卒業製作 2003 年度 (平成 15 年度)

再構成可能なハードウェアを用いたネットワークアプライアンスの提案

慶應義塾大学 環境情報学部
松谷 宏紀
t00870hm@sfc.keio.ac.jp

指導教員
村井 純
徳田 英幸
楠本 博之
中村 修
南 政樹

平成 15 年 12 月 29 日

概 要

ネットワーク技術の進歩と普及により，ネットワークに接続可能なセンサや情報家電が注目を浴びている．このようなネットワークアプライアンスには健康管理や介護支援などへの応用が検討されており，今後，人々の生活において重要な役割りを担うことが期待されている．

近年，ネットワークアプライアンスに求められる機能は高機能化している．例えば，ユーザインターフェイスにおいて音声や動画，静止画を扱う機器は増えつつあり，さらに，プライバシや安全性の面から考えて通信の暗号化や認証も必須となりつつある．マルチメディアデータの取扱いや通信の暗号化は負荷の高い処理であるため，処理能力が限られた計算機では専用ハードウェアの利用が現実的である．しかし，ネットワークアプライアンスは生活環境に深く埋め込まれたり数多く設置されることがあるので，新機能を追加したり利用形態に応じてハードウェア構成を変更することは難しく，専用ハードウェアの利用はシステムの応用範囲を制限し兼ねない．そこで，本研究では，ネットワークアプライアンスを対象に，利用形態に応じたハードウェアの自動更新を実現する．本研究によって，ネットワークアプライアンスの不具合の修正や新機能の追加はもちろん，設計者および利用者が意識せずとも各ネットワークアプライアンスが利用形態に最も適したハードウェア構成に自動調整されるようになる．短期間の利用目的のために，ネットワークアプライアンスを使い捨てることは無くなる．

本システムを実現するために論理の再構成が可能なハードウェアを用いる．この再構成可能なハードウェアは，構成データと呼ばれる情報をデバイスに書き込むことで任意のハードウェアを実現する．選択可能な構成データの種類が多いほど，選択可能なハードウェアのバリエーションが豊富になり，ネットワークアプライアンスはより利用形態に適した構成に近付く．そのためには，より多くの構成データを管理する必要があるが，記憶領域が限られたネットワークアプライアンスでは管理出来る構成データの数に制限がある．本システムでは，ネットワークアプライアンスが利用する構成データをネットワーク上で管理する．そして，それぞれのネットワークアプライアンスにおいて利用形態に変化が生じると，新たな利用形態に適した構成データをネットワーク上からダウンロードしハードウェアを更新する．利用形態に適したハードウェア構成は，実行している処理の利用頻度を基に判断する．

実装では，本システムの有効性を示すため，利用頻度が高い暗号アルゴリズムを自動的にハードウェア化する IPsec スタックを構築した．評価では，まず，回路変更にかかるオーバヘッドを測定し，そのオーバヘッドに見合った回路変更ポリシとして利用頻度に基づいた回路変更の必要性を示した．次に，利用頻度に基づいた回路変更の動作実験を行い，本システムによって回路変更の頻度を抑えつつ効率的に処理をハードウェア化出来ることを確認した．以上より，ネットワークアプライアンスにおいて，様々な処理を効率的にハードウェア化するには，本システムが効果的であることを確認した．

キーワード

1, 動的適応型ハードウェア　2, FPGA　3, 情報家電　4, IPv6　5, IPsec

Abstract

Due to the progress and wide diffusion of network technology, sensors and appliances with network interfaces stand in the center of the spotlight. These network appliances are considered for use with nursing and health care, and are considered to play an important role in our living, in the near future.

In recent years, functional requirement toward a network appliance is becoming more sophisticated. For example, appliances which use motion pictures and audio data for user interface is increasing, and cosideration for a secure communication and insuring privacy are becoming indispensable. Since these functions require high CPU load when processing, it is reasonable to use application-specific hardware for network appliances with limited resources. However, because many of these devices are embedded in the our living, it is often difficult to add or update both new and existing features dynamically according to their usage, which may end up limiting thier application range when used as an application-specific hardware. In this research, we provide network appliances which update their hardware features according to their environment. From this research, not only adding and updating of features in network appliances will be realized, but also an automatic adjustment for each network appliances to the most optimized hardware configuration will be realized. Network appliances for the short term use are not disposable.

In realizing the system, hardware which are capable of logical reconfiguration is used. Information called configuration data are written into each devices to build an arbitrary hardware. The more variety in these configuration data, the more variation in selecting the hardware, which will bring the netwobrk appliance to optimized the composition more suitable to their environment. However, the number of manageable configuration data is limited due to the limitation of available memmory space in a network appliance. In this system, configuration data used by the network appliances are managed at a network server. When a change in the environment arises, new configuration data suitable for the new environemnt is downloaded from the server and then applied to update the hardware. The suitable configuration data is selected based on the use frequency of performed processings.

In order to show the effectiveness of this system, IPsec stack which automatically create a hardware using the most frequently used encryption algorithm is implemented. As an evaluation, overhead when changing hardware features was measured and showed the need for this system. Conformance test conducted based on the use frequency verified that the system is capable of effectively applying application-specific hardware to processes while controlling the frequency of changing the hardware feature to its minimum. From this research, appling application-specific hardware efficientely for various processes on network appliances was verified to be effective.

Keywords

1, Dynamic Adaptive Hardware　2, FPGA　3, Network Appliance　4, IPv6　5, IPsec

目 次

図 目 次

表 目 次

第1章　はじめに

1.1　背景

次世代インターネットプロトコルとして実験的に運用されていたIPv6（Internet Protocol, Version 6）[1]は，すでに商用サービスが開始され，様々なIPv6対応機器が登場するなど社会に定着しつつある．IPv6にはいくつもの特徴があるが，その中でも，広大なアドレス空間とアドレス自動設定機能[2]は多数の機器を無設定でネットワークに接続可能にする．そのため，IPv6はセンサや情報家電などネットワークアプライアンスへの応用が期待されており，このような機器を対象としたIPv6スタック[3]も発表されている．一方，デジタル化によって進化を遂げた情報家電の分野では，ネットワーク対応機器が発売されるなどネットワーク化が進んでおり，今後，IPv6がネットワークアプライアンスにおける基盤技術となることが期待されている．

ネットワークアプライアンスには様々な形態が想定されるが，大きく分けて，AV系ネットワークアプライアンスと制御系ネットワークアプライアンスに分類出来る．

AV系ネットワークアプライアンスにはゲーム機や各種AV機器が含まれ，すでに広く普及したものもある．例えば，ゲーム機においてはネットワーク対応機種が広く出回っており，対応ソフトやコンテンツは充実している．また，ブロードバンド時代の到来と共に，インターネットを介したTV電話システムが遠隔会議や遠隔授業[4]などに利用されるようになった．

制御系ネットワークアプライアンスにはセンサやスイッチ，さらに，電子レンジや冷蔵庫などの白物家電が含まれ，様々なネットワークアプライアンスが実用化に向けて検討されている．例えば，みまもりほっとライン[5]では，無線通信機能を内蔵した電気ポットを利用して，電気ポットの利用状況を離れて暮らす家族に通知するサービスを商用化している．また，藤沢市が中心となり藤沢市保健医療財団や慶應義塾大学が参加しているe- ケアタウンふじさわ[6]では，IPv6を用いて遠隔コーチング可能なエアロバイクや万歩計などを利用して，健康管理や介護支援の実証実験を進めている．さらに，InternetCAR Workshop2003[7]では，IPv6通信機能を持った小型センサを用いて自動車内の情報を取得する実験が行われた．

このように，ネットワークアプライアンスはAV系機器がもたらすエンターテイメントだけでなく，健康管理や介護支援など人々の生活において重要な役割りを担う．

1.2　本研究の目的

ネットワークアプライアンスに求められる機能は日々高機能化している．例えば，ユーザインターフェイスにおいて音声や動画，静止画を扱う機器は増えつつあり，さらに，プライバシや安全性の面から考えて通信の暗号化や認証も必須となりつつある．

マルチメディアデータの取扱いや通信の暗号化は負荷の高い処理である．本研究の目的は，処理能力が制限された環境において，これらの処理を効率的にハードウェア化することである．

ネットワークアプライアンスが本格的に普及するには以下の2点を満たす必要がある．

- 低管理コスト
  PC などと違いユーザという概念が無いものや，設定のためのインターフェイスを持たないものがある．ユーザの介在なしに日常動作可能であり，機能の追加，更新出来なければならない．

- 長寿命
  利用環境の変化や標準化の動向によって，後に機器が使いものにならなくなる可能性がある．このような変化に適応するため，後天的に機能を追加，更新出来なければならない．

例えば，PC 上で動作するソフトウェアにおいて不具合の修正を行うことは珍しくないが，この作業をネットワーク化された家中の白物家電に対して行うことはユーザにとって負担が大き過ぎる．また，PC や携帯電話を 2〜3 年ごとに買い換えることはあっても，同じ様なペースで冷蔵庫などの白物家電や壁に埋め込まれたセンサ類を買い替えるとは考えにくい．

機器の管理コストを小さくしたり，機器の寿命を長くするには様々なアプローチがある．例えば，文献 [8] では，組み込み用途に適した Mobile IPv6 の実装方法を提案している．機器が Mobile IPv6 に対応すると通信に使用するアドレスが変化しなくなり，機器の移動によるアドレスの変化を把握する必要がなくなる．また，文献 [9] では，ネットワークを介してソフトウェアを更新出来る組み込み Operating System（OS）を提案している．このような OS を用いることで出荷後の不具合の修正が可能になり，機器の寿命は長くなる．

本研究ではネットワークアプライアンスを対象に，ハードウェアの自動更新を実現する．一度出荷したハードウェアに対する不具合の修正や新機能の追加手法はすでに実用化しているが，本研究では利用形態に応じてハードウェアを自動更新する．例えば，MP3 の再生と通信の暗号化を行うシステムにおいて，MP3 の再生が高頻度で実行される状況では MP3 再生処理がハードウェア化され，一方，暗号処理が高頻度で実行される状況では暗号処理がハードウェア化されるようになる．

本研究によって，ネットワークアプライアンスの不具合の修正や新機能の追加はもちろん，設計者および利用者が意識せずとも各ネットワークアプライアンスが利用形態に最も適したハードウェア構成に自動調整されるようになる．

1.3 本論文の構成

本論文では，まず，第 2 章にて現状のネットワークアプライアンスにおいて利用可能な技術を説明し，現状に欠けている点を指摘する．第 3 章にて目指すべきシステムを提示し，そのためのアプローチであるネットワーク上での構成データの管理およびネットワークを介したハードウェアの自動更新について述べる．そして，第 4 章にて本研究のアプローチを実現するためのシステムを設計する．第 5 章にて本研究の実装例として利用頻度が高い暗号アルゴリズムを自動的にハードウェア化する IPsec[10] スタックを説明する．最後に，第 6 章にて本研究のアプローチを評価し，第 7 章にて本論文をまとめる．

第2章　ネットワークアプライアンス

本章では，まず，2.1 節にて本研究の検討対象であるネットワークアプライアンスを定義する．そして，2.2 節にてネットワークアプライアンスの基本構造を説明し，処理能力の制限について考察する．2.3 節にて再構成可能なハードウェアを用いた専用ハードウェアの利用について説明する．最後に，2.4 節にて現状のネットワークアプライアンスに欠けている点を指摘する．

2.1　本研究の検討対象

第 1 章にて説明した通り，ネットワークアプライアンスとしてセンサや白物家電，ゲーム機など幅広い用途の機器が実用化されており，さらに，続々と新たなネットワークアプライアンスが実用化に向けて検討されている．これら全てに対し効果的なシステムを構築することは困難であり，本研究では以下の条件を満たす機器を検討対象とする．

(1) PC のような汎用性を持たず，特定用途のために利用される．

(2) ネットワーク機能を有する．

(3) マルチメディアデータの取扱いや通信の暗号化など負荷の高い処理を行う．

(4) FPGA に代表される再構成可能なハードウェア部分を有する．

条件 (1)〜(3) は今後登場し得るネットワークアプライアンスの多く該当すると想定している．第 1 章にて説明した通り，今後ネットワークアプライアンスに求められる機能はますます高機能化し，高負荷な処理を行う機会は増える．ネットワークアプライアンスが条件 (4) の通り再構成可能なハードウェア部分を持ち本システムを動作させることで，条件 (3) に示す高負荷な処理を効率的に実現出来るようになる．なお，再構成可能なハードウェアについては 2.3.2 小節にて説明する．

2.2　ネットワークアプライアンスの基本構造

本節では，まず，2.2.1 小節にてネットワークアプライアンスにおけるハードウェア構成，2.2.2 小節にてソフトウェア構成を説明する．そして，2.2.3 小節にてネットワークアプライアンスにおける処理能力の制限について考察する．

2.2.1　ハードウェア構成

本小節では，ネットワークアプライアンスにおけるハードウェア構成と利用可能な計算機資源について説明する．

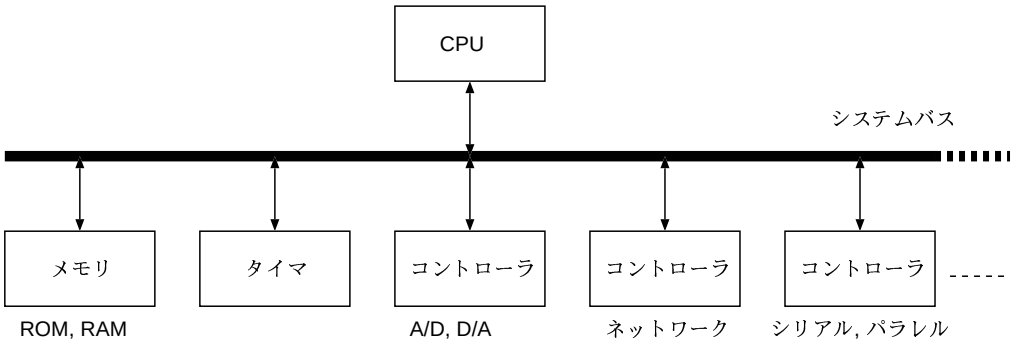

図 2.1: ネットワークアプライアンスのハードウェア構成

ネットワークアプライアンスにおける一般的なハードウェア構成を**図 2.1** に示す．

図の通り，システムバスを介して CPU やメモリ，各種コントローラが接続される．CPU はメモリに格納されたプログラムを処理する．メモリには ROM と RAM がある．ROM にはプログラムコードなど変化しないデータが格納され，RAM は一時的な記憶領域として利用される．コントローラには様々な種類があり，外部機器との入出力のために用いられる．例えば，Analog to Digital（A/D）変換器を用いてセンサからのアナログデータを取得したり，Ethernet コントローラを用いて外部との通信を行う．

ネットワークアプライアンスにおける計算機資源の目安を**図 2.1** に示す．[1] ．

表 2.1: ネットワークアプライアンスにおける計算機資源の目安

	メモリ	CPU 性能
PC	64MByte 以上	Pentium/64bits
PDA	2〜8MByte	RISC/32bit（50MHz 以下）
AV 機器	512KByte ROM	RISC/32bit（20MHz 以下），
	20〜64KByte RAM	8〜16bit MPU
センサ	512KByte ROM	8〜16bit MPU
	512KByte RAM	
白物家電	512KByte ROM	8〜16bit MPU
	16〜32KByte RAM	

本研究の検討対象であるネットワークアプライアンスは，表中の「AV 機器」「センサ」「白物家電」である．このような機器では厳しいコスト制限から必要最低限の計算機資源しか持たない場合が多く，PC などと比較して圧倒的に計算機資源が限られている．

2.2.2 ソフトウェア構成

本小節では，ネットワークアプライアンスにおけるソフトウェア構成を説明し，その代表例を挙げる．

ネットワークアプライアンスにおける一般的なソフトウェア構成を**図 2.2** に示す．

[1] 出典は，文献 [11]「Table 1. Resource restrictions of LCNA」.

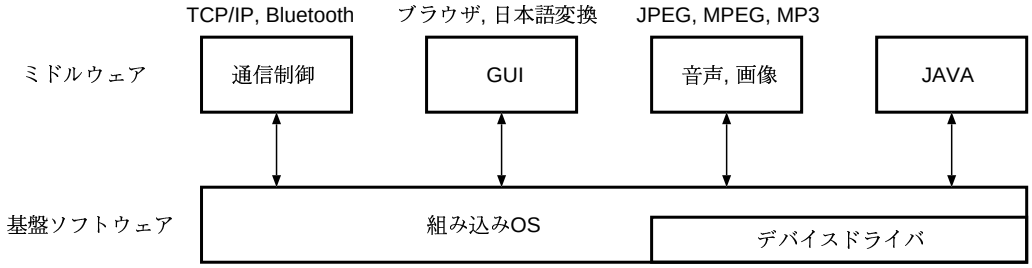

図 2.2: ネットワークアプライアンスのソフトウェア構成

図の通り，基盤ソフトウェアである組み込み OS 上で様々なミドルウェアが動作している．組み込み OS はその上で動作するアプリケーションのスケジューリングを行う．ミドルウェアは組み込み OS 上でアプリケーションとして動作したり，他のアプリケーションから呼び出される共有関数として利用される．

組み込み OS では µ ITRON[12] や組み込み Linux が著明である．ミドルウェアはシステムに様々な機能を提供する．例えば，TCP/IP スタックを導入すればネットワークに接続可能となり，JavaVM[13] を導入すれば JAVA プログラムを実行可能となる．また，音声 CODEC や動画および静止画 CODEC，暗号処理などもミドルウェアとして提供されることがある．

2.2.3 処理能力に関する考察

本小節では，ネットワークアプライアンスにおける処理能力の制限について，暗号処理を例に挙げて考察する．

2.2.1 小節にて説明した通り，ネットワークアプライアンスでは厳しいコスト制限から必要最低限の計算機資源しか持たない場合が多い．そのため，PC などと比較して圧倒的に計算機資源が限られている．例えば，文献 [14] では 20MHz で動作する 16bit CPU 上で，23KByte 程度の大きさの IPv6 スタックを動作させてセンサの制御を行っている．

通信の暗号化は計算負荷の高い処理であるため，このような環境では実用的な性能が得られない可能性がある．その一例として，文献 [14] と同様の環境において通信を暗号化した際の Round-Trip Time（RTT）を**表 2.2** に示す．なお，通信の暗号化には 5.1.1 小節にて説明する IPsec を用い，ping6 プログラム[2] を用いて 3 種類のペイロード長において RTT を測定した．

表 2.2: IPsec 使用時の ping6 RTT（ミリ秒）

暗号アルゴリズム	ペイロード長		
	64Byte	256Byte	1024Byte
No IPsec	4.26	9.06	28.28
DES[16]	24.36	80.71	304.59
3DES[17]	61.01	213.59	**824.26**
Rijndael[18]	13.36	36.81	130.48

実験対象: H8/3069F[19]@20MHz, Ethernet(10BaseT)

[2] ネットワーク到達性の確認や RTT を測定するためのツール．ICMPv6 Informational Messages[15] を使用する．

表 2.2 より，暗号アルゴリズムとして 3DES を用いた暗号化では，ペイロード長が 1024Byte の時に RTT が 0.8 秒を越えた．このように通信の遅延が増大すると，制御系ネットワークアプライアンスにおいてはリアルタイム性が損なわれ問題となる．また，パケットの送受信に大きな時間がかかると，通信量の多い AV 系ネットワークアプライアンスにおいてスループットの低下を招き問題となる．

このような場合，リアルタイム性を実現したり一定のスループットを確保するため，専用ハードウェアを利用することは古くから一般的であり，筆者らも暗号回路を利用したネットワークアプライアンスを実装してきた．そこで，2.3 節にて再構成可能なハードウェアを用いた専用ハードウェアについて説明する．

2.3 専用ハードウェアの利用

2.2.3 小節にて説明した通り，音声 CODEC や動画および静止画 CODEC，暗号処理などの高負荷な処理においては，リアルタイム性を実現したり一定のスループットを確保するために，専用ハードウェアの利用が現実的である．

本節では，まず，2.3.1 小節にて専用ハードウェアを用いる際のハードウェア構成を説明する．そして，2.3.2 小節にて再構成可能なハードウェアを用いたハードウェアの更新について説明する．

2.3.1 ハードウェア構成

本小節では，まず，専用ハードウェアを用いる際のハードウェア構成と代表的な実装デバイスを説明する．次に，専用ハードウェアの効果について考察する．

専用ハードウェアとして暗号回路を用いる際のハードウェア構成例を**図 2.3** に示す．

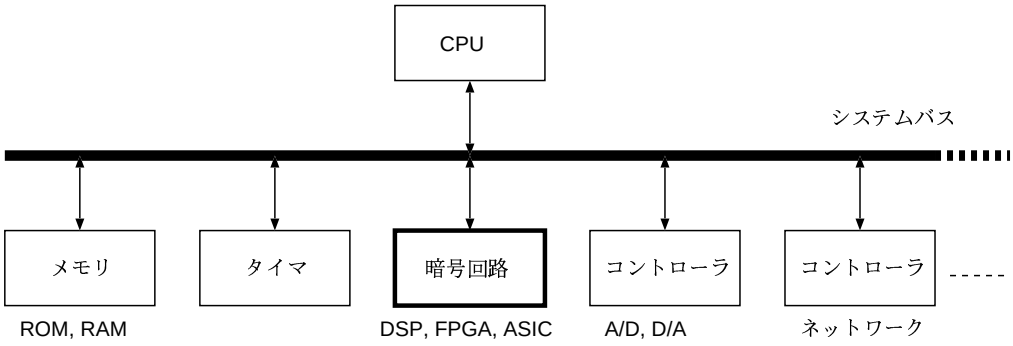

図 2.3: 専用ハードウェアを用いる際のハードウェア構成

図では，システムバスを介して CPU と専用ハードウェアである暗号回路が接続されている．システムバスを介した CPU と暗号回路間の通信は，コントローラを介すよりも高速である．

専用ハードウェアの実装デバイスには Digital Signal Processor（DSP）や Field Programmable Gate Array（FPGA），Application Specific Integrated Circuit （ASIC）などがあり，用途に応じて使い分けられる．

DSP はデジタル処理に最適化されたプロセッサであり，積和演算などデジタル信号処理で頻繁に実行される演算を高速に実行出来る．

FPGA は Programmable Logic Device（PLD）の一種であり，任意に論理を書き換えることが

出来る．SRAM セルベースの FPGA は何度も書き換えることが出来，本研究ではこのようなデバイスを**再構成可能なハードウェア**と定義する．FPGA は近年大容量化が著しく，FPGA 内に CPU や DSP を取り込むことも可能である．

ASIC はスタンダードセルなどを用いて設計される LSI である．FPGA と比べて，デバイスの単価は安く，高速動作かつ低消費出力を実現するが，莫大な開発コストがかかる上に開発期間は長い．FPGA と違い任意に論理を書き換えることは出来ない．

ネットワークアプライアンスにおける専用ハードウェアの効果を示すため，ソフトウェアで暗号処理する場合と，暗号回路を用いる場合のスループットを測定した．その結果を**表 2.3** に示す．実験対象 1 はネットワークアプライアンスを想定した環境にてソフトウェアで暗号処理を行う場合で，実験対象 2 は同様の環境にて暗号回路を利用する場合，実験対象 3 は一般的な PC にてソフトウェアで暗号処理を行う場合である．暗号回路は実装デバイスとして FPGA を用い，CPU と暗号回路は 8bit のシステムバスを介して接続される．なお，使用した暗号回路の詳細は 5.3 節にて説明する．

表 2.3: 暗号処理のスループット（kbps）

暗号アルゴリズム	実験対象 1	実験対象 2	実験対象 3
DES	**62.01**	**683.10**	32000
3DES	**21.15**	**683.10**	12800
Rijndael	210.18	714.36	48302

実験対象 1: H8/3069F 20MHz，ソフトウェアのみ
実験対象 2: H8/3069F 20MHz，暗号回路の利用
実験対象 3: Celeron 400MHz，ソフトウェアのみ

表 2.3 より，DES や 3DES を用いた場合，暗号回路の利用によって 10 倍以上スループットが向上した．このような専用ハードウェアの利用はホスト CPU に負荷をかけずに高負荷な処理を実現出来るため，リアルタイム性が求められるネットワークアプライアンスにおいて特に効果的である．

2.3.2 ハードウェアの更新技術

本小節では，まず，ハードウェアを更新する利点を説明し，既存の研究例を挙げる．そして，ハードウェアの更新が可能なデバイスとして，再構成可能なハードウェアの種類や仕組みを概説する．

2.3.1 小節にて説明した通り，専用ハードウェアの利用によって特定の処理を高速化出来る．しかし，同時にシステムの寿命を制限する可能性がある．例えば，暗号回路においては，暗号アルゴリズム自体の安全性が失われ，将来的にその回路が役に立たなくなる可能性がある．過去には，DES 暗号アルゴリズムが金融分野など幅広い用途で利用されていたが，DES 解読マシンの登場により安全ではなくなった．その後，DES を用いたシステムでは 3DES 暗号アルゴリズムへの移行が進み，さらに現在では Rijndael を用いるシステムも増えつつある．

長寿命が求められるネットワークアプライアンスにおいて，将来必要となるハードウェアを出荷前に予想することは難しい．現在，ソフトウェアにおいては不具合の修正や新機能の追加は当り前のように行われるが，専用ハードウェアにおいても同様のことが求められる．

ハードウェアの更新は，2.3.1 小節にて説明した再構成可能なハードウェアを用いることで実現出来る．再構成可能なハードウェアは後に論理を書き換えることが出来るので，例えば，利用中の暗号回路に不具合が発見されても速やかに不具合を修正出来る．このような研究は文献 [20] で行われており，複数の暗号アルゴリズムを積極的に変更可能な暗号処理ボードを提案している．

一度出荷されたネットワークアプライアンスの不具合を修正したり新機能を追加するには，新たな論理を対象ハードウェアに設定する必要がある．生活環境に深く埋め込まれたり数多く設置されることがあるネットワークアプライアンスにおいて，この作業を一台一台手動で行うことは難しい．そこで，ネットワークを介したハードウェアの自動更新の枠組みが提案されている．例えば，文献 [21] の Internet Reconfigurable Logic（IRL）では，設計者が遠隔からターゲットに新たな論理を設定する方法（PUSH 型）と，ターゲットが自動的に最新の論理をダウンロードする方法（PULL 型）を規定している．

このように再構成可能なハードウェアとネットワークを介した自動更新の枠組みを備えることで，専用ハードウェアを常に最新の状態に保つことが可能となり，システムの寿命を延ばすことが出来る．

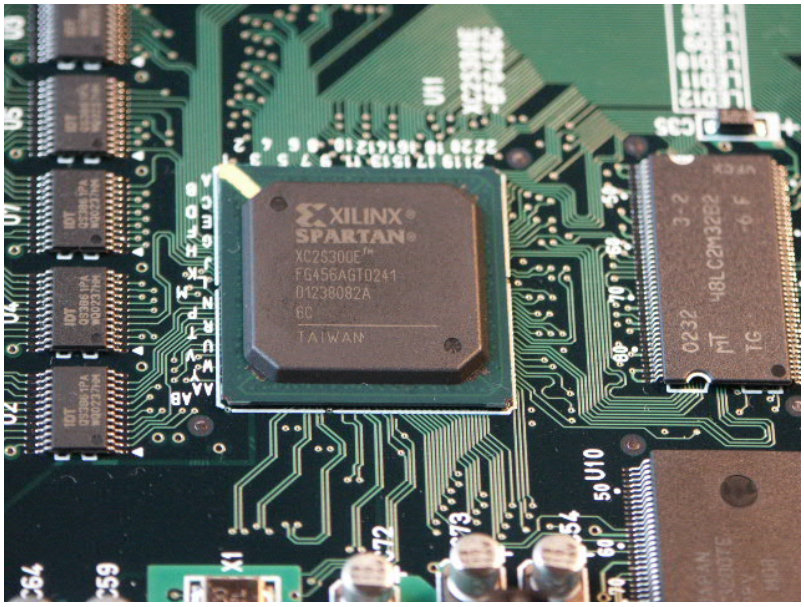

図 2.4: FPGA (Xilinx Spartan-IIE)

再構成可能なハードウェアの一種である FPGA を**図 2.4** に示す．

再構成可能なハードウェアには 2.3.1 小節にて挙げた FPGA の他にも，Complex Programmable Logic Device（CPLD），Simple ProgrammableLogic （SPLD）などがあり，用途に応じて使い分けられる．

FPGA の構造を**図 2.5** に示す．FPGA は 4 入力 1 出力などの Look Up Table（LUT）をベースにした複数の論理ブロックと配線リソースから構成される．プログラム素子は SRAM ベースものとアンチヒューズ技術[3] を用いたものがあり，SRAM ベースのものは再書き込み可能である．

CPLD は AND-OR ゲートをベースにした複数の論理ブロックとそれらを接続するための配線リソースから構成される．プログラム素子は EEPROM ベースのものが多く，同じく再書き込み可能である．

[3] 絶縁層を焼き切ることで信号間の結線を行う方式．

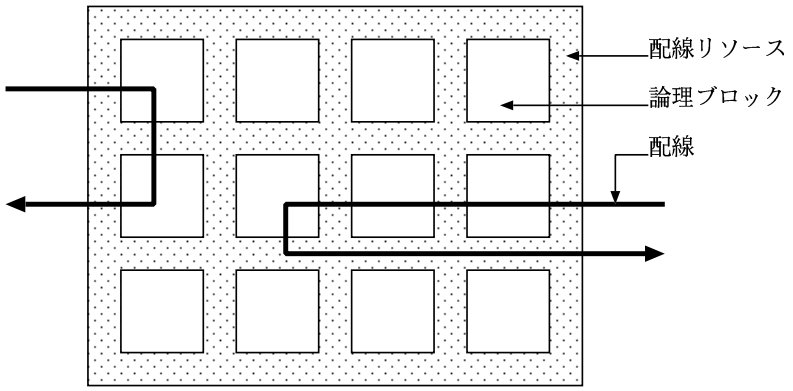

図 2.5: FPGA の構造

SPLD は同様の論理ブロックを 1 個搭載するデバイスである．

FPGA は CPLD や SPLD よりも大規模であるため，音声 CODEC や動画および静止画 CODEC，暗号処理などを実現するには FPGA が利用される場合が多い．よって，本研究では以降，再構成可能なハードウェアとして SRAM ベースの FPGA を用いる．

再構成可能なハードウェア上に所望の論理を実現するには，論理ブロック内の論理や論理ブロック間の配線方法などの情報（**構成データ**）をデバイスのプログラム素子に設定する必要がある．このように再構成可能なハードウェア上に新たな論理を実現することを本研究では**回路変更**と呼ぶ．

ネットワークアプライアンスのような民生機器においては，一般的にコストが重視される．実際，FPGA は ASIC よりもデバイス単価が高く，ASIC の試作用途でのみ用いられるケースが多かった．しかし，近年 FPGA の大規模化および低価格化が著しく，コストが重視される民生機器においても FPGA が徐々に採用されつつある．参考までに，FPGA の規模と価格の一例 [22] を **表 2.4** [4] に示す．

表 2.4: FPGA の規模と参考価格

FPGA の規模	価格
5/20/40 万ゲート	6.50 ドル以下
100 万ゲート	12 ドル以下

例えば，本研究にて用いた DES 回路は約 10 万ゲート，3DES 回路は約 16 万ゲート規模なので 6.50 ドル以下で提供可能な 20 万ゲート規模の FPGA に実装可能である．FPGA の大規模化および低価格化は今後も続く傾向にあるので，コスト制限が厳しいネットワークアプライアンスにおいて FPGA が利用可能な機会は増えつつあると言える．

2.4 現状の問題点

2.3.2 小節にて説明した通り，再構成可能なハードウェアとネットワークを介した自動更新の枠組みを備えることで，専用ハードウェアを常に最新の状態に保つことが出来る．よって，マルチ

[4] Xilinx Spartan-3 ファミリにおける 2004 年末での量産価格で 25 万個単位での単価.

メディアデータの取扱いや通信の暗号化などの処理を専用ハードウェアで実現しても，システムの寿命は犠牲にならなくなった．

しかしながら，

- 数多くのマルチメディアデータの符号化方式や暗号アルゴリズムが利用されている現在，実行する可能性のある処理全てをハードウェアで対応することは難しい．
- 当然ハードウェア化する処理を選択する必要が生じるが，コストを払い予めハードウェア化した処理が出荷後，頻繁に利用されるとは限らない．

例えば，MP3 の再生と通信の暗号化を行うシステムにおいて，ある利用形態では暗号処理のハードウェア化が効果的だが，別の利用形態では MP3 再生処理のハードウェア化が効果的かもしれない．

つまり，ソフトウェアでの処理を基本とするネットワークアプライアンスにおいて，設計者はハードウェア化する処理の選択を強いられ，利用者は利用形態に適した製品の選択を強いられる．この選択を誤ると専用ハードウェアの利用は逆にシステムの応用範囲を制限し兼ねないが，後の利用形態を予想することは利用者にとっても難しいのが現状である．

ネットワークアプライアンスにおいてハードウェア化が有効な処理を行う機会は増えつつあるので，第 3 章にて以上の問題を解決するためのモデルを検討する．

第3章　システムモデル

本章では，まず，3.1 節にて本研究が実現するモデルを説明し，3.2 節にてそのモデルを実現するためのアプローチを示す．そして，3.3 節にて関連研究を挙げる．

3.1　実現するモデル

2.4 節にて説明した通り，ネットワークアプライアンスの設計者はハードウェア化する処理の選択を強いられ，利用者は利用形態に適した製品の選択を強いられる．しかし，後の利用形態を予想することは利用者にとっても難しいのが現状である．

第 1 章にて説明した通り，ネットワークアプライアンスにおいては無設定かつ長寿命でなければならず，出荷後の利用形態に応じて柔軟に特徴を変更出来なければならない．そこで，本研究では，出荷後の利用形態に応じてハードウェア構成を自動調整するネットワークアプライアンスを目指す．そうすることで，ネットワークアプライアンスの不具合の修正や新機能の追加はもちろん，設計者および利用者が意識せずとも各ネットワークアプライアンスが利用形態に最も適したハードウェア構成に自動調整されるようになる．

出荷後の利用形態に応じてハードウェア構成を自動調整するネットワークアプライアンスを実現するために，以下の 2 点について明確にする．

(1) 利用形態に応じて処理をハードウェア化する方法.

(2) 選択可能な構成データを管理する方法.（選択可能な構成データを保持し，構成データを最新に保つこと）

(1) と (2) とも，それぞれいくつかのモデルが考えられる．本システムを実現するためには，本システムに適したモデルを選び組み合わせる必要がある．

そこで，3.1.1 小節にて (1) について，3.1.2 小節にて (2) についてそれぞれ検討する．なお，本節にて検討したモデルを実現するためのアプローチは 3.2 節にて示す.

3.1.1　ハードウェア更新に関するモデル

本研究では出荷後の利用形態に応じてハードウェア構成を自動調整するネットワークアプライアンスを実現する．また，そのために FPGA に代表される再構成可能なハードウェアを用いる．

利用形態に応じて処理をハードウェア化するには，以下に示すモデル (a) または (b) の適用が考えられる．

(a) 利用頻度の高い処理はハードウェアで実行するが，それ以外の処理はソフトウェアで実行する．

(b) 全ての処理をハードウェアで実行する．

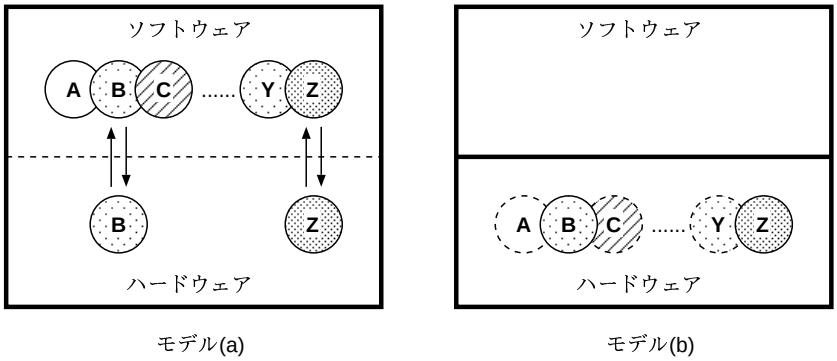

図 3.1: ハードウェア更新に関するモデル

モデル (a) および (b) の違いを**図 3.1** を用いて説明する．図では，モデル (a) および (b) が，処理 A〜Z をソフトウェアまたはハードウェアで実行している．

モデル (a) および (b) の処理の流れを以下にまとめる．

(a) 一定期間ごとに処理 A〜Z の中から利用頻度の高い処理を選ぶ．その処理がすでにハードウェア化されていれば何も実行せず，ハードウェア化されていなければハードウェア化する．処理 A〜Z を実行する際は，その処理がすでにハードウェア化されていればハードウェアで実行し，ハードウェア化されていなければソフトウェアで実行する．

(b) 利用頻度の高い処理を選んでハードウェア化することはない．処理 A〜Z を実行する際は，その処理がすでにハードウェア化されていればハードウェアで実行し，ハードウェア化されていなければハードウェア化してからハードウェアで実行する．

モデル (a) および (b) とも，利用頻度に偏りがあればハードウェア化，すなわち，回路変更の頻度は低くなる．

モデル (a) および (b) の得失を以下にまとめる．

(a) 利用頻度の高い処理を選ぶ間隔を長くとれば，回路変更の頻度は低くなる．つまり，回路変更の頻度を調節出来る．だたし，利用頻度の低い処理はソフトウェアでの実行となる．

(b) それぞれの処理を実行する度に回路変更が発生する可能性がある．つまり，回路変更の速度によってシステムの性能が左右される．ただし，全ての処理をハードウェアで実行出来る．

高速に回路変更可能なデバイスが利用出来れば，全ての処理をハードウェアで実行出来るモデル (b) が有利である．このようなデバイスは盛んに研究されているが，まだ一般に入手可能ではなく，切り換え可能な構成データの数や規模に制限がある．

2.3.2 小節に示したようなデバイスは一般に入手可能であり，回路変更は数ミリ秒オーダで実現出来ると言われている．しかし，CPU からデバイスに数 Mbit 程度の構成データを転送するため，CPU とデバイス間の帯域がボトルネックになることがある．実際に，ネットワークアプライアンスを想定した環境で回路変更の時間を測定したが，回路変更が終了するまでに 1 秒以上かかった．その間，処理は停止してしまうため，複数の処理を連続的に書き換えながら実行するモデル (b) での実現は難しいと言える．

一方，モデル (a) では上記した通り，回路変更の速度に合わせて回路変更の頻度を調節出来る．利用頻度の高い処理を選ぶ間隔を長くとれば，小さな利用形態の変化では回路変更は発生せず，回

路変更の発生を必要最低限まで絞り込むことが出来る．また，ハードウェア化に時間がかかったとしても，その間はソフトウェアにて処理を実行出来るので，一般に入手可能なデバイスを用いて回路変更に 1 秒かかったとしても大きな問題にはならない．よって，モデル (a) での実現はネットワークアプライアンスにおいて妥当であると言える．

そこで，本研究ではモデル (a)，すなわち，利用頻度の高い処理はハードウェアで実行するが，それ以外の処理はソフトウェアで実行するモデルを用いて，出荷後の利用形態に応じてハードウェア構成を自動調整するネットワークアプライアンスを実現する．

3.1.2 構成データ管理に関するモデル

3.1.1 小節にて示したモデルによって，出荷後，頻繁に利用される処理は常にハードウェア化される．よって，ネットワークアプライアンスの設計者および利用者が意識せずとも各ネットワークアプライアンスが出荷後の利用形態に適したハードウェア構成に自動調整されるようになる．

利用形態に適したハードウェア構成を選択可能にするには，ネットワークアプライアンスが選択可能な構成データを複数セット持つ必要がある．さらに，選択可能な構成データの種類が多いほど，選択可能なハードウェアのバリエーションは豊富になり，ネットワークアプライアンスはより利用形態に適した構成に近付く．

多数の構成データを選択可能とし，新機能の追加方法を持つには，以下に示すモデル (c) または (d) の適用が考えられる．

(c) 選択可能な構成データをネットワークアプライアンス自身が管理する．

(d) 選択可能な構成データをネットワーク上のサーバなど外部で管理する．

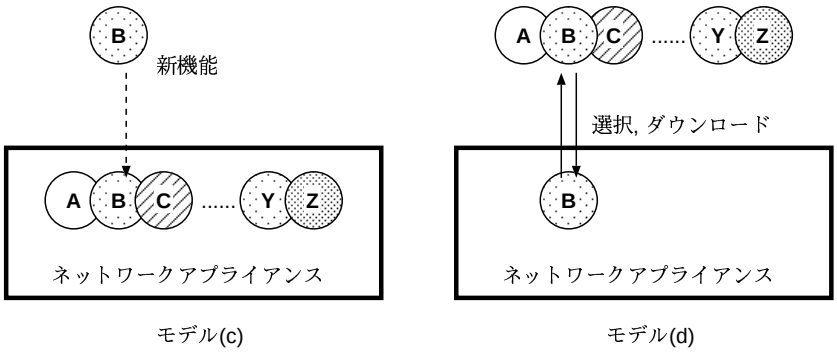

図 3.2: 構成データ管理に関するモデル

モデル (c) および (d) の違いを**図 3.2** を用いて説明する．図では，モデル (c) および (d) が，構成データ A〜Z の中から利用形態に適したものを選択しハードウェア化している．

モデル (c) および (d) の動作を以下にまとめる．

(c) 回路変更では，自身が管理する構成データ A〜Z の中から利用形態に適したものを選び，ハードウェア化する．新機能の追加では，外部から新たな構成データをダウンロードする．

(d) 回路変更では，ネットワーク上など外部で管理されている構成データ A〜Z の中から利用形態に適したものを選び，ダウンロードしてから，ハードウェア化する．新機能の追加では，外部で管理されている構成データを更新する．

モデル (c) および (d) の得失を以下にまとめる．

(c) ネットワークアプライアンス自身が多数の構成データを管理する必要がある．ただし，構成データはローカルにあるので，回路変更のオーバヘッドはモデル (d) よりも小さい．

(d) ネットワークアプライアンス自身が多数の構成データを管理する必要がない．ただし，回路変更の度に構成データをダウンロードする必要があるので，回路変更のオーバヘッドはモデル (c) よりも大きい．

ネットワークアプライアンスにおいては以下の 3 点の性質があるので，自身が多数の構成データを持つモデル (c) には問題がある．

(1) 構成データの種類の多さ

利用形態により適したハードウェア構成を実現するには，選択可能な構成データの種類を多くすることが効果的である．しかし，構成データ 1 つ分のサイズは数 Mbit 以上になる場合があるので，記憶領域が限られたネットワークアプライアンスにおいては保持出来る構成データの数は限られる．参考までに，FPGA の規模と構成データのサイズの一例 [23] を**表 3.1** に示す．

表 3.1: 構成データのサイズの例

FPGA の規模	構成データのサイズ
5 万ゲート	0.63Mbit
30 万ゲート	1.88Mbit
60 万ゲート	3.96Mbit

2.3.1 小節にて例示した通り，本研究にて用いた DES 回路は約 10 万ゲート，3DES 回路は約 16 万ゲート規模である．よって，それぞれを 30 万ゲート規模の FPGA に実装した場合，各暗号アルゴリズムごとに 1.88Mbit の記憶容量が必要となる．

そして，暗号アルゴリズムごとに別々の構成データを利用するなら，利用する暗号アルゴリズムの数だけ構成データが必要となる．例えば，IPsec や SSL V3.0[24] などの一般的な暗号通信プロトコルにおいては 10 種類近くの暗号アルゴリズムとハッシュアルゴリズム[1] が利用されている．

また，ネットワークアプライアンスによっては，さらに多くの構成データを保持する必要が生じる．例えば，暗号回路と同時に音声 CODEC や動画および静止画 CODEC などの構成データを選択可能にしたり，高速動作指向や省電力指向などの特徴を持った構成データを選択可能にする場合が考えられる．

(2) 新たな構成データへの対応

2.3.2 小節にて説明した通り，ネットワークアプライアンスにおいては出荷後の不具合の修正や新機能の追加は必要不可欠である．また，出荷時期を早めるために基本的な機能のみを備えた状態で出荷し，後に付加機能を追加する場合が考えられる．

[1] IPsec では DES，3DES，Blowfish，Cast128，Rijndael，SHA1，MD5 など．SSL V3.0 では DES，3DES，RC4，RC2，RSA，DSA，SHA1，MD5 など．

このように状況では，ネットワークアプライアンスが選択可能な構成データは出荷時に予め持っている構成データだけでなく，後に追加されるであろう構成データにも対応する必要がある．そのために予めより大きな記憶領域を確保しなければならない．

(3) 回路変更の頻度の低さ

本研究の検討対象であるネットワークアプライアンスは特定用途のために利用されるので，利用形態が頻繁に変化するとは考えにくい．

例えば，ネットワークアプライアンスに対してネットワーク機器の管理に用いられる Simple Network Management Protocol（SNMP）[25] のトラフィックが定期的に発生し，このトラフィックを通信相手ごとに異なる暗号アルゴリズムで暗号化したいとする．この場合，発生するトラフィックパターンは一定なので，使用される暗号アルゴリズムは変化しない．ずっと同じ暗号アルゴリズムがハードウェア化されていれば良いので回路変更する必要もない．

つまり，ネットワークアプライアンスの利用開始時などには回路変更が頻発するが，その後は処理内容に大きな変更がない限り，回路変更は発生しないと考えられる．

以上の 3 点をまとめると，ネットワークアプライアンスのように記憶領域が制限された環境では，利用可能な構成データの全てを持つことはコストが大きい．にもかかわらず，ネットワークアプライアンスにおいては利用形態が一定であり，ハードウェア構成を変更する機会は限られる．

そこで，本研究ではモデル (d)，すなわち，構成データをネットワーク上で管理し，ネットワークを介してハードウェアを更新するネットワークアプライアンスを実現する．

3.2 本研究のアプローチ

3.1.1 小節にて説明した通り，本研究では利用頻度の高い処理はハードウェアで実行するが，それ以外の処理はソフトウェアで実行する．また，3.1.2 小節にて説明した通り，本研究では構成データをネットワーク上で管理し，ネットワークを介してハードウェアを更新する．

以上のモデルを実現するためのアプローチを**図 3.3** を用いて説明する．本研究のアプローチでは，ネットワークアプライアンスが利用する構成データをネットワーク上のサーバにて管理する．そして，クライアントであるネットワークアプライアンスが利用形態の変化を検出すると，新たな利用形態に適した構成データをサーバからダウンロードしハードウェアを更新する．

このアプローチの利点は，各ネットワークアプライアンスが数 Mbit 以上にもなる構成データを複数セット管理する必要がない点と，不具合の修正や新機能の追加機能を兼ねる点である．一方，欠点は，回路変更の度にネットワーク上から構成データをダウンロードするため，構成データをローカルに保持する場合と比べ回路変更のオーバヘッドが大きくなる点である．しかし，ネットワークアプライアンスにおいては用途が限られているため利用形態の変化は少なく，さらに，利用頻度に基づいた回路変更によって回路変更の頻度を必要最小限に出来るので，この欠点は大きな問題にはならないと仮定した．

この仮定が正しいことを実証するために，第 6 章にて回路変更にかかるオーバヘッドの測定および利用頻度に基づいた回路変更の動作実験を行い，本システムによって回路変更の頻度を抑えつつ効率的に処理をハードウェア化出来るか確認する．

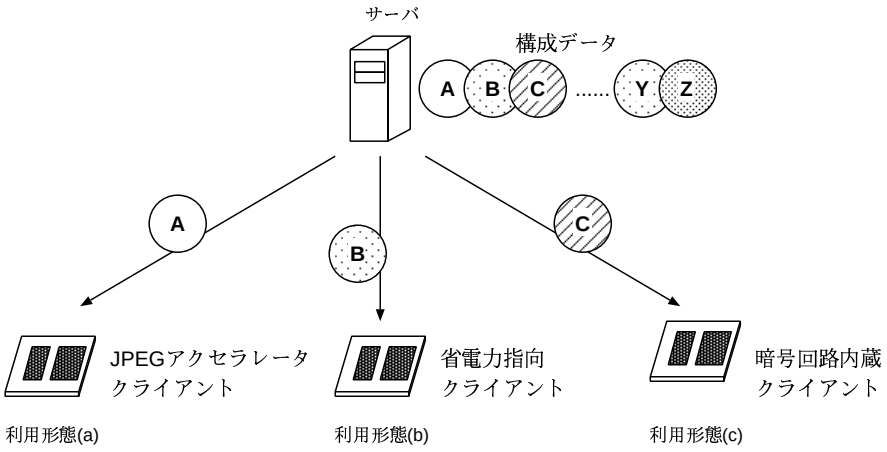

図 3.3: 本研究のアプローチ

3.3 関連研究

本節では，本研究の関連研究をまとめる．

暗号処理ボード SEBSW-2

文献 [20] では，複数の暗号アルゴリズムを積極的に変更可能な暗号処理ボードを提案している．第 5 章にて説明する本研究の実装例では，利用頻度が高い暗号アルゴリズムを自動的にハードウェア化する IPsec スタックを構築した．

本研究では，実装例として暗号処理におけるハードウェアの更新を取り扱っている．その点では同論文に類似しているが，本研究は暗号処理に特化したものではなく，処理ごとの利用頻度に基づいて自律的に回路変更を行う．

IRL（Internet Reconfigurable Logic）

文献 [21] では，ネットワークアプライアンスを対象に，ネットワークを介してハードウェアを更新するためのフレームワークを提案している．また，同システムを実現するためのミドルウェアを提供しており，一般的な組み込み OS 上で動作する．

IRL では，設計者が遠隔からターゲットのハードウェアを変更する方法（PUSH 型）と，ターゲットが自動的にハードウェアを最新のものに変更する方法（PULL 型）を規定している．特に，IRL の PULL 型は 3.1.2 小節にて説明したモデル (c) に近い．

本研究では，ネットワークアプライアンスがネットワークを介して自律的に回路変更する．その点では IRL の PULL 型に類似しているが，本研究はハードウェアを最新の状態に保つことだけが目的ではない．本研究では，処理ごとの利用頻度に基づいて回路変更を行うことで，利用形態に応じたハードウェアの自動調整を可能にする．

動的適応型ハードウェア

文献 [26] では，マルチコンテキストリコンフィギャラブルデバイスを用いて，状況に応じて最適の性能または消費電力を実現するモデルを提案している．

このモデルは，外部から構成データを設定可能なコンテキストをチップ内に複数持ち，かつ，これらのコンテキストを 1 クロックで切り替え可能なデバイス上での動作を想定している．このようなデバイスを対象に，全ての機能を実現出来るが低速だったり消費電が多いハードウェア A と，1 つの機能だけ実現するが高速だったり消費電力が少ないハードウェア A1，A2，...An を用意する．そして，実行時の性能をモニタしたり，経験をフィードバックする機能を持ったスケジューラがこれらのコンテキストを切り替える．

このように，同論文では高速にハードウェアを切り替え可能なデバイスを用いて複数のハードウェアを連続的に切り替えながら実行する．同論文は 3.1.1 小節にて説明したモデル (b) に近く，本研究の理想に近いと言える．

本研究では，ターゲットが実行時の性能や経験などの情報を基に自律的にハードウェアを切り替える．その点では同論文に類似しているが，本研究はネットワークアプライアンス用途を対象としていることから，特殊かつ高価なデバイスは使用出来ない．本研究では，一般的な再構成可能なハードウェアを用いて利用頻度の高い処理をハードウェアで，そうでない処理はソフトウェアで実行する．

動的機能変更可能な通信ネットワークノード

文献 [27] にて提案しているネットワークノードは，動的機能変更可能な複数のプロトコルプロセッシングモジュールを持ち，それらの負荷状態などに応じて通信パケットを振り分ける．

上記した通り，本研究の実装例として，利用頻度が高い暗号アルゴリズムを自動的にハードウェア化する IPsec スタックを構築した．

本研究の実装では，動的機能変更可能なプロセッシングモジュールに利用状況に応じて入力データを振り分ける．その点では同論文に類似するが，処理ごとの利用頻度に基づいて自律的に回路変更を行う．また，検討対象であるネットワークアプライアンスの特徴を考慮し，ネットワーク上で構成データを管理し，ネットワークを介してハードウェアを更新する．

モバイルシステムにおける動的再構成

文献 [28] では，バッテリが制限され様々なデータを扱うモバイルマルチメディアシステムを対象に機器の性能や機能を動的に変更する技術を提案している．

本研究では，同様の特徴を持ったネットワークアプライアンスを対象に機器の性能や機能を動的に変更する．その点では同論文に類似するが，本研究では，一般的な再構成可能なハードウェアを用いて利用頻度の高い処理をハードウェアで，そうでない処理はソフトウェアで実行する．また，回路変更の頻度を抑制する工夫がされている．

第4章　設計

本章では，本研究のアプローチであるネットワーク上での構成データの管理およびネットワークを介したハードウェアの自動更新を実現するためのシステムを設計する．そのために，4.1 節にてサーバおよびクライアントの機能要件を挙げる．4.2 節にてクライアントの機能である利用頻度に基づいた回路変更を設計する．

4.1　機能要件

図 4.1 を用いて，ネットワーク上で構成データを管理する**サーバ**と，ネットワークを介してハードウェアの自動更新を行う**クライアント**の役割りを説明する．

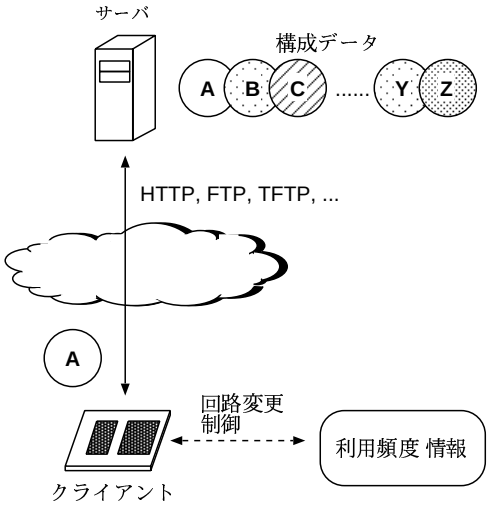

図 4.1: システム構成（サーバとクライアント）

図中のサーバはクライアントが使用する構成データ保持し，クライアントからの要求に応じて構成データを配信する．図中のクライアントは本研究の検討対象のネットワークアプライアンスである．FPGA に代表される再構成可能なハードウェアを持ち，自身に必要な回路の構成データをサーバから取得して，再構成可能部分に所望のハードウェアを実現する．

まず，クライアントの機能要件を以下にまとめる．

- **ネットワーク機能**
  構成データをダウンロードするために，HTTP，FTP，TFTP などサーバが使用するファイル転送プロトコルに対応する必要がある．これらの通信プロトコルは IPv4 や IPv6 の上で動作する．

- **セキュリティ機能**

信頼出来ない通信経路を用いて構成データをダウンロードする場合，不正な構成データの取得を防ぐために通信相手の認証が必要となる．そのためには，パスワードを使った認証から IPsec や SSL などの適用が考えられる．なお，IPsec は IPv6 における標準のセキュリティ機能となっている．

- **構成データ情報の管理**
  構成データをダウンロードするため，使用する通信プロトコル，構成データを保持するサーバのホスト名，構成データのファイル名，構成データのバージョンなどの情報を管理する必要がある．

- **再構成可能部分の制御**
  FPGA に代表される再構成可能部分を持ち，これらを書き換える機能が必要である．書き換え方法は使用するデバイスによって異なる．

- **回路変更のタイミング制御**
  回路変更の必要性を判断し，回路変更の頻度を調節する．この判断には，処理ごとの利用頻度（単位時間当たりに処理が実行された回数）に処理ごとの重みを掛け合わせた値を用いる．重みは設計者が設定し，場合によっては利用者が調整可能とする．

クライアントの機能要件に関して，ネットワーク機能およびセキュリティ機能は既存のプロトコルやソフトウェアの利用によって実現出来る．構成データ情報の管理および再構成可能部分の制御，回路変更のタイミング制御は，4.2 節にて利用頻度に基づいた回路変更として具体的に設計する．

次に，サーバの機能要件を以下にまとめる．

- **ネットワーク機能**
  構成データを配信するために，HTTP，FTP，TFTP などクライアントが使用するファイル転送プロトコルに対応する必要がある．

- **セキュリティ機能**
  パスワードを使った認証や IPsec，SSL などのセキュリティ機能に対応する必要がある．

- **構成データの管理**
  構成データを記憶したり，新たな構成データの追加や削除，更新などを行う必要がある．これらは既存の HTTP サーバや FTP サーバにおけるコンテンツ管理と同じである．

- **規模性**
  多数のクライアントからアクセスされる場合，1 台のサーバだけでは処理が追い付かない可能性がある．場合によっては，複数のサーバにトラフィックを分散させる必要がある．これらは既存の HTTP サーバにおける負荷分散技術を流用することが出来る．

サーバの機能要件に関しても，ネットワーク機能およびセキュリティ機能は既存のプロトコルやソフトウェアの利用によって実現出来る．また，構成データの管理は HTTP サーバや FTP サーバにおけるコンテンツ管理と同じであり，規模性も HTTP サーバにおける負荷分散技術などの技術を流用出来る．

サーバの設置場所はクライアントへのサービス形態によって異なる．例えば，クライアントの製造メーカがクライアントに構成データを配信する場合，サーバは製造メーカ内に設置される．ま

た，利用者自身がクライアントに構成データを配信する場合，サーバはホームサーバなどとして利用環境内に設置される．

4.2 利用頻度に基づいた回路変更

回路変更の必要性を判断したり回路変更の頻度を調節するために，回路変更のタイミングを制御する必要があり，そのために本システムでは利用頻度に基づいた回路変更を行う．

本システムでは全ての処理をソフトウェアで実行可能とするが，単位時間当たりの利用頻度が高い処理は再構成可能部分を書き換えハードウェアで実行可能にする．ハードウェア化する処理でも，利用頻度が低いとハードウェア化せずにソフトウェアで実行するため，ハードウェア化する処理は必ずソフトウェアでも実行可能でなければならない．このような処理を本研究では**ハードウェア化可能な処理**と呼ぶ．

本節では，利用頻度に基づいた回路変更について，4.2.1 小節にてそのアルゴリズム，4.2.2 小節にてそのデータ構造を説明する．

4.2.1 アルゴリズム

図 **4.2** は利用頻度に基づいた回路変更のモジュール相関図である．本システムは**カウンタ部**，**コントロール部**，**書き換え部**から成り立っている．図では，利用頻度に基づいた回路変更によってハードウェア化可能な処理 A〜C が制御されている．各モジュールの動作を以下にまとめる．

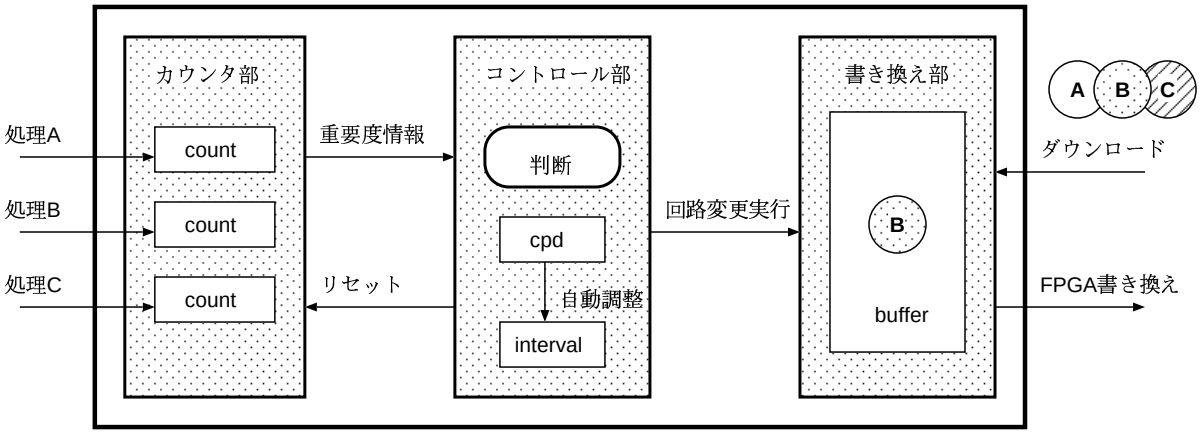

図 4.2: モジュール相関図

カウンタ部

カウンタ部の動作を**図 4.3** を用いて説明する．処理 A〜C はハードウェア化可能な処理であり，図中のプログラムの流れの通り処理 A〜C が繰り返し実行されている．

処理 A〜C をハードウェアまたはソフトウェアで実行する度に，それぞれの**利用回数**をカウントアップする．

処理ごとに予め**重み**が設定されており，利用回数に重みを掛けた数値をその処理の**重要度**とする．そして，単位時間当たり最も重要度が高くなった処理がハードウェア化の候補となる．例え

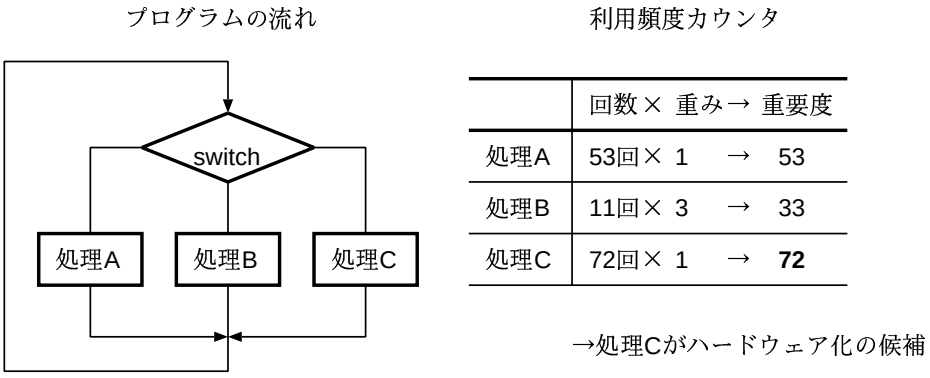

図 4.3: カウンタ部

ば，図 4.3 では重要度が 72 の処理 C がハードウェア化の候補である．

単位時間当たり最も重要度が高くなった処理をハードウェア化するので，それぞれの利用回数は単位時間ごとにリセットする．

処理ごとの重みは，処理をソフトウェアで実行した際の所要時間を基にプログラム設計者が決める．例えば，3DES 暗号アルゴリズムは DES の暗号化または復号化を 3 回繰り返すものであり，ソフトウェアで 3DES を処理すると DES よりも 3 倍近く時間がかかる．この場合，DES の重みを 1 とし 3DES の重みを 3 とすると，処理の重い 3DES が優先的にハードウェア化される．

また，リアルタイム性が求められる処理には重みを大きく設定したり，ユーザによって重みを調整可能にすることも出来る．

コントロール部

コントロール部は**単位時間**ごとに周期的に呼び出され，カウンタ部にアクセスして最も重要度が高い処理を調べる．最も重要度が高い処理が現在ハードウェア化されてなければ，書き換え部に回路変更を要求する．ハードウェア化が完了すると，その処理の**状態**を「ハードウェア化済」に変更する．コントロール部の処理が終了すると，カウンタ部にアクセスしてそれぞれの利用回数をリセットする．

コントロール部が呼び出されても，ハードウェア化する処理がすでに再構成可能部分に実現されていれば回路変更は発生しない．よって，同じ処理がハードウェア化され続ければ回路変更は発生しない．本システムでは回路変更は負荷の高い処理であり，回路変更が頻発することは望ましくないので，回路変更が頻発しないようにする必要がある．

コントロール部では，回路変更の頻度情報として**1 日当たりの回路変更回数**（Change Per Day: CPD）を管理する．回路変更の頻度を抑えるには，利用頻度をチェックする間隔（単位時間）を長くすることが効果的である．例えば，単位時間を 600 秒にすれば CPD は 144 以下になり，単位時間を 3600 秒にすれば CPD は 24 以下になる．

本システムでは，予め **CPD の上限値**を決めておき，CPD がその上限値を上回ったら単位時間を 2 倍にする．処理の流れを以下に示す．

```
if (CPD > CPD の上限値) {
        単位時間 = 単位時間 * 2;
}
```

そうすることで，CPD が上限値以下になるように単位時間が調節される．なお，CPD の上限値は設計者が設定する．

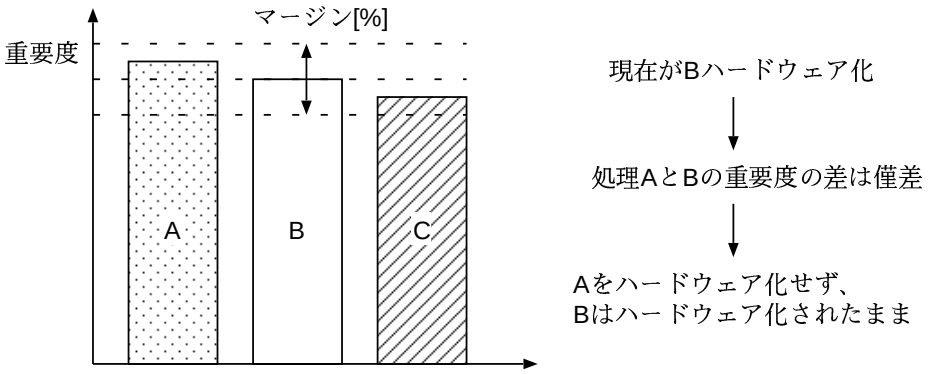

図 4.4: 重要度のマージン

効果の小さな回路変更を抑制するため，**重要度にマージン**を設ける．**図 4.4** を用いて重要度のマージンについて説明する．現在，処理 B がハードウェア化されており，現在の処理 A の重要度が 103，処理 B の重要度が 100 だとする．当然，処理 A がハードウェア化されるが，この場合，2 つの処理の重要度は僅差であるため回路変更の効果は大きくなく，次の回路変更でまた順位が入れ替わる可能性もある．本システムでは回路変更のオーバヘッドが大きいので，回路変更を安定させる必要がある．そこで，処理 A は現在ハードウェア化されている処理 B よりも一定の割合以上大きな重要度にならないと回路変更しないようにする．処理の流れを以下に示す．

```
if ((ハードウェア化候補の重要度 - ソフトウェア化候補の重要度) > マージン) {
        ハードウェア化候補をハードウェア化する
}
```

なお，重要度のマージンは設計者が設定する．

書き換え部

コントロール部によって回路変更する必要があると判断された場合，書き換え部が呼び出され回路変更を実行する．

書き換え部が回路変更を実行するために，処理ごとに以下の情報を管理する．

- 構成データを保持するサーバのホスト名
- 構成データのファイル名
- 使用する通信プロトコル

これらの情報を基に，サーバから構成データをダウンロードする．ダウンロードした構成データは一度バッファに格納してから再構成可能部分に転送する．

バッファが構成データのサイズより大きければ，頻繁に利用する構成データを保持するキャッシュとして働く．逆に，構成データのサイズより小さければ，構成データを分割してダウンロードし再構成可能部分を書き換える．

再構成可能部分の書き換えはシステムに依存した処理であり，使用する再構成可能なハードウェアの種類および CPU との接続形態によってその手順は異なる．例えば，図 2.4 に示した Xilinx Spartan-IIE には，構成データをシリアルで転送する方法やパラレルで転送する方法などが用意されており，状況に応じて使い分ける．

4.2.2 データ構造

利用頻度に基づいた回路変更にて利用するデータ構造を**図 4.5** に示す．

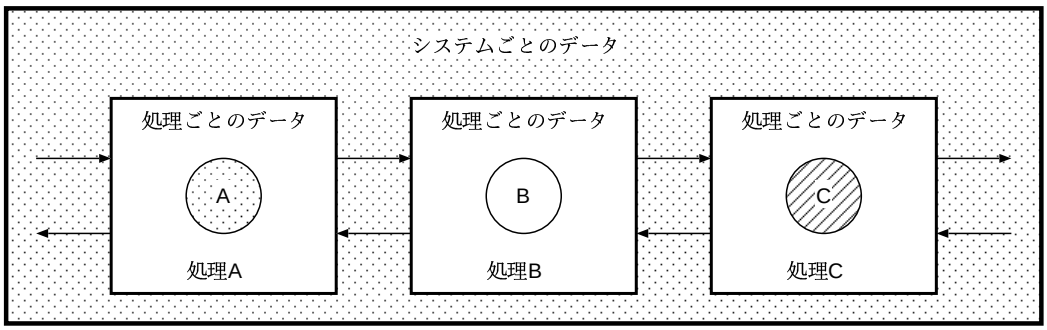

図 4.5: データ構造

クライアントは，大きく分けて 2 種類のデータ構造を管理する．図中の**処理ごとのデータ**はハードウェア化可能な処理ごとの情報を，図中の**システムごとのデータ**はハードウェア化可能な処理全てに共通の情報を管理する．

処理ごとのデータ

処理ごとのデータはハードウェア化可能な処理の数だけ存在する．最低限必要なデータを**表 4.1** に示す．

表 4.1: 処理ごとのデータ

名前	説明
id	固有な識別子
count	処理の利用回数
weight	処理ごとに定められた重み
importance	処理の重要度
file	構成データの在処
status	処理の状態（ハードウェア化済みなど）

id はシステム内のハードウェア化可能な処理を一意に識別する．

count に weight を掛け合わせた数値が importance であり，単位時間当たり最も importance が大きい処理がハードウェア化の候補となる．weight はハードウェア化の優先度であり，設計者が処理をソフトウェアで実行する際の時間を基に設定する．

file は構成データの在処を表し，使用する通信プロトコル名（protocol），構成データを保持するホスト名（hostname），ファイル名（filename）を「protocol://hostname/filename」の形式で記憶する．file は設計者が設定する．構成データの在処を変更する場合は，リンクを張るなどサーバ側で対応する．

status はその処理がハードウェア上に実現されているかどうかの情報を保持する．例えば，処理 A を実行する際，プログラムは必ず処理 A の status を確認する．処理 A がすでにハードウェア化されていれば status はハードウェア化済みとなっており，処理 A をハードウェアで実行する．そうでなければ処理 A をソフトウェアで実行する．

システムごとのデータ

システムごとのデータはハードウェア化可能な処理全てに共通の情報を管理するため，システムに 1 つだけ存在する．最低限必要なデータを**表 4.2** に示す．

表 4.2: システムごとのデータ

名前	説明
entries	処理ごとのデータのリスト
interval	単位時間（利用頻度をチェックする時間間隔）
cpd	1 日当たりの回路変更回数
max_cpd	cpd の上限値
margin	重要度のマージン

entries は表 4.1 に示す処理ごとのデータをリスト状に連結したものである．最も重要度が高い処理を選出する時など，ハードウェア化可能な処理全てにアクセスする時はこのリストを辿ることになる．

回路変更では単位時間当たりの重要度が高い処理をハードウェア化するが，interval はこの単位時間を表し，単位は秒である．

interval ごとの回路変更の判断において，すでにその処理がハードウェア化されていれば回路変更は発生しない．cpd は実際に回路変更が発生した割合として 1 日当たりの回路変更回数を表し，この cpd が高いと回路変更が頻繁に発生したことになる．

本システムでは，cpd の値が上限値である max_cpd を越えると interval を 2 倍にし，回路変更の頻度を小さくする．cpd_max は設計者が設定し，機器によってはユーザが変更可能にする．

margin は効果の小さな回路変更を抑制するための重要度のマージンで，単位は%である．margin も設計者が設定し，機器によってはユーザが変更可能にする．

第5章　実装

第 4 章にて設計したシステムの実用例として，利用頻度が高い暗号アルゴリズムを自動的にハードウェア化する IPsec スタック（**IPsec Switching**）を実装した．IPsec では複数の暗号アルゴリズムが利用されており，同一システム上で複数の高負荷な処理を実行する典型的な例である．本章では，まず，5.1 節にて IPsec Switching の概要を述べる．そして，IPsec Switching の実装として，5.2 節にてサーバの構築，5.3 節にて暗号回路の実装，5.4 節にてクライアントの実装について説明する．

5.1　IPsec Switching の概要

まず，5.1.1 小節にて既存の暗号化通信プロトコルを挙げ，その一つである IPsec について説明する．そして，5.1.2 小節にて IPsec Switching の動作概要を説明する．

5.1.1　通信の暗号化

ネットワークアプライアンスが本格的に人々の生活に浸透するには，プライバシや安全性の面から考えて，十分なセキュリティ機能が必須と言える．機器の利用者が被る可能性のある攻撃として，リプレイ攻撃など機器の所有者になりすまして機器を不正に操作する行為や通信内容の盗聴や改竄，サービス不能（DOS）攻撃などが挙げられる．これらの攻撃を防ぐ一般的な方法として通信の暗号化および認証がある．そのための通信プロトコルとして IPsec や Secure Socket Layer（SSL），SSL を拡張した Transport Layer Security（TLS）[29] などが提案されており，社内 VPN の構築や電子商取り引きなどで利用されている．

本システムの実用例である IPsec Switching は暗号化通信プロトコルとして IPsec を用いる．

IPsec はネットワーク層にて通信の暗号化および認証などのセキュリティ機能を実現する．上位層である TCP や UDP，アプリケーションが IPsec の存在を意識する必要はない．

通信の暗号化では，DES や 3DES，Rijndael などの共有鍵方式の暗号アルゴリズムが利用される．パケットを送信する際に，予め決めておいた暗号アルゴリズムおよび鍵を用いてペイロードを暗号化する．パケットを受信する際に，同様の方法でペイロードを復号化する．なお，暗号化されたペイロードは，Encapsulating Security Payload（ESP）と呼ばれる拡張ヘッダに格納される．

認証では，パケットの完全性保証と送信元の本人性確認を実現する．パケットを送信する際に，HMAC-SHA1 や HMAC-MD5 などの鍵付きハッシュ関数を用いてパケット全体のハッシュ値を計算し，その値を IP パケットに付与する．パケットを受信する際に，同様の方法でハッシュ値の再計算を行い，IP パケットに付与されたハッシュ値と一致するか確認する．なお，IP パケットに付与されるハッシュ値などの情報は，Authentication Header（AH）と呼ばれる拡張ヘッダに格納される．

IPsec は特定の暗号アルゴリズムおよびハッシュアルゴリズムに依存していない．この特徴は，特定のアルゴリズムに弱点が発見された際のリスクを軽減出来るので重要である．IPsec では，

Security Policy Database（SPD）で指定したパケットに対し，Security Association Database（SAD）で指定した暗号アルゴリズムや鍵を用いて通信の暗号化および認証を行う．SPD ではパケットの送信元アドレスや受信先アドレス，ポート番号などを指定出来るので，通信相手や通信内容に応じて異なった暗号アルゴリズムを利用出来る．つまり，IPsec は同一システム上で複数種類の高負荷な処理を実行する典型的な例であると言える．

IPsec の動作モードには Transport モードと Tunnel モードがある．Transport モードは，ホスト同士の暗号化および認証を実現する．Tunnel モードは，ネットワーク同士の暗号化および認証を実現する．後者では，各ホストは IPsec の処理を行わず，それぞれのネットワークの出口に当たるゲートウェイが IPsec の処理をまとめて行う．

5.1.2 動作概要

5.1.1 小節にて説明した通り，IPsec は特定の暗号アルゴリズムに依存していないので，複数の通信相手が全て同じアルゴリズムを使用するとは限らない．

想定する IPsec Switching の利用環境を**図 5.1** に示す．図では，本システムは通信相手 (a)〜(c) と IPsec を用いて通信しており，通信相手 (a) とは Rijndael，通信相手 (b) とは DES，通信相手 (c) とは 3DES を用いて通信の暗号化を行う．この利用環境は，通信の暗号化を行うネットワークアプライアンスが増え，ネットワークアプライアンス同士が自律的に協調動作する環境を想定している．

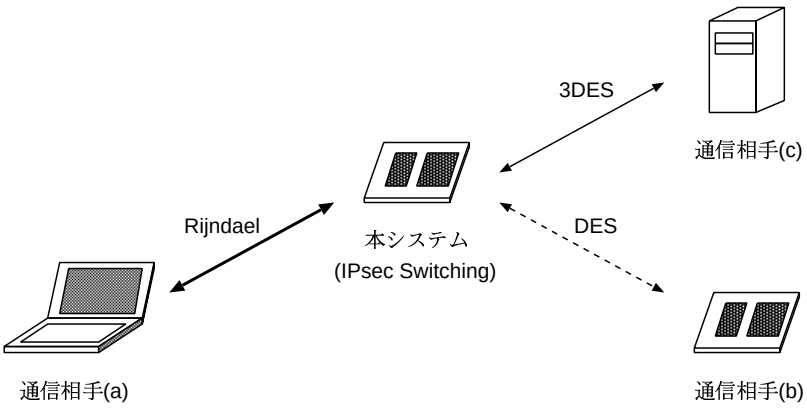

図 5.1: IPsec Switching の利用環境

このような環境では，後にどのような暗号アルゴリズムが頻繁に利用されるか予想出来ない上に，暗号アルゴリズムが新たに必要または使用不可になる可能性がある．暗号アルゴリズムを効率的にハードウェア化するには，各ネットワークアプライアンスがそれぞれの利用形態に最も適したハードウェア構成を自律的に判断する必要があり，本システムが必要になる．

本システムでは，複数の暗号アルゴリズムをソフトウェアで実行可能とするが，頻繁に利用された暗号アルゴリズムはハードウェアでも実行可能にする．例えば，通信相手 (a) との通信量が多く頻繁に Rijndael が利用されれば，自動的に Rijndael がハードウェア化される．

本システムによって，常に適切な暗号アルゴリズムがハードウェア化されるので，予めハードウェア化される暗号アルゴリズムが固定されているシステムと比べ，通信性能は向上する．

5.2 サーバの構築

4.1 節にて説明した通り，サーバの機能に関しては既存のプロトコルおよびソフトウェア，さらに，コンテンツ管理や負荷分散技術など既存の技術を流用出来る．本節では，構成データの配信サーバを構築する上で利用したプロトコルやソフトウェアを説明する．

本研究では，ネットワーク層プロトコルとして IPv6 および IPsec を用いた．IPv6 には広大なアドレス空間とアドレス自動設定機能があり，これらの特徴はネットワークアプライアンスにおいて有効である．また，通信の暗号化および認証には，IPv6 における標準のセキュリティ機能である IPsec を選択した．

アプリケーション層プロトコルとして Trivial File Transfer Protocol（TFTP）[30] を用いた．TFTP は UDP 上で動作するストップアンドウェイト型のファイル転送プロトコルである．プロトコルが単純なため，ディクレスシステムや組み込み機器など，計算機資源が限られた環境で用いられることが多い．本システムでも，実現が容易という理由から TFTP を選択した．

サーバに使用したマシンは一般的な PC であり，CPU は MMX Pentium 166MHz，メモリは 64MByte，通信メディアは Ethernet（10Base-T），OS は Linux（2.4.20）である．実験運用のため，サーバはクライアントは同一リンクに設置した．

記憶領域にはクライアントが使用する暗号回路の構成データファイルを置く．ネットワークアプライアンス上が TFTP クライアントとなり，構成データをダウンロードする．クライアントが使用する暗号回路は 5.3 節にて説明する．

5.3 暗号回路の実装

IPsec Switching の動作概要にて説明した通り，本システムでは DES，3DES，Rijndael 暗号アルゴリズムをハードウェアまたはソフトウェアで実行する．本小節では，本システムで使用する DES，3DES，Rijndael ハードウェアの実装を説明する．

実装した暗号回路の構成を**図 5.2** に示す．例として DES を用いるが，他の暗号アルゴリズムも構成はほぼ同じである．

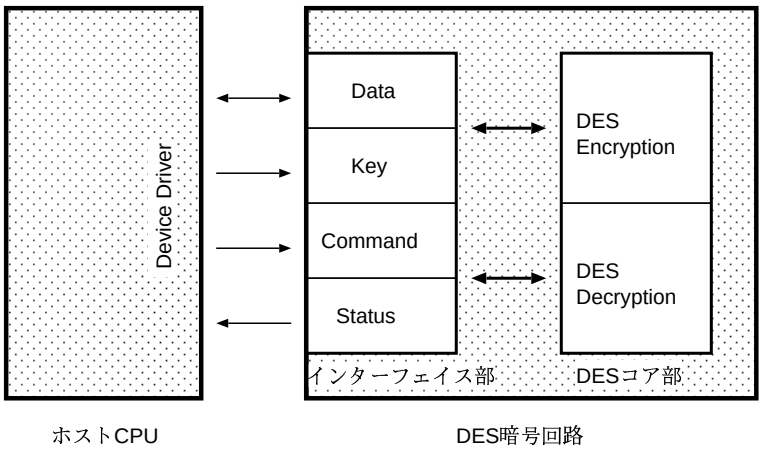

図 5.2: 暗号回路の構成

図中の DES 暗号回路は，DES コア部とインターフェイス部から構成される．

DES コア部は，DES の暗号化および復号化処理を行う．これらは OPENCORES[31] にて公開

されている実装が基になっている．

インターフェイス部は，DES コア部に対して暗号化または復号化を命令する．インターフェイス部は，システムバスを介してホスト CPU に接続され，ホスト CPU と DES コア部の両方からアクセスされるレジスタを持つ．レジスタには，鍵や入出力データ，DES コア部のステータスが保存される．

DES 暗号回路を用いた暗号化の流れを順に説明する．

(1) ホスト CPU が，Key レジスタに 64bit の共有鍵，Data レジスタに 64bit の平文を書き込み，Command レジスタの暗号化実行フラグを立てる．

(2) インターフェイス部が，Key レジスタ，Data レジスタの内容を DES コア部に転送し，暗号化を命令する．

(3) DES コア部が，暗号化を行う．

(4) インターフェイス部が，DES コア部から暗号文を読み出し，Data レジスタに書き込み，Status レジスタの DONE フラグを立てる．

(5) ホスト CPU が，Status レジスタの DONE フラグを確認し，Data レジスタの内容を読み出す．

このような動作をする DES，3DES，Rijndael 回路を Xilinx Spartan-IIE （xc2s300e-6fg456）上に実装した．なお，開発環境は Xilinx ISE WebPACK 5.1i[32]，開発言語は Verilog-HDL を使用した．ISE WebPACK により出力される.bin ファイルが構成データファイルとなり，サーバに置かれる．

実装した暗号アルゴリズム 3 種類のハードウェアにおけるスライス[1] 使用率と最高動作周波数を**表 5.1** に示す．なお，Rijndael に関しては，実装デバイスの資源量をオーバーしたため，暗号回路と復号回路を一度に実装出来ず，暗号回路と復号回路を別々に実装した．

表 5.1: 実装した各暗号回路の諸元

	スライス使用率	最高動作周波数
DES	27%	65.26MHz
3DES	55%	48.00MHz
Rijndael（暗号化）	69%	71.19MHz
Rijndael（復号化）	84%	56.56MHz

FPGA: Spartan-IIE（30 万ゲート規模）
論理合成オプション: 速度優先

5.4 クライアントの実装

本小節では，クライアント，すなわち，本研究の検討対象であるネットワークアプライアンスの実装を説明する．5.4.1 小節にてクライアントのハードウェア構成を説明し，5.4.2 小節にてその上で動作するソフトウェアについて説明をする．

[1] Spartan-IIE の場合，1 論理ブロック中にスライスが 2 つあり，1 スライス中に LUT が 2 つある．

5.4.1 ハードウェア構成

クライアントのハードウェア構成を**図 5.3** に示す.

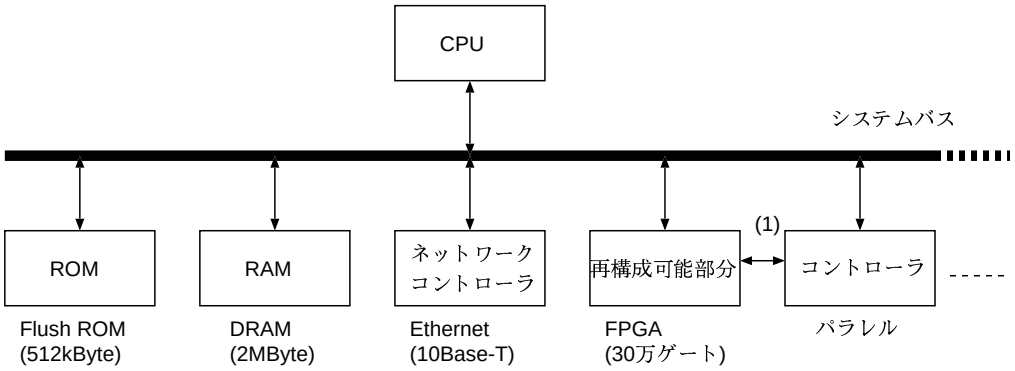

図 5.3: クライアントのハードウェア構成

クライアントではアプリケーションとして IPsec が動作する．図の通りネットワークインターフェイスを持ち，外部との通信を行う．

本システムでは，IPsec の性能を向上させるために再構成可能なハードウェアを持ち，頻繁に利用される暗号アルゴリズムをハードウェア化する．再構成可能なハードウェアはシステムバスを介してアクセスされ，パラレルコントローラを介して書き換えられる（図中の (1)）．ネットワークインターフェイスは，サーバから所望の構成データをダウンロードするためにも用いられる．

ハードウェア構成は，ネットワークアプライアンス用途を想定して選定した．クライアントを構成する主な部品を**表 5.2** に示す．ネットワークアプライアンス用途では特殊かつ高価な部品は使用されにくいので，本実装でも容易に入手可能な部品のみを利用した．

表 5.2: クライアントの実装に用いた主な部品

用途	型番
CPU	Hitachi H8/3069F
ネットワークコントローラ	Realtek RTL8019AS
DRAM（2MByte）	Hitachi HM5117800
FPGA（30 万ゲート規模）	Xilinx Spartan-IIE

再構成可能なハードウェアとして用いた Spartan-IIE は，書き換え方式としてバウンダリスキャンモード [33]，スレーブシリアルモードおよびスレーブパラレルモード [34] などに対応している．その中でも，スレーブパラレルモードは 8bit 単位つまりパラレルで構成データを転送出来る唯一の方法であり，最も高速に回路を書き換えることが出来る．よって，本システムでは FPGA の書き換え方式としてスレーブパラレルモードを用いる．

実装したクライアントの外観を**図 5.4** に示す．本実装はプロトタイプ実装であり，縦 30cm 横 20cm 高さ 5cm のケースに格納する大きさとなった．

図 5.4: プロトタイプ実装の外観

5.4.2 ソフトウェア構成

クライアントにおけるソフトウェアの開発には，GNU Development Tools for the Renesas H8/300[HS] Series[35] を使用した．

クライアントのソフトウェア構成を**図 5.5** に示す．

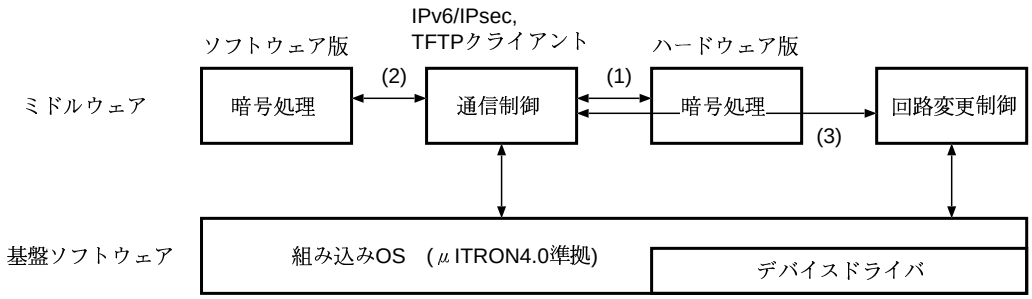

図 5.5: クライアントのソフトウェア構成

クライアントでは適用例である IPsec（通信制御）の性能を向上させるために，回路変更制御を行い利用頻度の高い暗号アルゴリズムをハードウェア化する．IPsec における暗号処理では，利用したい暗号アルゴリズムがハードウェア化されていれば，ハードウェア版の暗号処理関数を呼び出し（図中の (1)），ハードウェア化されていなければ，ソフトウェア版の暗号処理関数を呼び出す（図中の (2)）．また，回路変更を実行する際に，TFTP クライアント（通信制御）が呼び出される（図中の (3)）．

以降，クライアントの実装について，基盤ソフトウェア，通信制御，暗号処理，回路変更制御に分けて説明する．

基盤ソフトウェア

組み込み OS は，通信制御や回路変更制御などのアプリケーションを実現するために必要である．組み込み OS には様々な種類があるが，組み込み分野において採用実績が豊富であるという理由から，μ ITRON4.0 の仕様に準拠したリアルタイム OS を実装した．

実装したリアルタイム OS が提供する機能には，タスク管理（タスクの生成や起動），タスク間通信（セマフォやメールボックス），メモリ管理（固定長メモリプール），時間管理（システム時刻や周期起動ハンドラ），割り込み管理などがあり，合計 79 種類のシステムコールが利用可能である．

通信制御

通信制御として，上記のリアルタイム OS 上で動作する Ethernet ドライバ，IPv6 スタック，IPsec スタック，TFTP クライアントを実装した．

実装した IPv6 スタックはトランスポート層として UDP に対応し，アドレス自動設定に利用する Neighbor Discovery Protocol（NDP）はルータ関連以外の機能を持つ．

IPv6 スタックの UDP 上で動作する TFTP クライアントは，回路変更の際に構成データをダウンロードするために用いる．本システムではファイル転送プロトコルとして TFTP を用いたが，今後は HTTP や FTP にも対応する予定である．

IPv6 スタック上で動作する IPsec スタックは，本システムにおける適用例であるが，構成データをダウンロードするためにも利用される．実装した IPsec スタックは通信の暗号化として ESP および認証付き ESP，認証として AH に対応する．また，SPD および SAD の管理機能を持ち，IPsec における Transport モードに対応する．実際の暗号化およびハッシュ値の計算には，次に説明する暗号処理関数を利用する．

暗号処理

暗号処理にはソフトウェア版とハードウェア版がある．ソフトウェア版はソフトウェアで暗号化およびハッシュ値の計算を行うが，ハードウェア版は 5.3 節にて説明した暗号回路を制御して，ハードウェアで暗号処理を行う．

ソフトウェア版およびハードウェア版が対応しているアルゴリズムを**表 5.3** に示す．

表 5.3: 利用可能な暗号およびハッシュアルゴリズム

アルゴリズム	ソフトウェア版	ハードウェア版
HMAC-SHA1	O	X
HMAC-MD5	O	X
DES	O	O
3DES	O	O
Rijndael	O	O

一般的に，IP パケットにおけるハッシュ値の計算は暗号化に比べて計算負荷が小さい．本実装ではハッシュアルゴリズムはハードウェア化の対象外とし，どんなに利用頻度が高くてもソフト

ウェアで実行する．ハッシュアルゴリズムのハードウェア化は今後の課題とする．

ソフトウェア版の各種アルゴリズムは，Linux CryptoAPI Project[36] にて公開されている実装が基になっている．DES を例に，暗号アルゴリズムを呼び出すサンプルコードを以下に示す．

```
struct DES_CTX ctx;                 /* コンテキスト保存領域 */
unsigned char key[8] = "aaaaaaaa";  /* 鍵（64bit） */
unsigned char str[8] = "bbbbbbbb";  /* 平文（64bit） */

/* 鍵をコンテキストに設定する */
des_set_key(&ctx, key, 8);
/* 平文 str を暗号化し，暗号文を str に上書きする */
des_encrypt(&ctx, str, str, 8);
```

ハードウェア版の各種アルゴリズムは，5.3 節にて説明した暗号回路が基になっている．DES を例に，暗号回路を制御するサンプルコードを以下に示す．なお，本実装では暗号回路はホスト CPU のメモリ空間にマッピングされており，そのメモリ空間にアクセスすることで暗号回路を制御する．

```
#define __BASE__     0x600000   /* メモリ空間における暗号回路のアドレス */
/* メモリアドレス（0x600000 + off）を読み出すマクロ */
#define crp_read(off)        {*(volatile uint8_t *)(__BASE__ + off)}
/* メモリアドレス（0x600000 + off）に d を書き込むマクロ */
#define crp_write(off, d)    {*(volatile uint8_t *)(__BASE__ + off) = d;}

int i;
unsigned char key[8] = "aaaaaaaa";  /* 鍵（64bit） */
unsigned char str[8] = "bbbbbbbb";  /* 平文（64bit） */

/* 鍵を暗号回路の Key レジスタ（0x600000〜0x600007）に設定する */
for (i = 0; i < 8; i++)
        crp_write(i, k[i]);
/* 平文を暗号回路の Data レジスタ（0x600008〜0x60000f）に設定する */
for (i = 0; i < 8; i++)
        crp_write(i + 8, str[i]);
/* 暗号回路の Command レジスタ（0x600010）の暗号化フラグを立てる */
crp_write(0x10, ENCRYPTION_START);
/* 暗号回路の Status レジスタ（0x600008）の DONE フラグが立つまで待つ */
while (!(crp_read(0x08) & ENCRYPTION_DONE))
        ;
/* 暗号文を暗号回路の Data レジスタ（0x600008〜0x60000f）から読み出す */
for (i = 0; i < 8; i++)
        str[i] = crp_read(i + 8);
```

DES 復号化を例に，ハードウェア化対象の暗号処理を呼び出す IPsec Switching のコードを以

下に示す．以下のコードでは，IPsec スタックが ESP を受信すると ipsec_esp_input() 関数が呼ばれる．ipsec_esp_input() 関数で SAD を引き，使用する復号化アルゴリズムを決定する．この例では DES が選ばれ，ハードウェア版またはソフトウェア版の DES 復号化関数が呼び出される．

```
/* IPsec ESP（Encapsulated Security Payload）入力関数 */
int ipsec_esp_input(...)
{
        /* SAD を引き，使用する暗号アルゴリズムを決定する */
        ...
        /* DES 復号化（Cipher Block Chanining: CBC モード）を使用する場合 */
        if (esp_algo == ESP_DES_CBC) {
                des_cbc_decrypt(...);

/* DES 復号化関数（CBC モード） */
int des_cbc_decrypt(...)
{
        /* Initialization Vector（IV）の処理 */
        ...
        /* ペイロードをブロックサイズごとに復号化する */
        for (size -= DES_BLOCKSIZE; size >= 0; size -= DES_BLOCKSIZE) {
                /* DES の利用回数をカウントアップする */
                des->count++;
                /* 現在，DES がハードウェア化されているか確認する */
                if (des->status == HARDWARE_IMP_DONE) {
                        /* DES 関数（ハードウェア版）を呼び出す */
                        hw_des_decrypt(...);
                } else {
                        /* DES 関数（ソフトウェア版）を呼び出す */
                        sw_des_decrypt(...);
                }
```

回路変更制御

回路変更制御は，上記のリアルタイム OS における周期起動タスクとして実装した．回路変更制御を周期起動タスクとして OS に登録する IPsec Switching のコードを以下に示す．以下のコードでは，カーネル初期化直後に rconf_init() が呼ばれ，回路変更制御を行う rconf_cychdr() を周期起動タスクとして登録する．

```
/* 回路変更制御の初期化関数（カーネル初期化直後に実行される）*/
void rconf_init()
{
        T_CCYC pk_ccyc;    /* Creation information of CYClic handler */
        pk_ccyc.cycatr = TA_STA;
        pk_ccyc.exinf  = (void *)NULL;
        pk_ccyc.cychdr = (void *)rconf_cychdr;
        pk_ccyc.cyctim = DEFAULT_INTERVAL;   /* 周期 */
        pk_ccyc.cycphs = DEFAULT_INTERVAL;
        /* 周期起動ハンドラの生成（μ ITRON4.0 のシステムコール） */
        cre_cyc(0, &pk_ccyc);

/* 回路変更制御関数（OS によって周期的に実行される） */
void rconf_cychdr(VP_INT exinf)
{
        /* 回路変更制御 */
        ...
```

回路変更制御すなわち rconf_cychdr() 関数の処理を順に説明する．

rconf_cychdr() は初期化時に周期を DEFAULT_INTERVAL に設定したが，interval の自動調整機能によって interval は 2 倍，4 倍，... と変化する．以下のように，interval が経過したか確認し経過していなければ処理を終了する．

```
        /* 単位時間（interval）が経過したか確認する */
        if ((base.timer -= DEFAULT_INTERVAL) > 0)
                return;
        base.timer = base.interval;
        /* 経過時間の加算 */
        ...
```

上記した通り，ハードウェア化対象のアルゴリズム（DES，3DES，Rijndael）は，利用される度に利用回数をカウントアップする．本実装では，再構成可能なハードウェアを 1 つだけ持ち，各アルゴリズムの重みは全て 1，マージンは± 0%とした．よって，以下のように，最も利用回数の多いアルゴリズムがそのままハードウェア化の候補となる．

なお，利用回数の集計と同時に利用回数カウンタをリセットする．

```
max_count = 0;
candidate = -1;
for (id = 0; id < 3; id++) {
        element = &base.element[id];
        if (element->count > max_count) {
                max_count = element->count;
                candidate = id;
        }
        element->count = 0;     /* 利用回数のリセット */
}
```

ハードウェア化の候補が決まると，以下のように，その処理がすでにハードウェア化されているか確認する．ハードウェア化されていれば，回路変更の頻度情報（CPD）を更新し処理を終了する．ハードウェア化されていなければ，回路変更を実行する．

```
element = &base.element[candidate];  /* ハードウェア化の候補 */
if (element->stauts == HARDWARE_IMP_DONE) {
        /* ハードウェア化されていれば，CPD を更新し終了する*/
        ...
}
```

回路変更の必要性があれば，構成データファイルをダウンロードする．本システムではファイル転送プロトコルとして TFTP を用いる．以下のように，TFTP クライアントに構成データを保持するサーバのホスト名とそのファイル名を渡し，構成データファイルをダウンロードする．

```
/* hostname から filename をダウンロードし，buffer に格納する */
tftp_client_get(element->hostname, element->filename,
                buffer, buffer_length);
```

本実装では，サーバよりダウンロードした構成データファイルは 1.88Mbit 程度となる．構成データファイルには以下の情報が含まれる．

- デザイン名
- 実装デバイス名
- 構成データが生成された時間
- 構成データ本体

まず，ファイルに記載された実装デバイス名がこれから書き換える FPGA と一致するか確認する．問題がなければ，構成データファイルから構成データ本体を取り出す．

構成データの取得が完了すると，FPGA を書き換える．5.4.1 小節にて説明した通り，FPGA の書き換えはパラレルポートを介し 8bit 単位で行う．FPGA を書き換えのサンプルコードを以下に示す．

```
/* FPGA 書き換えのための初期化処理 */
...
/* FPGA の DATA[0-7] 信号に構成データを 1Byte ずつ出力し，CLOCK を刻む */
for (i = 0; i < configuration_data_size; i++) {
        FPGA_DATA = configuration_data[i];
        FPGA_CLOCK = 0;   /* CLOCK Low  */
        FPGA_CLOCK = 1;   /* CLOCK High */
}
```

FPGA の書き換えが完了すると，以下のように，ハードウェア化した暗号アルゴリズムの状態をハードウェア化済みに変更する．そうすることで，次回，IPsec スタックがそのアルゴリズムを利用する時は，ハードウェア版の暗号処理関数を呼び出せるようになる．

```
        element->stauts = HARDWARE_IMP_DONE;
```

以下のように，回路変更の頻度情報（CPD）を更新する．回路変更を実行した場合は回路変更回数を加算する．

本実装では，CPD の上限値を 10 とした．よって，CPD を計算した結果，1 日 10 回以上の割合で回路変更が発生していると，単位時間（interval）が 2 倍となる．

```
        /* 回路変更回数の加算 */
        base.change++;
        /* CPD の計算 */
        base.cpd = 86400 * base.change / base.elapsed_time;
        /* CPD の上限値を越えた場合 */
        if (base.cpd > base.max_cpd) {
                base.interval *= 2;
                ...
```

第6章　評価

本章では，第5章にて実装したIPsecスタックを用いて，第3章にて説明した本研究のアプローチを評価する．まず，6.1節にて評価のポイントを2つ挙げ，6.2節および6.3節にてそれぞれの評価を行う．

6.1　評価のポイント

まず，第5章にて実装したIPsec Switchingを動作させ，本システムが以下の2点を満たすことを確認した．

- 利用頻度の高い処理はハードウェアで実行し，それ以外の処理はソフトウェアで実行する．
- 構成データをネットワーク上で管理し，ネットワークを介してハードウェアを更新する．

よって，本システムは第3章にて説明したモデルを満たし，本システムによるハードウェア化は第1章にて示した低管理コストおよび長寿命を実現した．

本研究のアプローチであるネットワーク上での構成データの管理およびネットワークを介したハードウェア構成の自動更新では，ネットワーク上から構成データをダウンロードするためのオーバヘッドが問題となる．しかし，ネットワークアプライアンスにおいては用途が限られているため利用形態の変化は少なく，さらに，利用頻度に基づいた回路変更によって回路変更の頻度を必要最小限に出来るので，この欠点は大きな問題にはならないと仮定した．

そこで，本システムがネットワークアプライアンスにおいて効果的か確かめるために以下の評価を行う．

(1) **回路変更のオーバヘッドの測定**
回路変更にかかるオーバヘッドを測定し，そのオーバヘッドに見合った回路変更ポリシとして利用頻度に基づいた回路変更の必要性を示す．

(2) **利用頻度に基づいた回路変更の動作確認**
利用頻度に基づいた回路変更の動作実験を行い，上記の仮定の正しさ，すなわち，本システムによって回路変更の頻度を抑えつつ効率的に処理をハードウェア化出来ることを示す．

以降，6.2節にて(1)について，6.3節にて(2)についてそれぞれ実験および評価する．

6.2　回路変更のオーバヘッド

まず，**図 6.1** に示す(a)〜(c)の3種類の実装を用いて回路変更に必要な時間を測定する．3種類の測定結果を比較することで回路変更のオーバヘッドの原因が明確になる．オーバヘッドの原因が明確になれば，別の環境においても，オーバヘッドの目安を得易くなる．

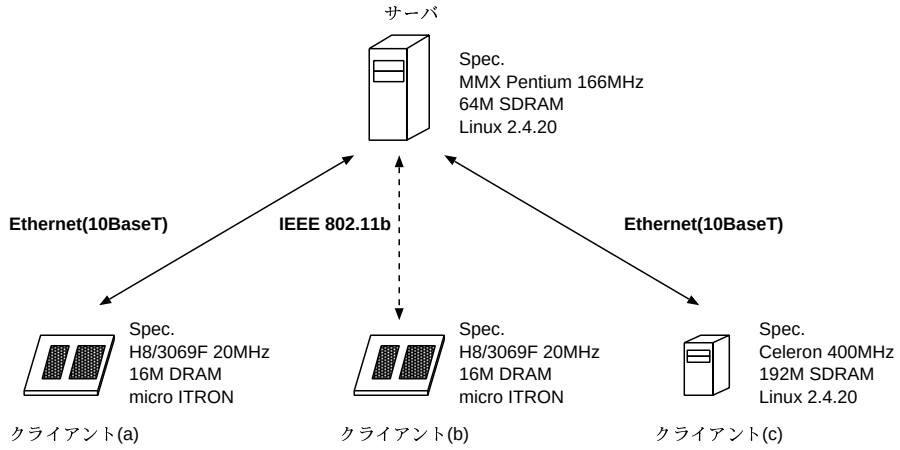

図 6.1: 回路変更の所用時間の測定環境

(a) は第 5 章にて実装したシステムであり，CPU は H8/3069F 20MHz，通信メディアは Ethernet（10BaseT）である．

(b) も第 5 章にて実装したシステムであるが，通信メディアは無線 LAN（IEEE802.11b）である．

(c) は一般的な PC であり，CPU は Celeron 400MHz，通信メディアは Ethernet （10BaseT）である．パラレルポートを使いスレーブパラレルモードで FPGA を書き換える．

(a)〜(c) における回路変更の所要時間とその内訳を**表 6.1** に示す．表中の「ダウンロード時間」とは構成データをサーバからそれぞれの通信メディアを用いてダウンロードする時間,「書き換え時間」とは全ての構成データを FPGA に転送する時間を表す.

表 6.1: 回路変更の所用時間とその内訳（単位:秒）

	ダウンロード時間	書き換え時間	合計時間
(a)	4.55	1.43	5.98
(b)	6.60	1.43	8.03
(c)	0.61	0.81	1.42

構成データのサイズ: 1.88Mbit

(a)〜(c) の結果より回路の書き換え時間は 1 秒前後と比較的小さいが，数 Mbit の構成データをダウンロードするための時間が (a) と (b) でかなり大きい．(c) では構成データのダウンロード時間は小さいが，(c) と (a) の通信メディアは同じなので，帯域がボトルネックの原因だとは言えない．(a) と (b) は，(c) よりも圧倒的に処理能力が限られているため，構成データのダウンロード時間が大きくなった原因は (a) と (b) のパケットの処理能力にあると言える．

上記の実験では構成データを一括してダウンロードし，ダウンロードが完了してから FPGA を書き換えた．この場合，数 Mbit にも及ぶ構成データを一時的に保存しておくためのバッファが必要となる．ネットワークを介した回路変更では構成データを分割してダウンロード出来るので，小さなバッファしか持たない環境でも本システムを実現出来る．そこで，分割して書き込む場合を想定して，以上の実験と同様に回路変更のオーバヘッドを測定した．その結果を**表 6.2** に示す.

(a)〜(c) が分割ダウンロード，(d) が一括ダウンロードの場合である．(a)〜(c) と (d) を比較してもその差は 1 秒程度と小さく，分割ダウンロードによるオーバヘッドは小さいと言える．

表 6.2: 通信バッファサイズと回路変更の所用時間（単位:秒）

		バッファサイズ	所用時間
(a)		512Byte	7.09
(b)	分割書き込み	4KByte	7.00
(c)		32KByte	6.89
(d)	一括書き込み	256KByte	5.98

構成データのサイズ: 1.88Mbit

以上の実験より，本研究のアプローチであるネットワーク上での構成データの管理およびネットワークを介したハードウェア構成の自動更新において，回路変更は機器のパケット処理能力によっては 6〜8 秒かかる重い処理となった．この場合，回路変更の間隔が最短でも 10 秒以上あれば本システムは動作可能であるが，回路変更の処理は重いので回路変更の頻度を減らす必要がある．

本研究では，利用頻度に基づいた回路変更によって回路変更の頻度を必要最小限に出来ると仮定しており，それを 6.3 節にて確かめる．

6.3 利用頻度に基づいた回路変更

本節では，本システムによって回路変更の頻度を抑えつつ効率的に処理をハードウェア化出来ることを示す．そのために，想定されるネットワークアプライアンスの利用環境で本システムの動作実験を行い，本システムが常に効果的であることを示す．

実装例である暗号回路では，回路変更制御の善し悪しは通信量，通信頻度などのトラフィックパターンに左右される．様々なトラフィックパターンを実機で発生させることは難しいが，PC 上で様々なトラフィックパターンをシミュレーションすれば，回路変更制御の網羅的な評価が出来る．そこで，本研究では回路変更制御の性能評価のためにシミュレーションプログラムを利用した．

シミュレーションプログラムの想定環境を**図 6.2** に示す．図では，本システムに対して複数の通信相手から SNMP や HTTP のアクセスが発生し，これらの通信は IPsec により暗号化される．なお，使用される暗号アルゴリズムは通信相手ごとに異なるものとする．

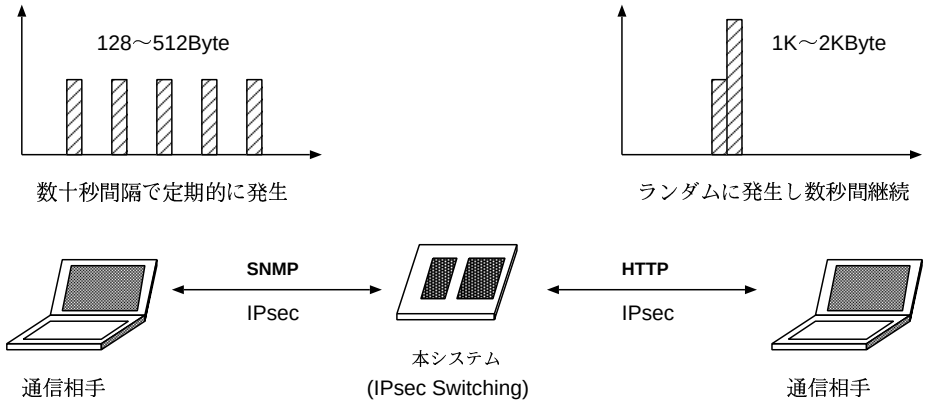

図 6.2: シミュレーションの想定環境

シミュレーションプログラムを用いて発生させたトラフィックパターンは以下の 3 種類である．

(1) SNMP 型:
定期的なセンサの監視など，SNMP によって発生するトラフィックパターンを想定している．数十秒ごとに往復 128〜512Byte と比較的小さなトラフィックが発生するものとする．

(2) HTTP 型:
ユーザによる機器の操作など，HTTP によって発生するトラフィックパターンを想定している．突発的に往復 1K〜2KByte と比較的大きなトラフィックが数秒間集中的に発生するものとする．

(3) 混合型:
(1)SNMP 型と (2)HTTP 型を合成したものとする．

6.3.1 小節にて (1)SNMP 型，6.3.2 小節にて (2)HTTP 型，6.3.3 小節にて (3) 混合型のトラフィックパターンを用いて利用頻度に基づいた回路変更制御を評価する．

評価では回路変更の間隔（interval）を変化させながら，1 日当たりどれだけ回路変更が発生するか（cpd），また，ハードウェア化した回路がどれだけ有効に利用されたか（hit[%]）を確認する．

シミュレーションでは，通信相手ごとに異なる暗号アルゴリズムが利用されるが，クライアントが一度にハードウェア化出来る暗号アルゴリズムの数は 1 つだけとする．

各暗号アルゴリズムのブロックサイズは全て 64bit で，重み（weight）は全て 1 とした．例えば，16Byte のデータを暗号化すると，64bit ブロックの暗号化処理が二度呼び出されるため，重要度が 2 加算される．

効果の小さな回路変更を抑制するため重要度のマージン（margin）を± 2.5%とした．

6.3.1 SNMP によるアクセス時

まず，利用頻度に応じた回路変更制御が設計通りに動作するか確認する．SNMP 型のトラフィックとして**図 6.3** に示すトラフィックパターンを生成した．3 種類のトラフィックはそれぞれ固有の時間間隔ごとに固有のサイズの通信を行い，それぞれ異なった暗号アルゴリズム (a)(b)(c) を用いる．

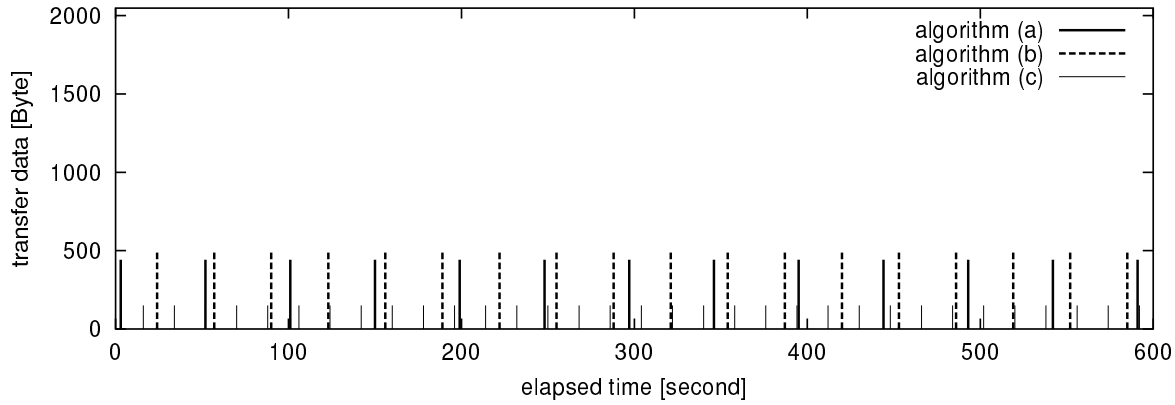

図 6.3: 生成したトラフィックパターン (1)

図 6.3 のトラフィックパターンにおいて，各暗号アルゴリズムの重要度の推移を**図 6.4** に示す．なお，ここでは回路変更の間隔を 100 秒としたので，100 秒ごとに最も重要度が高い暗号アルゴリズムがハードウェア化され，同時にそれまでの利用回数はクリアされる．図 6.4 では 100 秒経過後に暗号アルゴリズム (b) がハードウェア化され，その後も回路は暗号アルゴリズム (b) のままなので，回路変更は最初の 1 回だけで済んだ．

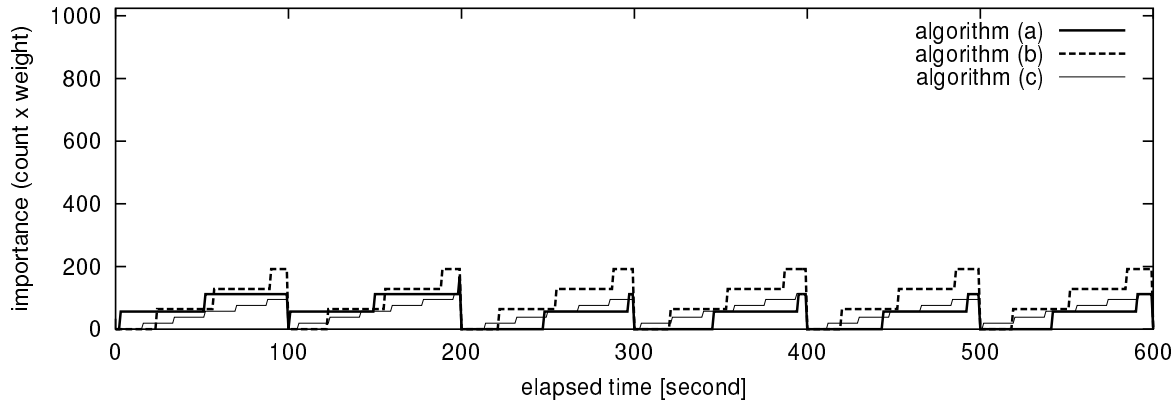

図 6.4: 各暗号アルゴリズムの重要度の推移 (1)

図 6.3 のトラフィックパターンにおいて，回路変更の間隔を 10〜300 秒の間で変化させた場合の 1 日当たりの回路変更回数（change）とハードウェア化した回路の利用率（hit）を計算した．その結果を**図 6.5** に示す．回路変更の間隔を 66 秒以上にすると回路変更はほとんど発生しなくなった．また，3 種類の暗号アルゴリズムのうち 1 つを無作為にハードウェア化した場合，ハードウェアの利用率は平均 33.3%となるが，本システムでは発生したトラフィックの約 47%をハードウェア上で実行出来た．以上より，図 6.3 のトラフィックパターンにおいて本システムが設計通りに動作していることを確認した．

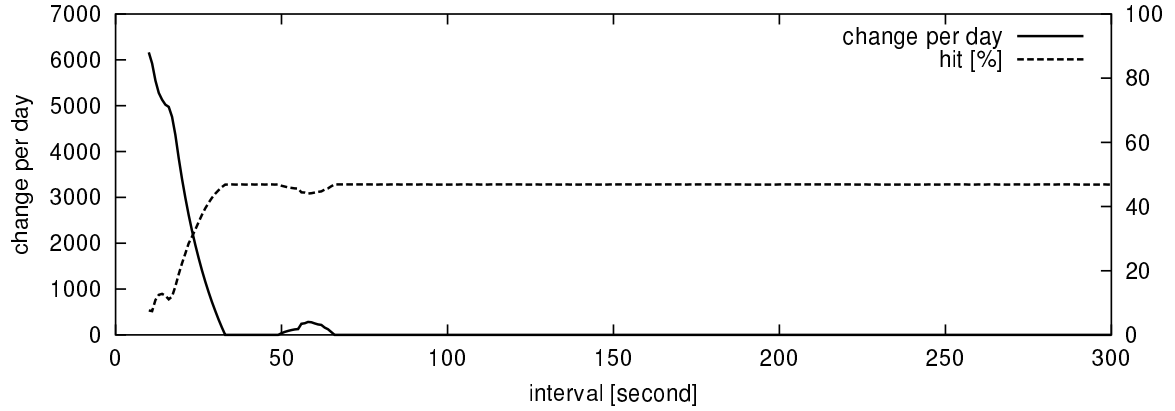

図 6.5: 1 日当たりの回路変更回数と回路の利用率 (1)

次に，一定だったトラフィックパターンを突然変化させ，利用頻度に基づいた回路変更制御が新たなトラフィックパターンに適応出来るか確認する．図 6.4 のトラフィックパターンにおいて，最も重要度が高い暗号アルゴリズム (b) が 300 秒経過後から利用されなくなったとする．その時のトラフィックパターンを**図 6.6** に示す．

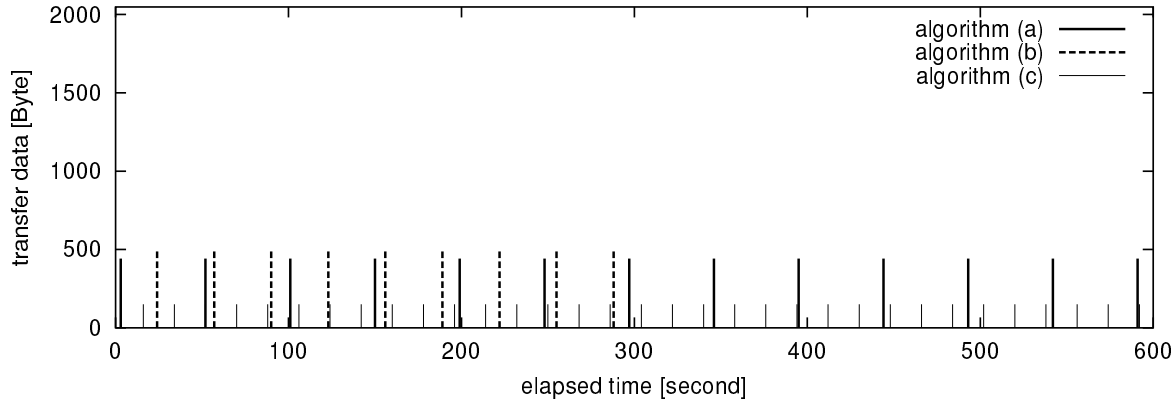

図 6.6: 生成したトラフィックパターン (1a)

図 6.6 のトラフィックパターンにおいて，各暗号アルゴリズムの重要度の推移を**図 6.7** に示す．図 6.7 では 300 秒経過後に暗号アルゴリズム (b) が利用されなくなり，次の回路変更で 2 番目に重要度が高い暗号アルゴリズム (a) がハードウェア化された．実験では回路変更の間隔を 100 秒にしたので，トラフィックパターンの変化から 100 秒後に回路が変更された．もちろん，この間隔を短くすればトラフィックパターンの変化にさらに迅速に適応出来るが，回路変更の頻度は高くなる．このように本システムはトラフィックパターンの変化に適応出来ることを確認した．

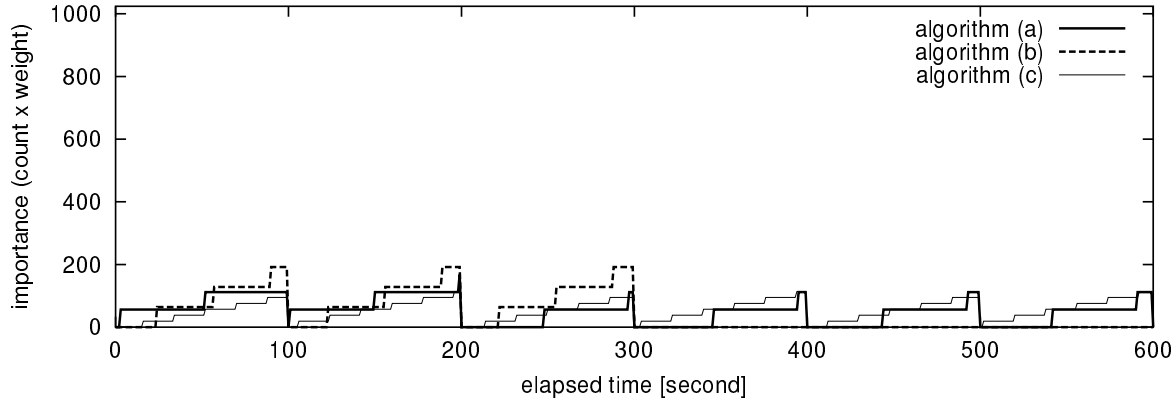

図 6.7: 各暗号アルゴリズムの重要度の推移 (1a)

本研究では効果の小さな回路変更を抑制するため，重要度に±数%のマージンを設けることを提案しており，最後にその効果を確認する．実験のため，今までと同様に 3 種類のトラフィックを生成するが，今回は全て同一の間隔で，同一のサイズの通信を行うものとする．このような条件では，3 種類の暗号アルゴリズムの重要度が僅差で交替し続け，回路変更の間隔をいくら大きくしても収束しない場合がある．3 種類の暗号アルゴリズムの重要度は常に僅差なので回路変更の効果は小さい．そこで，重要度のマージンによって，大幅に回路変更の回数が減るかどうか確認する．実験に利用したトラフィックパターンを**図 6.8** に示す．

図 6.8 のトラフィックパターンにおいて，回路変更の間隔を 10〜300 秒の間で変化させた場合の 1 日当たりの回路変更回数とハードウェア化した回路の利用率を計算した．その結果を**図 6.9** に示す．重要度のマージンは± 2.5%に設定されており，回路変更の間隔を大きくするに従いその効果は大きくなる．回路変更の間隔を 209 秒以上にすると回路変更はほとんど発生しなくなり，ハー

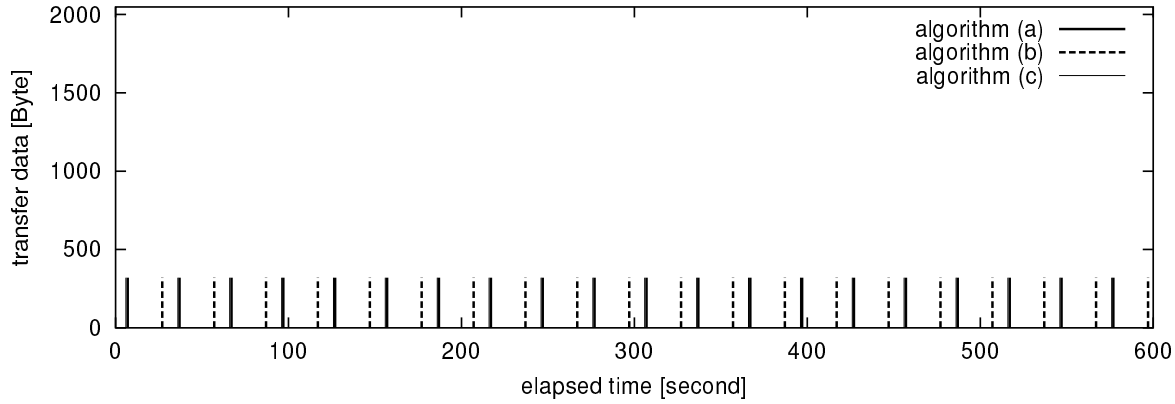

図 6.8: 生成したトラフィックパターン (1b)

ドウェアの利用率は約 33%となった．このように重要度にマージンを設けることで，効果の小さな回路変更を抑制出来ることを確認した．

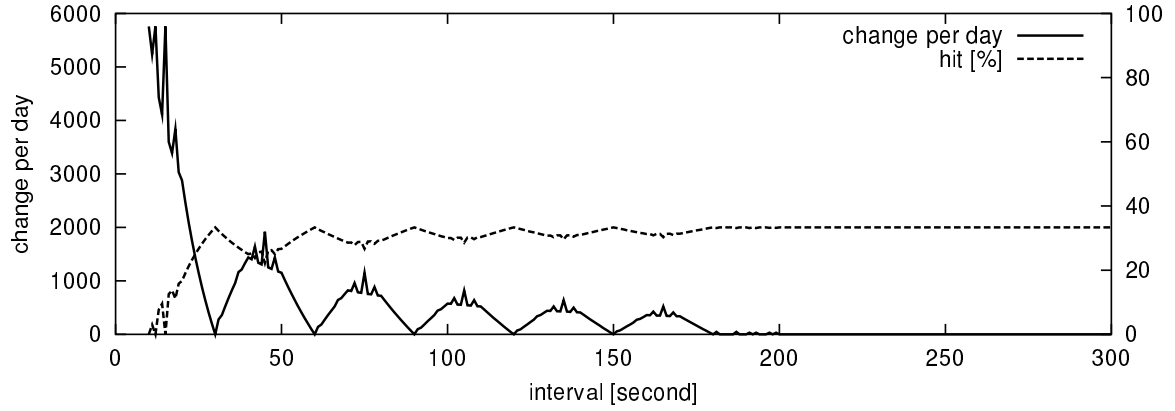

図 6.9: 1 日当たりの回路変更回数と回路の利用率 (1b)

6.3.2 HTTP によるアクセス時

SNMP 型のようにトラフィックパターンが規則的な場合，処理の発生パターンが一定であり変化しにくいので，暗号アルゴリズムの利用頻度も変化しにくい．そのため，1 単位時間前の利用頻度に基づいた回路内容が現在も有用である場合が多い．

逆に，HTTP 型のようにトラフィックパターンに規則性が乏しい場合，処理がランダムに発生するため，1 単位時間前の利用頻度に基づいた回路内容が現在も有用であるとは限らない．それでも，長期的に見れば「暗号アルゴリズム (a) と (b) は 7:3 の割合で利用されている」など，利用頻度の偏りを見出すことは出来る．HTTP 型の場合は長期的に見て利用頻度の偏りを見出すので，回路変更の間隔は SNMP 型よりはるかに長くする必要がある．

そこで，SNMP 型に続き HTTP 型のトラフィックパターンにおいても同様の実験を行い，本システムによって回路の利用率が向上するかどうか確認する．HTTP 型のトラフィックとして**図 6.10** に示すトラフィックパターンを生成した．3 種類のトラフィックはそれぞれ異なる確率で発

生し，数秒間集中的に通信を行う．暗号化には，それぞれ異なる暗号アルゴリズム (a)(b)(c) を用いる．

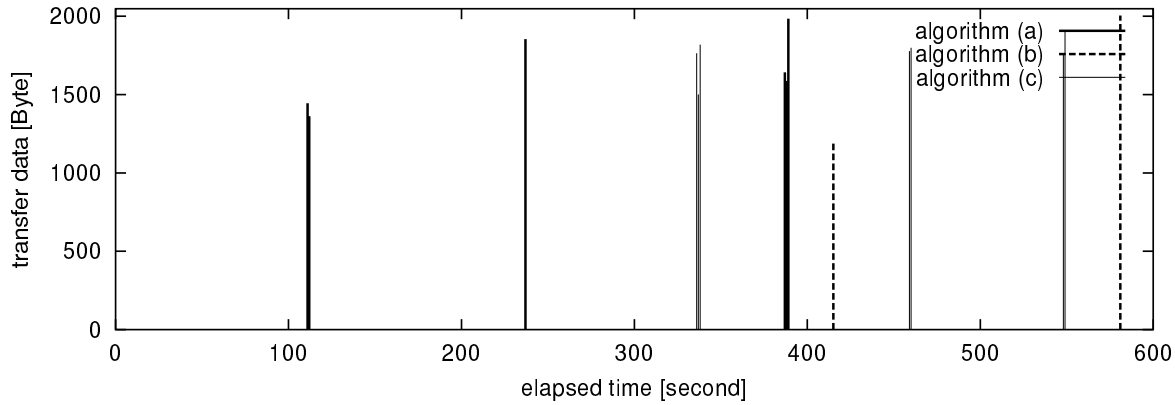

図 6.10: 生成したトラフィックパターン (2)

図 6.10 のトラフィックパターンにおいて，回路変更の間隔を 10〜3600 秒の間で変化させた場合の 1 日当たりの回路変更回数とハードウェア化した回路の利用率を計算した．その結果を**図 6.11** に示す．回路変更の間隔を 3600 秒（1 時間）にしても回路変更は完全には収束せず，1 日当たり 3 回の回路変更が発生した．この時のハードウェアの利用率は約 44%となり，3 種類の暗号アルゴリズムを無作為にハードウェア化した場合（平均 33.3%）と比べて向上している．このように回路変更の間隔を大きくすれば，規則性の乏しいトラフィックパターンにおいても利用頻度の偏りを見出し，利用率を向上出来ること確認した．

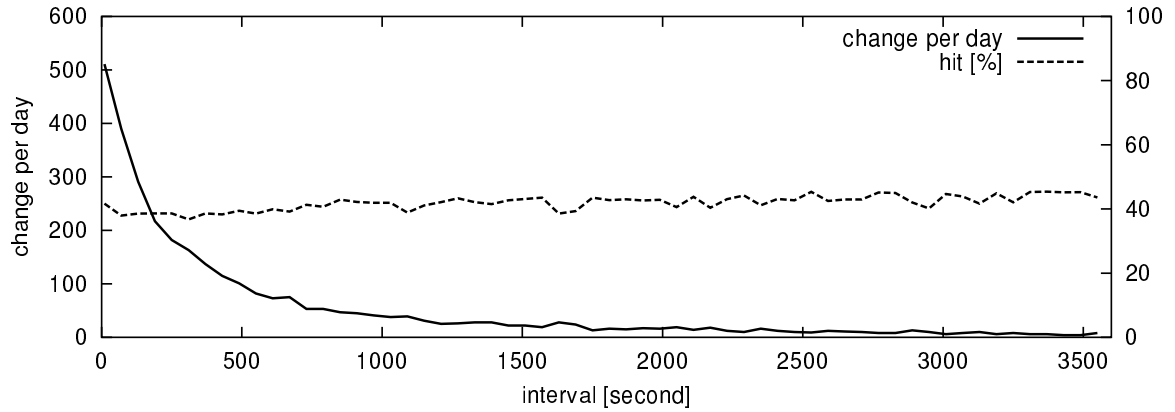

図 6.11: 1 日当たりの回路変更回数と回路の利用率 (2)

6.3.3 SNMP と HTTP によるアクセス時

現在でもルータやスイッチなどのネットワーク機器では SNMP 機能と HTTP 機能を持つ場合があるので，今後登場し得るネットワークアプライアンスでも両方の機能を持つ場合を想定する必要がある．そこで，混合型のトラフィックパターンにおいても，今までと同様に本システムが有効に働くか確認する．

混合型のトラフィックとして図 6.3 と図 6.10 のトラフィックパターンを単純に合成した．合成したトラフィックパターンを**図 6.12** に示す．トラフィックの種類は 6 種類となった．

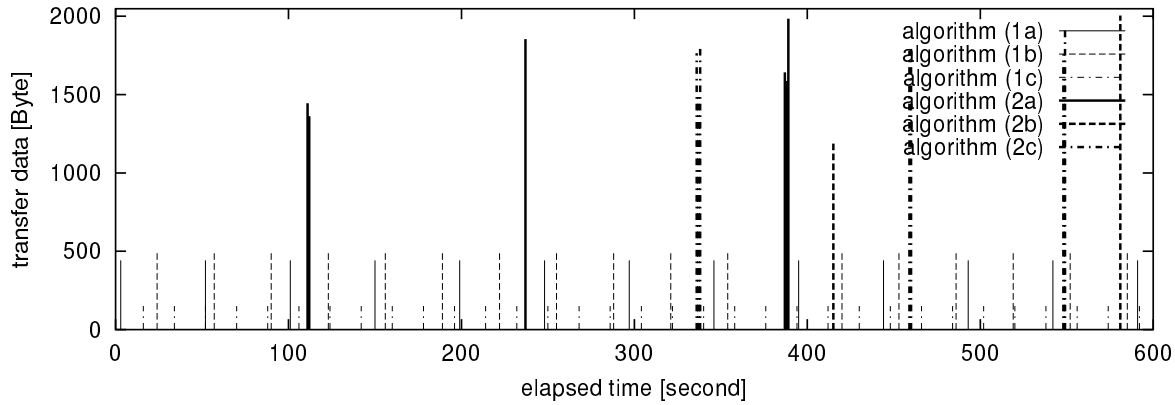

図 6.12: 生成したトラフィックパターン (3)

図 6.12 のトラフィックパターンにおいて，回路変更の間隔を 10〜3600 秒の間で変化させた場合の 1 日当たりの回路変更回数とハードウェア化した回路の利用率を計算した．その結果を**図 6.13** に示す．回路変更の間隔を 3600 秒（1 時間）にしても回路変更は完全には収束せず，1 日当たり 3 回の回路変更が発生した．この時のハードウェアの利用率は約 24%となり，6 種類の暗号アルゴリズムを無作為にハードウェア化した場合（平均 16.7%）と比べて向上している．

HTTP 型のトラフィックが無いと SNMP 型のトラフィックの重要度が安定して高いが，HTTP 型のトラフィックが発生すると重要度が大きく変化した．そのため，混合型のトラフィックパターンにおいては HTTP 型に近い実験結果となり，利用率も向上出来ることを確認した．

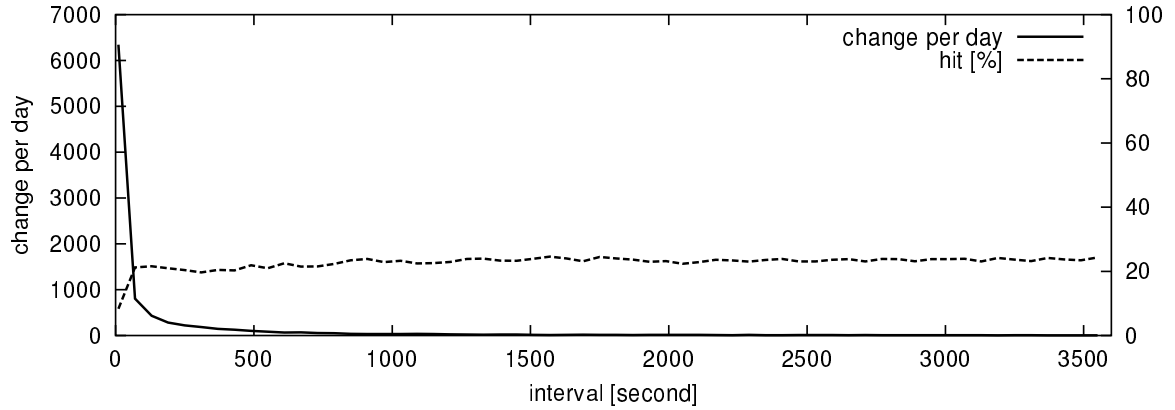

図 6.13: 1 日当たりの回路変更回数と回路の利用率 (3)

第7章　まとめ

7.1　本研究のまとめ

ネットワークアプライアンスにおいて，マルチメディアデータの取扱いや通信の暗号化などの高負荷な処理を行う機会が増えつつある．本研究の目的は，ネットワークアプライアンスを対象に，これらの処理を効率的にハードウェア化することである．

そのためのアプローチとして，再構成可能なハードウェアを用いて，利用形態に応じて自動的にハードウェア構成を更新出来るネットワークアプライアンスを実現した．

本システムでは，ネットワークアプライアンスが利用する構成データをネットワーク上で管理する．そして，それぞれのネットワークアプライアンスにおいて利用形態に変化が生じると，新たな利用形態に適した構成データをネットワーク上からダウンロードしハードウェアを更新する．本システムの利点は，各ネットワークアプライアンスが数 Mbit 以上にもなる構成データを複数セット管理する必要がない点と，不具合の修正や新機能の追加機能を兼ねる点である．一方，欠点は，回路変更の度にネットワーク上から構成データをダウンロードするため，構成データをローカルに保持する場合と比べ回路変更のオーバヘッドが大きくなる点である．しかし，ネットワークアプライアンスにおいては用途が限られているため利用形態の変化は少なく，さらに，利用頻度に基づいた回路変更によって回路変更の頻度を必要最小限に出来るので，この欠点は大きな問題にはならないと仮定した．

実装では，本システムの有効性を示すため，利用頻度が高い暗号アルゴリズムを自動的にハードウェア化する IPsec スタックを構築した．IPsec は同一システム上で複数の高負荷な処理を実行する典型的な例であるため，本研究では実装例として IPsec を選択した．

評価では，まず，回路変更にかかるオーバヘッドを測定した．その結果，処理能力が限られたネットワークアプライアンスにおいては，回路変更は負荷が高くコストの高い処理であると言え，回路変更の頻度を必要最低限まで減らしつつ最も利用されている処理をハードウェア化するための制御が必要となった．

次に，本研究にて設計し実装した利用頻度に基づいた回路変更によって，回路変更の頻度を必要最低限まで減らしつつ最も利用されている処理をハードウェア化出来るか確認した．実験ではシミュレーションプログラムを用いて SNMP や HTTP などネットワークアプライアンスで想定されるトラフィックパターンを生成し，回路変更の頻度と回路の利用率を計算した．その結果，上記の仮定の通り，処理内容が安定していれば最初の回路変更の後はほとんど回路変更は発生せず，処理内容に大きな変化があった場合にのみ回路変更が発生した．その際，無作為に処理をハードウェア化した場合よりも常に回路の利用率が向上した．

以上より，ネットワークアプライアンスにおいて，様々な処理を効率的にハードウェア化するには，本システムが効果的であることを確認した．

7.2 今後の課題

本研究の実用例である IPsec Switching では，ハードウェア化対象の暗号アルゴリズムは 3 種類だけである．今後は，より多くの暗号アルゴリズムに対応することはもちろん，IPsec だけでなく SSL へ本システムを適用する．SSL においては，セッション鍵を交換する際に RSA などの公開鍵暗号方式を利用し，その後の通信には DES などの共有鍵暗号方式を用いるため，回路変更のスケジューリングが IPsec よりも複雑になる．

音声 CODEC や動画および静止画 CODEC など，暗号処理以外の分野にも本システムを適用可能な場面は多い．特に，音声 CODEC と暗号アルゴリズムなど異なる用途の処理を利用頻度を基に比べ，ハードウェア化する点は本システムの大きな特徴であり，実用例を示す必要がある．

AV 系ネットワークアプライアンスなどユーザが存在する機器を対象に，再構成可能なハードウェアの拡張を可能にしたい．例えば，再構成可能なハードウェアを機器の拡張スロットに追加すると，再構成可能なハードウェアが追加認識され，同時にハードウェア化可能な処理の数が増えるようにする．そのためには，回路変更のスケジューラをより高度にする必要がある．

また，本システムを利用したシステムの実現が容易にするため，本システムをミドルウェア化する．そうすることで，本システムの応用範囲が広がる．

謝辞

本論文の作成にあたり，御指導頂きました慶應義塾大学環境情報学部教授 村井純博士，並びに同学部教授 徳田英幸博士，同学部助教授 楠本博之博士，同学部教授 中村修博士，同学部専任講師 南政樹氏に感謝致します.

絶えず御指導と御助言を頂きました慶應義塾大学院政策・メディア研究科後期博士課程 湧川隆次氏，同大学院修士課程 三屋光史朗氏に感謝致します.

本研究を進めていく上で御支援下さった慶應義塾大学政策・メディア研究科特別研究専任講師 植原啓介博士，同大学院特別研究助手 佐藤雅明氏，同大学院修士課程 小柴晋氏，同大学院修士課程 渡里雅史氏，並びに慶應義塾大学 徳田・村井・楠本・中村・南合同研究室 NACM 研究グループの皆様に感謝の念を表します.

参考文献

[1] Deering, S. and Hinden, R.: *Internet Protocol, Version 6 (IPv6) Specification* (1998). RFC 2460.

[2] Thomson, S. and Narten, T.: *IPv6 Stateless Address Autoconfiguration* (1998). RFC 2462.

[3] 株式会社エルミックシステム: *Web-page*. http://www.elmic.co.jp/.

[4] School on the Internet: *Web-page*. http://www.soi.wide.ad.jp/.

[5] みまもりほっとライン: *Web-page*. http://www.mimamori.net/.

[6] e-ケアタウンふじさわ: *Web-page*. http://www.e-care-project.jp/.

[7] Wakikawa, R., Watari, M., Matsutani, H., Mitsuya, K., Ernst, T., Uehara, K. and Murai, J.: Demonstration System supporting Host and Network Mobility, *IPSJ Symposium on Multimedia, Distributed, Cooperative and Mobile Systems (DICOMO2003)*, pp. 617–620 (2003).

[8] 松谷宏紀, 湧川隆次, 植原啓介, 村井純: 組み込み機器向け Mobile IPv6 プロファイルの設計, マルチメディア，分散，協調とモバイル (DICOMO2003) シンポジウム論文集, pp. 101–104 (2003).

[9] 原田雅章: 組込みソフトウェアのバージョンアップ機能を持ったμ ITRON 仕様 OS, 平成 14 年度 IPA 重点領域情報技術開発事業 (2002).

[10] Kent, S. and Atkinson, R.: *Security Architecture for the Internet Protocol* (1998). RFC 2401.

[11] Okabe, N., Sakane, S., Inoue, A., Ishiyama, M. and Esaki, H.: *Host Requirements of IPv6 for Low Cost Network Appliances* (2002). Work in Progress, draft-okabe-ipv6-lcna-minreq-02.txt.

[12] ITRON Committee, TRON ASSOCIATION: μ ITRON4.0 Specification Ver. 4.00.00 (2002).

[13] Sun Microsystems: *Web-page*. http://java.sun.com/.

[14] 松谷宏紀, 湧川隆次, 村井純: インターネット自動車のためのセンサーノードの設計と実装 (デモンストレーション), インターネットコンファレンス 2002 論文集, p. 121 (2002).

[15] Conta, A. and Deering, S.: *Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification* (1998). RFC 2463.

[16] US National Bureau of Standards: *Data Encryption Standard* (1977). FIPS PUB 46.

[17] American National Standards Institute: *Triple Data Encryption Algorithm Modes of Operation* (1998). ANSI X9.52-1998.

[18] NIST: *Advanced Encryption Standard (AES)* (2001). FIPS PUB 197.

[19] 株式会社 日立製作所: H8/3069 F-ZTAT ハードウェアマニュアル (2001).

[20] 梶崎浩嗣, 黒川恭一: 暗号処理ボード SEBSW-2 の設計と性能評価, 信学技報 VLD2002-123, pp. 19–24 (2002).

[21] Xilinx: *Architecting Systems for Upgradability with IRL (Internet Reconfigurable Logic)* (2001). XAPP412 (v1.0).

[22] Xilinx: *Web-page*. http://support.xilinx.co.jp/japan/j_prs_rls/silicon_spart /03143s3_90nm_j.htm.

[23] Xilinx: *Spartan-IIE 1.8V FPGA Family: Complete Data Sheet* (2003). DS077.

[24] SSL 3.0 Specification: *Web-page*. http://wp.netscape.com/eng/ssl3/.

[25] Case, J., Fedor, M., Schoffstall, M. and Davin, J.: *Simple Network Management Protocol (SNMP)* (1990). RFC 1157.

[26] 天野英晴: マルチコンテキストデバイスを用いた動的適応型ハードウェアの提案, 情報処理学会研究報告 (2002-ARC-150), pp. 59–64 (2002).

[27] 室岡孝宏, 宮崎敏明: 動的機能変更可能な通信ネットワークノード, 情報処理学会研究報告 (2001-SLDM-99), pp. 57–63 (2001).

[28] Smit, G. J., Havinga, P. J., Smit, L. T., Heysters, P. M. and Rosien, M. A.: Dynamic Reconfiguration in Mobile Systems, *Proceedings of FPL 2002*, pp. 171–181 (2002).

[29] Dierks, T. and Allen, C.: *The TLS Protocol Version 1.0* (1999). RFC 2246.

[30] Sollins, K.: *The TFTP Protocol (Revision 2)* (1992). RFC 1350.

[31] OPENCORES: *Web-page*. http://www.opencores.org/.

[32] Xilinx: *Web-page*. http://www.xilinx.com/.

[33] Xilinx: *Configuration and Readback of Spartan-II FPGAs Using Boundary Scan* (2002). XAPP188 (v2.1).

[34] Xilinx: *Using a Microprocessor to Configure Xilinx FPGAs via Slave Serial or SelectMAP Mode* (2002). XAPP502 (v1.2).

[35] GNU Development Tools for the Renesas H8/300[HS] Series: *Web-page*. http://h8300-hms.sourceforge.net/.

[36] Linux CryptoAPI Project: *Web-page*. http://www.kerneli.org:8080/cryptoapi/.