Vol.2013-ARC-206 No.17
2013/8/1

ネットワークアプリケーションにおける ハードウェアハッシュの有用性検証

山口 史人[†1,a)]　石田 慎一[†2,b)]　西 宏章[†3,c)]

概要：ネットワークにおいて，リッチなサービスを提供するためには，TCP ストリームをネットワークトラフィックから再構築することが必要とされる．このような用途においては高速に処理が可能な機構をハードウェア実装し，膨大な量のストリームを管理することが求められる．TCP ストリームの識別として有効な手段は，それぞれのストリームにハッシュを使用して ID を割り振る手段であり，ハードウエアコストと動作遅延の観点から CRC ハッシュが使用される場合が多い．CRC ハッシュは XOR の木構造で構成され，ハードウェア実装が容易であり低遅延であるという特徴を有する．このような利点から，CRC ハッシュはネットワーク利用に限ってもデータの識別，負荷の割振り，チェックサムなど多岐の用途で利用されている．一方で CRC ハッシュはハードウェア実装には適しているものの，ハッシュ値の衝突率と均一な分散を考慮した場合，最適なハッシュ関数であるとは言い難い．ワークロードによっては，CRC ハッシュは不均一な分布のハッシュ値を出力し，結果的に衝突の増加によるシステムパフォーマンスの低下を招く可能性がある．本報告では，CRC ハッシュ，Jenkins ハッシュと MurmurHash を FPGA に実装し，ネットワークアプリケーションにおける有用性を検証した．

キーワード：ハッシュ，FPGA，TCP コネクション管理

1. はじめに

近年，インターネットの普及により，提供されるサービスはより複雑化・多用化の一途を辿っている．ビッグデータは増大なデータを活用し，よりユーザのニーズに合ったサービスを提供する試みであり，その膨大な情報源としてネットワークトラフィックの利用が想定されている[1]．このネットワークトラフィックを活用した，新たなネットワークサービスを模索する試みがいくつかなされているが，これらのサービスを Fog Computing[2]やルータなどネットワークの中継点で提供するためには，ネットワーク機器において TCP ストリームの再構築を行うことが必要となる．

例えば，Fog Computing において，クラウドアプリケーションにとって重大な問題は，近隣のネットワーク機器の遅延がシステムが要求を満たす上での障害となる点であるとしている．近年研究が盛んにおこなわれている IoT（Internet of Things）などのインターネットに接続する機器では，低遅延に加え，モバイル性や地理的分散も求められる．ストリームやコンテンツ解析は Fog Computing のノードで行うことが望ましい．各ノードによってコンテンツ解析を行うことにより，常に変化するデマンドを把握し，最適な情報ルーティングに生かした低遅延アプリケーションを実現できる．

我々は Internet Open Innovation Platform[3] (IOIP)を実現する，サービス指向型ルータ(Service oriented Router : SoR)[1][4]を提案している．SoR は，ルータにパケットから抽出したデータを蓄える機構とパケット解析機構を備え，ネットワークトラフィックの解析に基づいた，ユーザにとって有益な情報を提供する．SoR はユーザによって定義されたクエリーをもとにパケットの探索とデータベースへの保存を行う．データを蓄える機構として，DBMS を用いる場合，クエリーは SQL をベースにした仕様で構成され，ユーザが得たい内容を定めた仕様に基づいて SoR に伝えることで各種サービスを提供する．このレイヤ 7 までの解析を行える DPI（Deep Packet Inspection）を備えていることが SoR の特徴であるといえる．しかしながら，このコンテンツ，つまりレイヤ 7 解析をネットワークの中間ノードで行うことから，通常のルータでは不要であるトラフィックトレース内部の TCP コネクションの管理が必要とある．

IOIP や Fog Computing そしてサービス指向型ルータなど，ネットウェークの中間地点でパケットを取得する機器において，TCP コネクション管理は DPI を行う上で必要不可欠である．さらにワイヤーレートでの処理を行うにはハードウェアで実装できる事が望ましい．DPI の作業においては，すべてのパケット処理を行うので，欠損なく処理を行うためには膨大な数のコネクション情報を管理する必要がある．先行研究によると，学術情報ネットワーク SINET では，タイムアウト時間を 30 秒とした場合に同時に確立されるコネクションは最大 70,000 に達する．膨大な数のコネクションを管理する場合，ハッシュ値をそれぞれのコネクションの識別 ID として使うのが有効である．そのため低コストで高速なハッシュ関数を用いることが重要である．

ハッシュ関数に着目した TCP ストリームの再構築はあまり議論されていない．Snort[5]などの不正侵入検知システムや Wireshark[6]のようなペイロード解析システムなど，ソフトウェアでは TCP コネクション管理が実装されているものがあるものの，サービス指向型ルータの想定する 10Gbps クラスのネットワークには対応できない．ハードウェアにおける TCP コネクションのハッシングは菅原ら[7],

†1 現在，慶應義塾大学
a) yamaguchi@west.sd.keio.ac.jp
b) sin@west.sd.keio.ac.jp
c) west@sd.keio.ac.jp

ミチルら[8]によって議論されているが，どのようなハッシュ関数を使用したかについては触れておらず，その比較の議論は行われていない．XML，文字列探索，エンコード・デコード，gzip 展開などをハードウェアコプロセッサで行っている IBM 社の PowerEN においても，ストリームの管理はユーザの作成するプログラムが責任を持って管理しなければならない[9]．この論文はネットワークアプリケーションにおける異なるハッシュの特性を評価し，その有用性について検証する．

2. ハッシュ関数

この章では，現在行われているネットワーク機器に使用されるハッシュのサーベイをし，ハードウェア実装をするハッシュ関数に必要な要件について検討する．

ハッシングとは入力データに計算処理を施すことにより固定長の値を出力する方法である．ほとんどの場合，出力される値は入力データより長さが短いため，ハッシュは入力データのダイジェストととらえることができる．主な使用用途はデータストレージ，負荷分散，暗号化，データ識別，そしてチェックサムなどである．

ハッシュ値の特性はハッシュ関数によって大きく変化する．多くのハッシュ関数は四則演算，論理演算，ビットシフトなどの簡単な計算の組み合わせで行われる．ソフトウェア実装による暗号化目的の研究は数多く存在する[10][11][12]が，ネットワーク機器での活用が期待できる非暗号目的でハードウェア実装が可能なハッシュ関数に関する研究は少ない．Jiang らによる研究ではネットワークアプリケーションにおけるハッシュ関数について調べているが，ハードウェア実装した場合のハッシュ関数の特性などについては触れていない[13]．ハッシュ関数をハードウェア実装し，満足するパフォーマンスを得るためには，以下の項目を考慮する必要がある．

2.1 遅延

ハードウェア実装されたハッシュ関数のスループットはその回路の遅延によって決まる．高速なネットワーク機器で使用する場合，ハッシュ値の計算が 1 クロックで終了すること，さらには組み合わせ回路によって構成可能であることが望ましい．組み合わせ回路で実装可能なハッシュ関数の場合でも，論理段数をより少なくすることや，多入力ロジックの利用を抑えるなど，ハッシュ関数を構成する回路の遅延をできる限り小さくすることが重要である．たとえば，掛算器の回路は加算器に比べ複雑で遅延も大きくなるため，ハッシュ関数の計算式から掛け算を排除したほうが低レイテンシな回路が構成でき，好ましい．このことから，FPGA や ASIC 上にハッシュ関数を実装する場合，以下の 2 つの条件は重要である．

- 組合せ回路で構成可能
- クリティカルパス遅延が小さい

2.2 回路規模

ハードウェア実装を行う際は回路規模が小さいことも求められる．ハードウェア実装では，回路が大きいほどファブリケーションコストが上がるほか，リーク電流による消費電力の増加を引き起こす．分散処理エンジンなど，多数のハッシュ関数の使用が想定される環境ではこの回路規模を考慮する必要がある．

2.3 均一な分布

理想的なハッシュ関数は，あらゆる入力データを与えても出力されるハッシュ値の分布がハッシュ値の出力範囲内で均一であるべきである．また，入力データによるハッシュ値の偏りを無くし，均一な分布を得るためには，ハッシュ関数にはわずかな入力データの変化でも出力されるハッシュ値が劇的に変わる特性が求められる．本報告では，この指標としてハッシュ関数のアバランシュ特性を評価した．アバランシュ特性については第 3 章で詳しく述べる．

2.4 衝突率

ハッシュ値の衝突は，異なる 2 つ以上の入力データから同じハッシュ値が生成された場合に発生する．ハッシュ値は入力データの単なるダイジェストであるため，限定的な条件下を除きハッシュ衝突を無くすことは不可能であり，確率的に必ず発生する．しかし，ハッシュ値の生成に偏りのあるハッシュ関数は，他のハッシュ関数に比べハッシュ衝突率が大きくなる．ハッシュの偏りが「密」な範囲は「粗」な部分と比べより多くの入力データが示されているため衝突率が高くなる．

以上の 4 つの項目は低レイテンシネットワーク機器に搭載するハードウェアハッシュ関数に欠かせないことである．この要求をもとに，本報告では以下の 3 つのハッシュ関数を評価対象とした．

- CRC ハッシュ
- Jenkins ハッシュ
- MurmurHash

これらのハッシュ関数は，計算式が単純でハードウェア実装に適していること，そして組み合わせ回路で実装可能であることから本報告の調査対象とした．それぞれのハッシュについて簡潔に説明する．

A) CRC ハッシュ

CRC ハッシュはもともとデータ通信などにおけるエラー検出を行うためのチェックサムとして提案された[14]．チェックサムとして幅広く使われており，一例としてイーサネットフレームのエラー検出用 FCS などが挙げられる．CRC ハッシュは攪拌能力が高いため，チェックサム以外でもデータテーブルの参照などにも使われてきた．

CRC ハッシュは排他的論理和のみで構成可能であるため，組み合わせ回路によるハードウェア実装は容易に可能である．CRC ハッシュはハードウェア実装においてこのよ

うなメリットがあるが，ワークロードによってはハッシュ値の偏りが大きい場合があり，ネットワーク機器全体のパフォーマンスにも影響を与える可能性がある．この報告では CRC-32 で用いられる多項式を用いた．

B) Jenkins Hash

Jenkins ハッシュは Bob Jenkins によって提案されたハッシュ関数で，Lookup3（2009）や SpookyHash（2011）などいくつかのバリエーションがある[15][16]. この報告では評価対象として Lookup3 を使用する．Lookup3 はビットシフト，加算，論理和，そして排他的論理和によって構成される．Lookup3 はソフトウェア実装においてはすぐれたアバランシュ性を示す．

C) MurmurHash

MurmurHash は Austin Appleby により提案されたハッシュ関数で，積算，排他的論理和，そしてビットシフトにより構成される[17]. MurmurHash にはいくつかのバリエーションがあるが，評価には MurmurHash2 を使用した．ハッシュ値の計算プロセスはシンプルで，mix と finalization の 2 つによってのみ構成される．単純な構造に加え，ソフトウェア実装においては，すぐれたアバランシュ性を示す．

3. 評価方法

ハッシュ関数の評価は 2 つのアプローチから行った．まずハッシュ自体の特性を調べるためにアバランシュ性と動作遅延の評価を行う．この評価ではハッシュ値の分散とハッシュ関数の高速性を検証する．

2 つ目のアプローチは実際にハッシュ関数をネットワーク機器に実装し，それぞれのハッシュ関数におけるシステム全体のパフォーマンスを評価する．それぞれのハッシュ関数は TCP コネクション管理機構に実装され，実ネットワークトラフィックでのパフォーマンスを比較する．

3.1 アバランシュ性評価

ハッシュ値の分布を調べる最も正確な方法は入力の取りうる値すべてに対するハッシュ値を計算し，その分布を調べることである．しかし全入力の組み合わせの数は膨大となるため，この方法は現実的ではない．そこで，ハッシュ値の特性を調べる有効な方法の一つとして，アバランシュ性評価が提案されている．

アバランシュ性評価は，入力のビットに変化があった場合，出力されるハッシュ値のビットごとに値が変化する確率を調査する. アバランシュ性評価を行い, すべての入力，出力ビットの組み合わせにおいて均一な変化の確率を示す場合，そのハッシュ関数はわずかな入力の変化でも劇的に異なるハッシュ値を出力できる能力を有することになる．アバランシュ性は直接的にはハッシュ値の均一な分布を示すわけではないが，この特性により出力が入力に依存しづらいので，間接的にハッシュ値の分布が均一になりやすいといえる．

今回の評価で行うアバランシュ性評価では，1 ビットごと差分のあるハッシュ値を与え，状態が変化した出力ビットを記録し，この操作を 1 万回行った場合の各ビットが変化する確率を求める．アバランシュ性を満たすハッシュ関数はこの確率が 50%に近づく．

このアバランシュ性の結果はグラフの色分布による表示と，50%の変化確率からの差分を示すバイアスの 2 つの方法で示す．

3.2 回路遅延

ハードウェア実装された回路の最大動作周波数は，その回路の持つ動作遅延によってきまる．回路内で最も大きな遅延を有するクリティカルパスにおける遅延を調べ，その逆数をとることにより最大動作周波数を得ることができる. ハッシュ関数のクリティカルパスの遅延を小さくすることは高速なハードウェアハッシュのためには重要である．

回路遅延の計測のため，評価を行うハッシュ関数を FPGA 実装用に論理合成した．ハードウェア記述は Verilog HDL を用いて行った．シミュレーションは Cadence NC-Verilog LDV および Simvision 06.20 を用いて行った．タイミング解析および合成は ISE Design Suite 14.2 で行った．FPGA 評価デバイスである LogicBench に搭載された Xlinix Vertex 5 XC5VLX330T へ実装した．

3.3 アバランシュ性評価

ハッシュの評価を行うネットワークアプリケーションとして TCP コネクション管理機構を用いた. TCP コネクション管理は DPI などコンテンツ解析を行う上では不可欠なので今回のハッシュの実装評価の対象とした. 図 1 に TCP コネクション管理機構の概要図を示す.

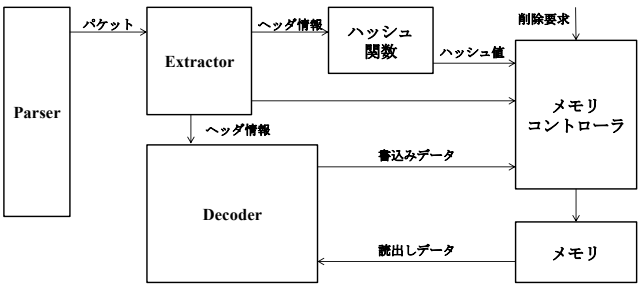

図 1　TCP コネクション管理機構の概要図

TCP コネクション管理は以下の手順で行われる．

(1) ヘッダ情報の抽出

TCP/IP ヘッダの情報は Parser と Extractor によって抽出される. Parser は今回実装を行った LogicBench によって提供されるもので，各ヘッダ情報とペイロードをフォーマットする．Extractor はそのフォーマットされたヘッダ情報から 4 タプル（Src IP，Dst IP，Src Port，Dst Port）を抽出する．

(2) ハッシュ値の計算

Extractor によって抽出された 4 タプルのハッシュ値を計算

する．4 タプルが同じ場合は同じコネクションと判断できるのでこのハッシュ値を用いてコネクションの管理を行う．本システムではハッシュ値をメモリのアドレスとして使用するハッシング法を用いる．

(3) **デコード**

Decoder は到着したすべてのパケットに対し，メモリや Extractor からの情報をもとに現在のコネクション状態を調べる．また，状態の更新が必要な場合はメモリ内のエントリを変更する．図 2 に Decoder のフローチャートを示す．

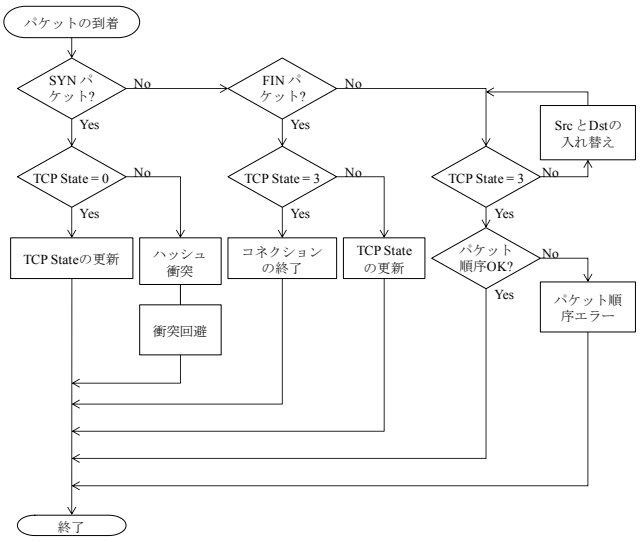

図 2　Decoder のフローチャート

本機構は FPGA に実装され，動作確認を行った．出力の確認は Agilent 16823A ロジックベンチによって観測した．また TCP コネクション管理機構におけるハッシュ関数の特性はデータ処理の都合上ソフトウェアシミュレータで行った．このシミュレータによって，TCP コネクション管理を行った際のメモリアクセス回数の変化を計測した．今回のシステムにおいてメモリアクセスが増加するのは，ハッシュ値の衝突が起き，リニアプロービングによって空いているメモリ領域を探すときのみである．そのため，同じワークロードにおいてメモリアクセス回数が少なくなった場合，それはハッシュ値の衝突が少ないことを意味する．

この評価で用いるワークロードは WIDE ネットワークから取得した．それぞれのワークロードは 15 分間のインターネットトラフィックを含む．

TCP コネクション管理機構では使用していないが，ブルームフィルタにおけるハッシュの振る舞いも検証した．ブルームフィルタはデータ集合のメンバを管理するためのデータ構造であり，その効率性からネットワーク機器でも用いられる[18]．ブルームフィルタはハッシュ値を使用し，入力されたデータが登録されているメンバであるかを参照するため，ハッシュ値の衝突が少ないほど高精度になる．そこで，WIDE ネットワークで取得した 4 万件の IP アドレスのハッシュ値を計算し，固有なエントリの減少数を比較した．利用したハッシュ値は 20 ビットである．

4. 評価結果

4.1 アバランシュ性評価

CRC ハッシュ，Jenkins ハッシュ，MurmurHash のそれぞれのアバランシュ性を図 3 に示す．横軸は入力ビット，縦軸は出力ビットを示し，グラフ上の各点は該当する入力ビットが変化したときに出力ビットが反転される確率を示す．反転する確率は色で表記している．なお，Jenkins ハッシュおよび MurmurHash は特性を見やすくするために色表示を強調していることに注意されたい．

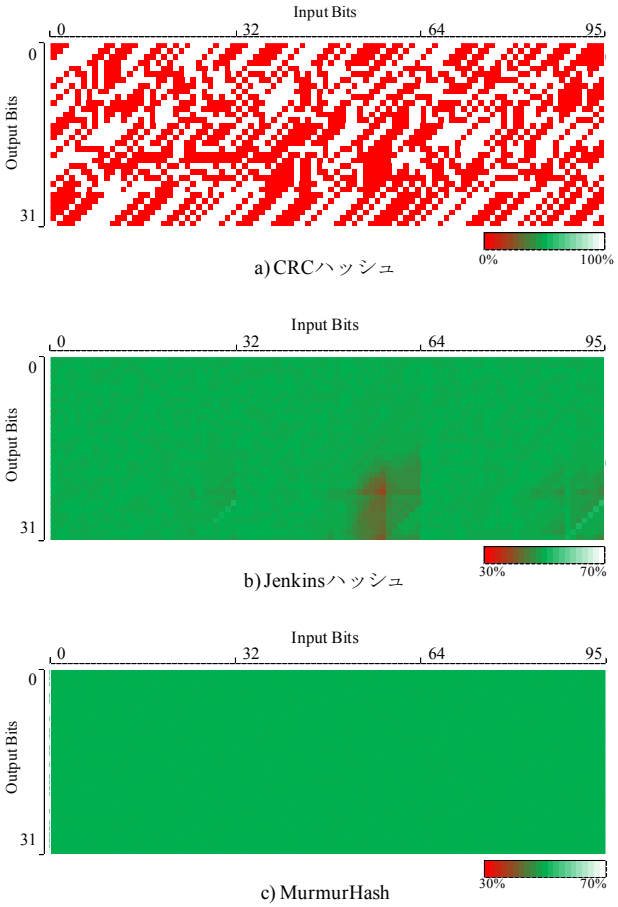

図 3　アバランシュ性評価

CRC ハッシュのアバランシュ性評価では，出力ビットの反転確率が 0%もしくは 100%だけとなっている．この特性は，入力に 1 ビットの変化があった場合，必ず出力の 1 ビット以上が反転するのでチェックサムに適していると言える．しかし，他の用途で使用した場合はハッシュ値の偏りを生じ，衝突率の上昇につながりかねない．

Jenkins ハッシュと MurmurHash はともにおおむね 50%の反転確率を示す．表 1 に各ハッシュの反転確率の 50%からのバイアスを示す．

表 1　ハッシュ関数のバイアス

Hash	CRC	Jenkins	Murmur
Bias	-50.0% +50.0%	-12% +3%	-0.21% +0.16%

Jenkins ハッシュは局所的に特性が悪化している箇所が見られるが，Jenkins ハッシュと MurmurHash はともに優れたアバランシュ性を示すと言える．

4.2 動作遅延

それぞれのハッシュ関数を FPGA に実装した場合の動作遅延を図 4 と表 2 に示す．遅延は入力データ幅によって変化するので，それぞれの入力幅における遅延を調べた．なお，表 2 の括弧内の値はクリティカルパスのゲート数を示す．

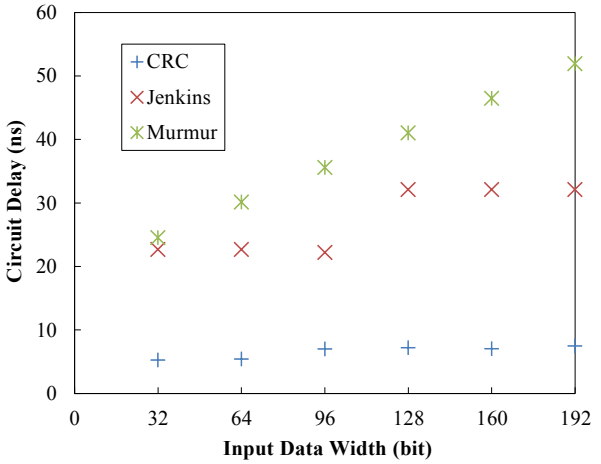

図 4　ハッシュ関数の動作遅延

表 2　ハッシュ関数の動作遅延

Input Width	CRC	Jenkins	Murmur
32	5.241ns (5)	22.673ns (96)	24.513ns (14)
64	5.429ns (5)	22.665ns (96)	30.136ns (17)
96	6.987ns (9)	22.172ns (96)	35.584ns (20)
128	7.210ns (9)	32.114ns (125)	41.032ns (23)
160	7.023ns (9)	32.086ns (125)	46.480ns (26)
192	7.453ns (9)	32.086ns (125)	51.929ns (29)

CRC ハッシュはいずれの入力データ幅においても最も小さな遅延で回路を構成できた．一方 Jenkins ハッシュと MurmurHash は最も遅延が少なかった入力 32 ビットにおいても CRC に比べ約 4 倍の遅延がある．評価の中で最も遅延の大きかった 192 ビットにおいてはそれぞれ約 4 倍，7 倍の遅延となった．ハッシュ単体で実装するケースは少ないので，単純に最大動作周波数が数分の一になるとは言えないが，高速なアプリケーションにおいては考慮する必要がある．今回実装した TCP コネクション管理機構の場合，CRC ハッシュ，Jenkins ハッシュを用いた場合のシステム全体の遅延はそれぞれ 97.7ns，120.3ns となり，Jenkins ハッシュを用いた場合は約 19%動作周波数が低下する．

Jenkins ハッシュは 96 ビット単位で計算を行うため，動作遅延が 96 ビットごとに大きくなる．また，MurmurHash はゲート数が少ないにも関わらず，動作遅延が 3 つのハッシュの中で最も大きくなる．これは MurmurHash のハッシュ計算式の中に掛算が含まれており，掛算器の構成が必要となるためである．

4.3 TCP コネクション管理

図 5 に TCP コネクション管理機構を実装した FPGA からの出力信号をロジックアナライザで取得した際の様子を示す．Crcout1 と Crcout2 はハッシュ値，valid は確立済みの TCP コネクションであるかどうかを示す．

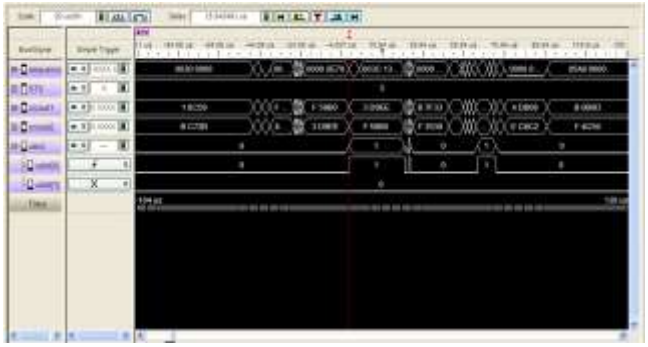

図 5　TCP コネクション管理機構の出力

表 3 に TCP コネクション管理を行った際の各ワークロードにおける合計メモリアクセス数を示す．濃い塗りつぶしはより良い結果を示す．

Jenkins ハッシュと MurmurHash は 12 個のワークロードのうち，それぞれ 5 つで最も少ないメモリアクセス数であった．Jenkins ハッシュと MurmurHash を使った場合のメモリアクセス数の差はわずか 0.45%であり，両者の TCP コネクション管理機構におけるパフォーマンスはほぼ等しいといえる．

CRC ハッシュは 2 つのワークロードにおいては最もメモリアクセス数が少なくなったが，Jenkins ハッシュと MurmurHash と比べ，メモリアクセス数の削減率は最大でも 8.9%にとどまる．一方，ワークロード *200904020215* および *200904020130* においては，Jenkins ハッシュと MurmurHash を用いることによりそれぞれ 83%，72%のメモリ削減率を実現している．これは CRC ハッシュが示した削減率を大きく上回る．全ワークロードでの CRC ハッシュに対する平均メモリアクセス数の削減率は，Jenkins ハッシュで 25.2%，MurmurHash で 25.3%となる．以上を踏まえると，TCP コネクション管理機構においては Jenkins ハッシ

ュと MurmurHash を用いることによりシステム全体のパフォーマンスが向上することが確かめられた．

ブルームフィルタにおけるハッシュ化前後の固有なエントリ数を表 4 に示す．またエントリ数の欠損率を表 5 に示す．

表 3　メモリアクセス数

Workload	CRC	Jenkins	Murmur
200904020400	31694	31689	31387
200904020215	2256101	391449	391905
200904020500	694543	267523	267760
200904020730	91229	99130	98928
200904020200	48647	48388	48089
200904021230	109547	95828	96319
200904022215	129020	113313	113182
200904021245	63310	63280	63708
200904020145	42555	42597	42771
200904022345	180958	142337	141133
200904020130	1004644	283942	285021
200904021215	390852	199243	198386

表 4　固有エントリ数

Workload	Original	CRC	Jenkins	Murmur
l2res	1084	973	1074	1078
l5res	649	608	642	647
l10res	557	521	554	555

表 5　欠損率

Workload	CRC	Jenkins	Murmur
l2res	10.2%	0.9%	0.6%
l5res	6.3%	1.1%	0.3%
l10res	6.5%	0.5%	0.4%

Jenkins ハッシュと MurmurHash はともに欠損率が 1%未満であり，元の固有エントリの 99%を保持している事が分かる．これはブルームフィルタに用いるハッシュ関数として適しているといえる．

以上の 4 つの評価により，Jenkins ハッシュと MurmurHash は TCP コネクション管理機構とブルームフィルタにおいて CRC より優れたパフォーマンスを有することが分かる．このパフォーマンスの向上が，動作遅延の増加に伴うスループットの低下によって相殺されない場合は CRC ハッシュに変わり Jenkins ハッシュと MurmurHash を用いるメリットがあると言える．

5. 結論

CRC ハッシュ，Jenkins ハッシュ，そして MurmurHash のハードウェア実装されたネットワーク機器におけるハッシュ関数としての有用性を検証した．それぞれのハッシュ関数は FPGA へ実装され，その動作遅延を計測した．また TCP コネクション管理機構とブルームフィルタにそれぞれのハッシュ関数を用いた場合のパフォーマンスの変化も比較した．TCP コネクション管理機構の場合，CRC ハッシュに比べ，Jenkins ハッシュは 25.2%，MurmurHash は 25.3%メモリアクセス数を削減した．またブルームフィルタでは Jenkins ハッシュ, MurmurHash はともに欠損率が 1%未満となった．以上を踏まえ，ハードウェア実装されたネットワーク機器において，CRC ハッシュの代わりに Jenkins ハッシュと MurmurHash を用いるとパフォーマンスがおおむね向上すると言える．

謝辞　この研究は，文部科学省　社会システム改革と研究開発の一体的推進気候変動に対応した新たな社会の創出に向けた社会システムの改革プログラム「グリーン社会 ICT ライフインフラ」，環境省 地球温暖化対策技術開発・実証研究事業「大学キャンパスの省 CO2 化に向けたキャンパスエネルギーマネジメントの実証研究」, 文部科学省科学技術研究費補助金基盤研究 B「コンテンツベース・スマートコミュニティインフラの構築と展開」(25280033)の一環としてなされた．

本研究は東京大学大規模集積システム設計教育研究センターを通し，シノプシス株式会社，日本ケイデンス株会社の協力で行われたものである．

参考文献

1) K. Inoue, D. Akashi, M. Koibuchi, H. Kawashima, H. Nishi, “Semantic Router using Data Stream to Enrich Services,” 3rd International Conference on Future Internet Technologies, pp. 20-23, June 2008.
2) F. Bonomi, R. Milito, J. Zhu, S. Addepalli, “Fog computing and its role in the internet of things,” In Proceedings of the first edition of the MCC workshop on Mobile cloud computing (MCC '12). ACM, New York, NY, USA, pp. 13-16.
3) www.openinter.net
4) K. Takagiwa, R. Kubo, S. Ishida, K. Inoue, H. Nishi, “Feasibility Study of Service-oriented Architecture for Smart Grid Communications”, 22nd IEEE International Symposium on Industrial Electronics, TF-006505, 2013 (to be appeared)
5) http://www.snort.org/
6) http://www.wireshark.org/
7) Y. Sugawara, M. Inaba, and K. Hiraki, “High-speed and Memory Efficient TCP Stream Scanning Using FPGA,” in Proc.

International Conference on Field Programmable Logic and Applications, Aug 2005.

8) Mythili Vutukuru, Hari Balakrishnan, Vern Paxson, “Efficient and Robust TCP Stream Normalization,” In Proceeding of the 2008 IEEE Symposium on Security and Privacy, pp. 96-110, 2008”

9) A. Krishna, T. Heil, N. Lindberg, F. Toussi, S. VanderWiel. “Hardware acceleration in the IBM PowerEN processor: architecture and performance,” In Proceedings of the 21st international conference on Parallel architectures and compilation techniques (PACT '12). ACM, New York, NY, USA, pp. 389-400.

10) S. Bakhtiari, R. Safavi-Naini, J. Pieprzyk, “Cryptographic Hash Functions: A Survey,” Technical Report pp. 95-09, Department of Computer Science, University of Wollongong, July 1995

11) S. Al-Kuwari, J.H. Davenport, R.J. Bradford, “Cryptographic Hash Functions: Recent Design Trends and Security Notions,” IACR Cryptology ePrint Archive, 2011, pp. 565-565.

12) R. Purohit, U. Mishra, A. Bnasal, “A Survey on Recent Cryptographic Hash Function Designs”, International Journal of Emerging Treands & Technology in Computer Science, vol. 2, issue 1, pp. 117-122, January - February 2013

13) P. Jiang, J. Liu; Z. Qin, “On hashing techniques in networking systems,” Information Networking and Automation (ICINA), 2010 International Conference on , vol.2, no., pp. V2-444,V2-449, 18-19 October 2010

14) W.W. Peterson, D.T. Brown, “Cyclic Codes for Error Detection,” Proceedings of the IRE , vol.49, no.1, pp. 228-235, January 1961.

15) B. Jenkins, “Algorithm alley: Hash functions,” Dr. Dobb's Journal of Software Tools, 22(9), September 1997.

16) B. Jenkins, “A Hash Function for Hash Table Lookup”, http://www.burtleburtle.net/bob/hash/doobs.html

17) A. Appleby, “MurmurHash,” https://sites.google.com/site/murmurhash/

18) B. Bloom, “Space/time trade-offs in hash coding with allowable errors,” Communications of the ACM 13, July 1970, pp. 422-426.