. R-FIFO にパケットが到着するまで待機し、到着すればパケットを R-FIFO から SDRAM 領域へ転送する。
  4. SDRAM 領域への転送が完了すれば、到着したパケットが出力されるポート j を調べ、出力ポート j の平均キュー長 $\overline{q}_j$ を式 (4) を用いて導出する。
  5. $\overline{q}_j < min_{th,j}$ の場合、到着したパケットを出力ポート・キューに格納し、$q_j \longleftarrow q_j + 1$ とする。
  6. $min_{th,j} \leq \overline{q}_j < max_{th,j}$ の場合、パケット廃棄率を式 (5)、(6) を用いて導出し、乱数を用いて到着したパケットの廃棄判定を行う。廃却しない場合には $q_j \longleftarrow q_j + 1$ とする。
  7. $\overline{q}_j \geq max_{th,j}$ の場合、到着したパケットを全て廃棄する。
- 送信側処理
  1. パケット転送の終了後、出力ポート i のキュー長 q_i を、$q_i \longleftarrow q_i - 1$ とする。

3.3.3 確率分布指定機能

本報告で構築するネットワークエミュレータシステムにおいては、RED 機構およびパケットロス発生機能が用いるパケット廃棄確率、パケット伝搬遅延発生機能が用いる遅延時間等の設定の際に、固定値や一様分布にもとづく確率だけではなく、以下に示すさまざまな確率分布を用いることができる。以下に、利用できる確率分布とその実現方法について述べる。

- 一様分布
  CSR 命令を用いて得られるマイクロエンジンのクロックサイクル数 C_t を乱数の種として用い、下記の線形合同法の式 [21] により擬似乱数列 r_n の生成を行う。
  $$r_n \longleftarrow A * C_t \pmod P$$
  ここで、A は $A > 0$ である定数、P は乱数の周期を示す。本報告では $A = 7$、$P = 1024$ を用いており、0 から 1023 までの乱数を発生している。
- ポアソン分布
  一様分布を用いて生成される擬似乱数列を用いて、ポアソン定数を指定し、以下に示すポアソン分布乱数列の生成式の計算を行うことにより、ポアソン分

布によるパケット廃棄を実現する。λ をポアソン定数、u_i を一様分布の乱数列とすると式 (7) を満たす最小の k がポアソン乱数列 p_n の要素となる [22]。また、λ、$exp(-\lambda)$ はあらかじめ計算し、Scratch Pad メモリに格納しておくことで、マイクロエンジンにおける計算量を減少させる。

$$\prod_{i=1}^{k} u_i \quad < \quad \exp(-\lambda) \tag{7}$$

$$p_n \quad \longleftarrow \quad k \tag{8}$$

- 指数分布
  一様分布を用いて生成される擬似乱数列を用い、ポアソン定数を指定し、指数分布乱数列の生成を式 (9) を用いて行う [23]。ここで u_i は一様分布の乱数列であり、超越関数である ln の計算は ln をテイラー展開し、近似計算を行う。

$$e_n \quad \longleftarrow \quad \frac{-\ln(1-u_i)}{\lambda} \tag{9}$$

- 一般分布
  本報告において提案するネットワークエミュレータシステムにおいては、上記の 3 種類の確率分布以外にも一般的な確率分布を用いることができる。確率分布の指定方法は 3.3.4 節に示す。

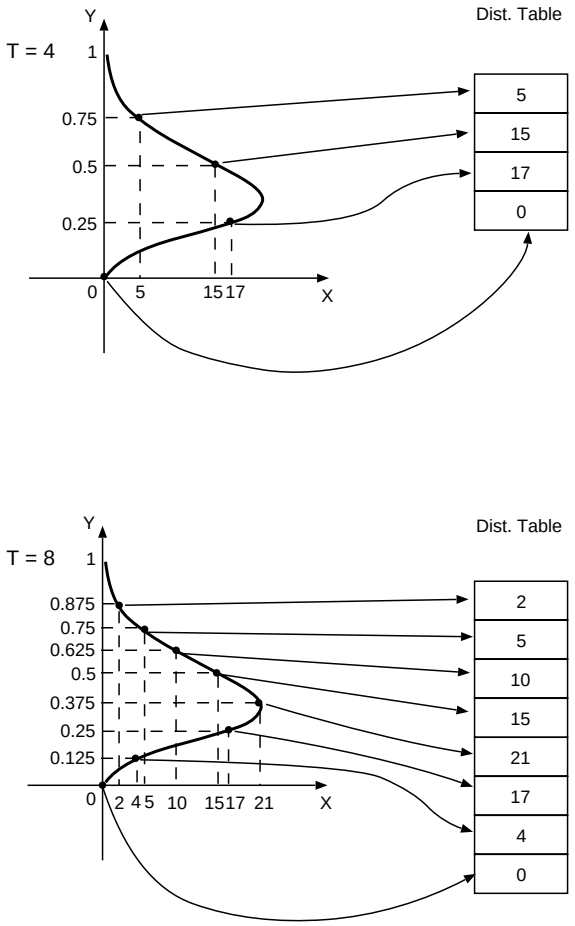

図 7: 確率分布テーブル

3.3.4 一般分布指定

実ネットワークにより近い環境をエミュレートするためには、パケットロス発生機能の廃棄確率、パケット伝搬遅延発生機能の遅延時間等に、前節で示した一様分布、ポアソン分布、指数分布だけではなく、対数正規分布、パレート分布といったの任意の確率分布にしたがう値を用いることができる必要がある。しかし、それらの複雑な確率分布を与える式には cos、sin、exp、ln 等の超越関数が含まれているため、それらの関数演算をマイクロエンジンにおいて行うためには莫大な処理工数が必要となる。したがって、これらの確率分布にしたがう値をマイクロエンジンにおいて動的に発生させることは現実的ではない。近似手法を用いて処理工数を減少させることはある程度可能であるが、近似によって得られる分布は粒度が荒い離散的な分布になるため実用的ではない。この問題に対しては、以下の 2 つの解決策が考えられる。

- 複雑な確率分布の計算は StrongARM が行い、マイクロエンジンは適宜 StrongARM と通信を行い、確率分布にしたがう値を取得する。
- あらかじめ SRAM 領域に確率分布を示すテーブルを書き込んでおき、それを利用することで確率分布にしたがう値を取得する。

本報告で提案するネットワークエミュレータシステムにおいては、後者の方式を用いている。すなわち、あらかじめ計算した確率分布値のテーブル (確率分布テーブル T) を作成し、SRAM 領域に保持する。マイクロエンジンは確率分布テーブル内のインデックス値を一様分布乱数を用いて求め、インデックスで指定された確率分布テーブル内の分布値を利用することにより、パケット廃棄確率、遅延時間等の設定に利用する。テーブルに格納される確率分布値は、図 7 に示すように、確率分布関数のグラフを Y 軸方向に等間隔に分割して得られる座標値である。また、複数の確率分布テーブル i を保持し、ポートごとに用いる分布を選択することができる。その場合、確率分布テーブルは通し番号をつけて管理する。

確率分布テーブルは複数作成することが可能であり、作成したテーブルは SRAM 領域に図 8 のように格納される。まず SRAM 領域上に各テーブルへのポインタリストを用意する。ポインタは各テーブルが格納される先頭アドレスを指し、ポインタリストのインデックスがテーブル番号として各テーブルの指定に用いられる。複数のテーブルをこの方式を用いて SRAM 上に格納することにより他の機能はテーブル番号とテーブル内のインデックスの二つの値を元にテーブルから確率分布にしたがう

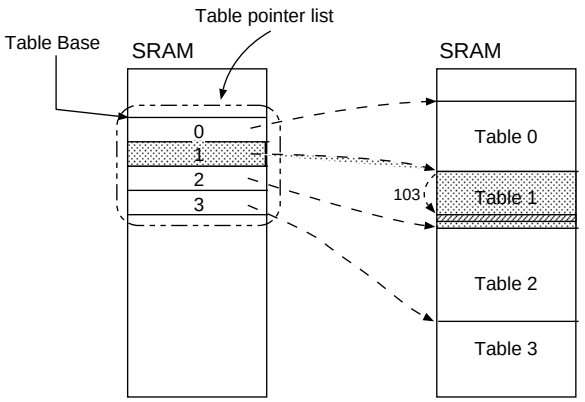

図 8: 確率分布のテーブルのメモリマップ

値を取得できる。例えば、テーブル番号が 1、テーブル内のインデックスが 103 である場合は図中の斜線部を参照する。

3.3.5 パケットロス発生機能

提案するネットワークエミュレータシステムにおいては、3.3.3 節および 3.3.4 節において示したさまざまな確率分布関数を用いて、システムに到着するパケットを廃棄することができる。以下に、確率分布を用いる場合と、固定確率を用いる場合についてそれぞれマイクロエンジンにおいて実行される受信側処理の手順を示す。送信側に関しては、特別な処理は行わない。

確率分布を用いる場合

1. 各出力ポート i 毎に用いる分布のテーブル番号 T_i を設定し、Scratch Pad メモリに変数として保持する。
2. R-FIFO にパケットが到着するまで待機し、到着すればパケットを R-FIFO から SDRAM 領域へ転送する。
3. SDRAM 領域への転送が完了すれば、到着したパケットが出力されるポート j を調べ、該当するポートが用いる確率分布テーブル番号を取得する。
4. テーブル内のインデックスとして用いる乱数 r を一様分布で発生させる。
5. 発生した乱数 r がテーブルサイズ S_t より大きい場合、$r \longleftarrow r \pmod{S_t}$ とする。
6. 確率分布テーブル番号およびテーブル内のインデックスより、確率分布にしたがう値を取得する。
7. 取得した値を用いてパケット廃棄判定を行う。廃棄しない場合は到着したパケットを出力ポート・キューの末尾に追加する。

固定確率を用いる場合

1. 各出力ポート i 毎に用いる固定確率 D_i を設定し、Scratch Pad メモリに変数として保持する。
2. R-FIFO にパケットが到着するまで待機し、到着すればパケットを R-FIFO から SDRAM 領域へ転送する。
3. SDRAM 領域への転送が完了すれば、到着したパケットが出力されるポート j を調べ、該当するポートが用いる固定確率 D_i を取得する。
4. 取得した値を用いてパケット廃棄判定を行う。廃棄しない場合は到着したパケットを出力ポート・キューの末尾に追加する。

3.3.6 パケット伝搬遅延発生機能

パケット伝搬遅延時間発生機能は、遅延を発生させるパケットを管理するための遅延パケットキュー、それぞれのパケットに対して発生させる遅延時間を管理する遅延時間キュー、遅延時間キューの末尾の要素の遅延時間を格納する変数 D および遅延時間キューの最終更新時刻を格納する変数 L を用いて実現する。遅延パケットキューおよび遅延時間キューは SRAM 領域に作成し、変数 D および L は Scratch Pad メモリに格納する。

遅延を発生させる必要のあるパケットが到着すると、遅延パケットキューの末尾にパケットを追加し、遅延パケットキューに対応する遅延時間キューに遅延させる時間を格納する。パケットは遅延時間キューによって指定された時間、遅延パケットキュー内に待機する。遅延時間キューは一定の時間間隔で更新され、その際に指定された時間以上待機したパケットは、パケットヘッダ情報より出力ポートを選択し、該当する出力ポートキューの末尾に追加される。次に遅延時間キューの管理アルゴリズムを以下に示す。

1. 新しく到着する遅延パケットの遅延時間を A、遅延時間キュー内の i 番目の要素値は T_i、現在の時刻 (マイクロエンジンのクロックサイクル数) を C とする。
2. 遅延パケットが到着する。
3. 遅延時間キューが空である場合は以下のようにする。

$$T_0 \longleftarrow A \tag{10}$$
$$D \longleftarrow A \tag{11}$$
$$L \longleftarrow C \tag{12}$$

4. 遅延時間キューの要素数が n $(n > 0)$ である場合は以下のようにする。

$$T_{n+1} \longleftarrow A - D \tag{13}$$
$$D \longleftarrow A + D \tag{14}$$

5. ある時間が経過し、遅延時間キューの更新を行う場

合は以下のようにする。

$$T_0 \longleftarrow T_0 - (C - L) \quad (15)$$

$$D \longleftarrow D - (C - L) \quad (16)$$

$$L \longleftarrow C \quad (17)$$

6. 式 (15) の結果が 0 以下であれば、遅延パケットキューから先頭のパケットを出力ポートへ移動させ、遅延時間キューの先頭要素を取り出し、次の要素を新しく先頭要素とする。これを式 (15) の結果が 0 より大きくなるまで繰り返す。

以下にマイクロエンジンにおいて実行される受信側処理の手順を示す。送信側に関しては、特別な処理は行わない。

1. 各出力ポート i 毎に用いる分布のテーブル番号を設定し、Scratch Pad メモリに変数として保持する。
2. SRAM 領域に遅延パケットキュー、遅延時間キューを用意する。
3. R-FIFO にパケットが到着するまで待機し、到着すれば到着したパケットを R-FIFO から SDRAM 領域へ転送する。
4. SDRAM 領域への転送が完了すれば、到着したパケットを遅延パケットキューの末尾に追加する。
5. 到着したパケットの遅延時間は式 (10) および式 (13) 式を用いて計算し、遅延時間キューに追加し、式 (14)、式 (16) および式 (17) を用いて変数の更新を行う。
6. パケットが格納 (ディスパッチ) される際に、遅延時間キューの先頭の要素値を (15) 式を用いて更新し、遅延時間キューの先頭に格納されているパケットが送信可能であれば、そのパケットを遅延パケットキューから出力ポートキューの末尾に移動させる。

3.3.7 設定用シグナリングプロトコル

設定用シグナリングプロトコルとは、ネットワークエミュレータシステムを用いて実験を行う際、各ポート毎の様々な機能の指定およびパラメータ設定を、マイクロエンジンのソースコードを書き換えることなく、外部のユーザ端末から行うためのプロトコルである。この機能により、ユーザはマイクロエンジンプログラミングの知識の有無に関わらず、実験用ネットワークの設定を容易に行うことが可能になる。本報告ではこのプロトコルを NESP (Network Emulator Setup Protocol) と呼ぶ。

NESP は IP (Internet Protocol) を用いてユーザホストとインテル IXP1200 との間の通信を行うため、あらかじめインテル IXP1200 に IP アドレスを割り当てておく。インテル IXP1200 は自身の IP アドレス宛てのパケットを NESP パケットであると判断し、パケットのペイロードを参照してインテル IXP1200 自身の設定を行う。NESP パケットには、実験環境設定開始パケット (SoS パケット)、実験環境設定終了パケット (EoS)、パラメータ設定開始パケット (SoP)、パラメータ設定終了パケット (EoP)、受信確認パケット (ACK) 、パラメータ設定パケット (Prm) がある。パケットの種類はパケット中のフラグによって指定する。パケットは 32 ビットであり、うち 12 ビットのシーケンス番号を持ち、4 ビットで NESP パケットの種類を指定する。残りの 16 ビットの用途はパケットの種類、設定内容によって異なる。

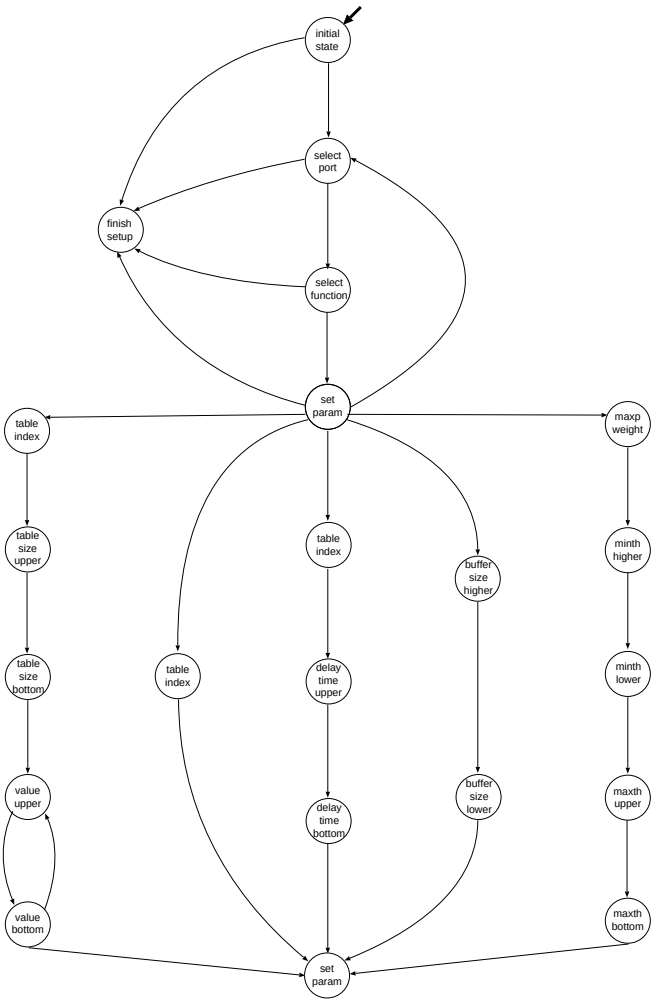

図 9: NESP の状態遷移

図 9 は、インテル IXP1200 側の NESP に基づく状態遷移図と、それに伴う NESP パケットのフォーマットを示す。インテル IXP1200 は起動直後、Initial State にあり、ユーザホストからの NESP パケットの到着を待つ。ユーザは環境設定を行う場合、まず SoS パケットをインテル IXP1200 に送信し、ACK パケットの受信後、パラメータ設定を行う。パラメータ設定は、まず設定を行う入出力ポートを SoP パケットの src および dst フィールドで指定し、SoP パケットに対する ACK を受信後、設定する機能を指定し、その後、機能に応じたパラメータ

の設定を行う。以下に各機能に関する設定パラメータの内容を示す。

- TD
  設定するパラメータはバッファサイズのみであり、バッファサイズは32ビットであるため、2つのNESPパケットで設定できる。
- RED
  設定するパラメータは制御パラメータである w_q、max_p、min_{th}、max_{th} の4つでwqおよびmaxpは8ビット、minthおよびmaxthは32ビットであるため、5つのNESPパケットを用いて設定する。
- 遅延時間分布設定
  設定するパラメータは確率分布テーブル番号、分布のテーブル番号は16ビットであり、32ビットであり、2つのパケットで設定できる。
- 廃棄確率分布設定
  設定するパラメータは確率分布テーブル番号のみであり、番号は16ビットであるため、１つのパケットで設定可能である。
- 確率分布テーブル作成
  設定するパラメータは作成する確率分布テーブル番号、テーブルサイズ、テーブル要素値であり、テーブル番号は16ビット、テーブルサイズは32ビット、テーブル要素値は32ビットであるため、3パケットでテーブル番号およびテーブルサイズを設定し、その後テーブルサイズ分のテーブル要素値を設定する。

パラメータの設定が終了すると、最後にEoPパケットを送信し、指定機能に対する設定の終了をインテルIXP1200に通知する。その後、ACKパケットを受信することにより、設定を終了する。またその後、他の機能、および他の出力ポートに対する設定を続けて行うことも可能である。

4 性能評価

5 おわりに

謝辞

本研究の一部は、日本学術振興会未来開拓学術研究推進事業における研究プロジェクト「高度マルチメディア応用システム構築のための先進的ネットワークアーキテクチャの研究」(JSPS-RFTF97R16301)、および及び平成13年度(財)大川情報通信基金研究助成(01-09)によっている。ここに記して謝意を表す。

参考文献

[1] Kenjiro Cho, “Managing Traffic with ALTQ,” in *USENIX 1999 Annual Technical Conference*, FREENIX Track, (Monterey CA), June 1999.

[2] NISTNet Home Page. available at `http://www.itl.nist.gov/div892/itg/carson/nistnet/`.

[3] Sally Floyd and Van Jacobson, “Link-sharing and Resource Management Models for Packet Networks,” *IEEE/ACM Transactions on Networking*, vol. 3, pp. 365–386, August 1995.

[4] Sally Floyd and Van Jacobson, “Random Early Detection Gateways for Congestion Avoidance,” *IEEE/ACM Transactions on Networking*, vol. 1, pp. 397–413, August 1993.

[5] D. Clark and W. Fang, “Explicit Allocation of Best Effort Packet Delivery Service,” *IEEE/ACM Transactions on Networking*, vol. 6, pp. 362–373, August 1998.

[6] T. E. N. Ion Stoica, Hui Zhang, “A Hierarchical Fair Service Curve Algorithm for Link-Sharing, Real-Time and Priority Services,” in *Proceedings of ACM SIGCOMM ’97*, September 1997.

[7] STORM. available at `http://www.quality-net.co.jp/images/PDF/STORM_web.pdf`.

[8] MicroNET. available at `http://www.adsystems.co.jp/products/micronet/micro_1.html`.

[9] The Network Simulator ns 2. available at `http://www.isi.edu/nsnam/ns/`.

[10] インテル株式会社ホームページ. available at `http://www.intel.co.jp/`.

[11] Intel IXP1200 Network Processor Family. available at `http://www.intel.com/design/network/products/npfamily/ixp1200.htm`.

[12] Intel IXA Software Developers Kit 2.0 for IXP1200. available at `http://www.intel.com/design/network/products/npfamily/sdk2.htm`.

[13] Intel Internet Exchange Architecture Software Developers Kit 2.0 for the IXP1200 Network Processor. available at `http://www.intel.com/design/network/prodbrf/27904101.pdf`.

[14] Intel IXP1200 Evaluation Kit. available at `http://www.intel.com/design/network/products/npfamily/eval_kit.htm`.

[15] Intel StrongARM Processors. available at `http://www.intel.com/design/strong/collateral.htm?iid=strongarm+leftnav%&`.

[16] インテル株式会社プレスルーム. http://www.intel.com/jp/intel/pr/press2000/000531b.htm.

[17] Tammo Spalink, Scott Karlin and Larry Peterson, “Evaluating Network Processors in IP Forwarding,” Tech. Rep. 626-00, Princeton University, November 2000.

[18] Teunis J. Ott, T. V. Lakshman and Larry H. Wong, “SRED: Stabilized RED,” in *Proceedings of IEEE INFOCOM ’99*, March 1999.

[19] Dong Lin and Robert Morris, “Dynamics of Random Early Detection,” in *Proceedings of ACM SIGCOMM ’97*, September 1997.

[20] Steven Black, David Black, Mark Carlson, Elwyn Davies, Zheng Wang, and Walter Weiss, “An Architecture for Differentiated Services.” Internet Engineering Task Force RFC 2475, *Request for Comments 2475*, December 1998.

[21] 線形合同法. available at http://www.ysr.net.it-chiba.ac.jp/data/rand/node5.html.

[22] ポアソン乱数の発生. available at http://aoki2.si.gunma-u.ac.jp/Hanasi/Algo/p_rand.html.

[23] 指数乱数の発生. available at http://aoki2.si.gunma-u.ac.jp/Hanasi/Algo/exp_rand.html.