高速ネットワーク向けネットワークモニタ回路の設計と実装

桐村昌行[†] 高本佳史[††] 森亮憲[†]
安本慶一[†††] 中田明夫[††] 東野輝夫[††]

インターネットや高速ネットワークの発展にともない，ネットワークを流れる大量のトラフィックをリアルタイムで監視するネットワークモニタの必要性が高まってきている．ネットワークモニタに対しては監視項目の追加・変更に柔軟に対応することが要求される．本論文では，柔軟性を持ち，かつ，高速ネットワークに対応するために，FPGA（Field Programmable Gate Arrays）を用いてネットワークモニタを実装することを考える．そこで，ネットワークモニタを設計，実装するための手法，および自動合成可能なネットワークモニタの仕様記述法を提案する．仕様は並行同期 EFSM モデルで記述する．提案手法で合成される回路は監視項目などに基づいてパイプライン処理や並列処理を行うが，設計者はこれらの処理の詳細を指定する必要がない．FPGA 回路を自動合成するツールを開発し，合成される回路の速度や大きさについて評価した．

Design and Implementation of Network Monitoring Circuits for High Speed Networks

Masayuki Kirimura,[†] Yoshifumi Takamoto,[††] Takanori Mori,[†]
Keiichi Yasumoto,[†††] Akio Nakata[††] and Teruo Higashino[††]

Due to the recent progress of the Internet, we need high-speed network monitors which can observe millions of packets per second. We need to modify monitoring facilities and their capacities depending on monitoring items and network speed. In this paper, we propose (1) a methodology for designing and implementing such network monitors flexibly and (2) a methodology for describing specifications of network monitors for automatic synthesis. Specifications are described as concurrent synchronous EFSMs. The proposed technique makes it possible to synthesize an FPGA circuit suitable for given monitoring items and parameters where the designer need not consider about how pipe-line processing and parallel processing should be adopted. We have developed a tool to automatically derive FPGA circuits and evaluated the speed and size of derived circuits.

1. まえがき

近年，インターネットが急速に発展，普及してきている．これにともない，サービス妨害（DoS: Denial of Service）攻撃やフラッディング（flooding）などの悪意のある行為が増加してきており，このような行為の防止およびホストの防御が重要になってきている[1)~3)]．このことから，ネットワークに対する攻撃を検出するための様々なシステムおよびソフトウェアが提案されている[4)~6)]．文献 4) では，FDDI ネットワークに対するリアルタイムネットワークモニタが提案されている．文献 5) および 6) では，分散 DoS 攻撃を防止する手法が提案されている．

近年，ネットワークやホストに対する攻撃を検出するツールとしてネットワークモニタが注目されている．攻撃を検出する方法としてネットワークを流れたパケットのログをすべてディスクなどに記録したあとで解析する方法がある[17)]．この方法は，モニタ対象とするネットワークの規模によっては大量のデータを記憶するための装置が必要となる．また，リアルタイムで攻撃の検出ができないなどの欠点がある．

また，ネットワークモニタは，検出項目の頻繁な追加・変更が行われるため，ソフトウェアとして実装されることが多い．Snort などソフトウェアとして実現

† 大阪大学大学院基礎工学研究科
Graduate School of Engineering Science, Osaka University
†† 大阪大学大学院情報科学研究科
Graduate School of Information Science and Technology, Osaka University
††† 奈良先端科学技術大学院大学情報科学研究科
Graduate School of Information Science, Nara Institute of Science and Technology

されたネットワークモニタは，プロトコルや IP アドレス，ポート番号などをパラメータとした規則群を与え，規則に一致した場合に指定した動作を実行する．本論文では，インターネットバックボーンのような高速ネットワークに対応するためのネットワークモニタを実現することを考えている．ソフトウェアで実現されたネットワークモニタでは，規則が複雑になるにつれて，100 Mbps のトラフィックでもパケットの取りこぼしがあることが報告されている[19]．また，文献 20) では，パターンマッチングアルゴリズムを工夫することで，Pentium II 450 MHz の PC で，約 500 Mbps 程度のトラフィックから DoS 攻撃の 1 つである smurf を検出するネットワークモニタが実現されている．最新の CPU を用いることで，1〜2 Gbps 程度のトラフィックを扱えるネットワークモニタを実現することが可能であると考えられるが，10 Gbps を超える高速バックボーンで利用したい場合や，検出項目を追加したい場合には，ソフトウェアでの実現は困難であると考えられる．

より高速なネットワークに対応するために，ネットワークモニタをハードウェアで実装する手法が提案されている[8),9)]．しかし，ハードウェアでネットワークモニタを実装すると，検出項目の追加・変更に対応することが難しくなる．これを解決する 1 つの手法として，ネットワークモニタを FPGA（Field Programmable Gate Arrays）などの再構成可能なハードウェア回路で実装することが考えられる．

本論文では，並行プロセスをモデル化するための並行同期 EFSM（Extended Finite State Machine）[10),11)] を使い，ネットワークモニタの設計，実装手法，およびネットワークモニタの仕様記述法を提案する．提案手法では検出項目ごとに対応するモジュールを並行同期 EFSM で記述する．各モジュールに対する並行同期 EFSM をあらかじめ作成し，ライブラリに登録しておく．これにより，ネットワークモニタの設計者は，ライブラリに登録されている各モジュールを組み合わせることでネットワークモニタを設計することができる．また，提案手法で合成される FPGA 回路は，高速に動作するようにパイプライン処理や並行処理が自動的に適用される．この際，設計者は並行モジュール間のパイプライン処理の詳細を指定する必要はない．

本論文では，IP フラッディング[2)] 検出モジュールと，SYN フラッド[2)] 検出モジュールの設計と実装について述べる．どちらのモジュールも，ある特定の IP アドレスに対するパケット数が閾値を超えるかどうかを調べることで，フラッディングが発生しているかどうかを検出する．この閾値は検出対象の種類や通信路の帯域幅，統計データ[3),7),12)] などに基づいて決定する．ライブラリでは，閾値はパラメータとして指定されている．提案手法では，指定したパラメータ値に基づいて，異なる FPGA 回路を自動合成する．

上記のモジュールを並行同期 EFSM で記述し，文献 10)，11) の手法を用いてハードウェア記述言語 VHDL の記述に変換し，SYNOPSYS 社[13)] の論理合成ツール（FPGA Express）を用いて FPGA 回路を合成した．その結果，FPGA 回路の面積，動作速度ともに実用上問題ないことを確認した．

以降，2 章ではシステム記述モデルと回路合成について，3 章では提案するネットワークモニタについて述べる．また，4 章で回路合成の結果について説明する．

2. システム記述モデルと回路合成

2.1 並行同期 EFSM

並行同期 EFSM は，複数の EFSM と EFSM 間の同期制御情報で構成される．各 EFSM は有限個のレジスタ（変数）を持ち，各アクションでは，ゲートを介して，入力値をレジスタに取り込んだり，外部に値を出力したりすることができる．また，各アクションの実行条件として論理式（ガード式）を設定することができる．アクションは，$g\ v\ [f]$ と記述する．ここで，g はゲート，v は入力変数 $?x:t$（x は変数名，t は変数の型）と出力式 $!E$ の列，f はガード式を表す．

並行同期 EFSM では，与えられた EFSM の任意の部分集合が，あるゲートに対する遷移を同時に実行することで，そのゲートを介してデータを交換することができる．これをマルチランデブ[14)] と呼ぶ．LOTOS[14)] で用いられる並列オペレータを利用して以下のように並行同期 EFSM を記述する．

$$S ::= S|[gate\_list]|S \mid S|||S \mid \mathrm{M}$$

ここで，M は EFSM の名前であり，$gate\_list$ には EFSM 間で同期させたいアクションに対するゲートのリストを記述する．また，||| は，アクションを同期させないことを表す（$gate\_list = \emptyset$ に相当）．また，一対の EFSM 間での同期だけでなく，$E_1|[a]|(E_2|[a]|E_3)$ のように一対多や多対多の同期も記述することができる．これは，データの一斉配布や排他制御などを記述するときに有用である．

図 1 の並行同期 EFSM $E_1|[a,b]|(E_2|[a]|E_3)$ では，ゲート a に対するアクションが E_1，E_2，E_3 すべての EFSM 間で同時に実行される．たとえば，E_1，E_2，E_3 で $a!0$，$a?x$，$a?y$ がそれぞれ実行される場合，$a!0$ の出力値 0 が入力変数 x，y に代入される．このとき，

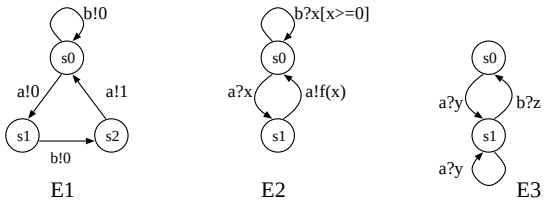

図 1 並行同期 EFSM の例（$E_1|[a,b]|(E_2|[a]|E_3)$）
Fig. 1 An example of concurrent synchronous EFSMs.

出力値 0 と変数 x，y の型が一致していなければならない．また，E_1 が $a!1$，E_3 が $a?y$ を実行し，E_2 が $a!f(x)$ を実行する場合，$f(x)$ の値は 1 でなければならない．

複数のマルチランデブが競合する場合は，いずれか 1 つが選択される．図 1 の並行同期 EFSM において，E_1 が $b!0$ を実行する場合，E_2 の $b?x[x \geq 0]$ または E_3 の $b?z$ のいずれか一方と同期実行される．

2.2 回路合成手法

我々は文献 10)，11) において，並行同期 EFSM 群として記述されたシステムの動作仕様をレジスタ転送レベル VHDL 記述へと変換する方法およびツールを提案している．以下では，変換手法の概要について説明する．

2.2.1 マルチランデブ制御のための情報

一般に並行同期 EFSM では，動的に同期アクションの組合せを決定し，ガード式が満たされるかを判定する必要がある．このような動的計算はパフォーマンスを低下させる．そこで，(1) 同期実行されるアクションの組と関係する EFSM 名の組，(2) (1) の組に対するガード式，に関する情報をあらかじめ求めておく．図 1 に対する同期アクションの組は**表 1** のようになる．p_1 から p_4 はゲート a に対するアクション，p_5 と p_6 はゲート b に対するアクションを表している．

一般に，同期アクションの組は非常に多くなるので，**表 2** のように各 EFSM の同一ゲートのアクションを 1 つにまとめて（ランデブ情報と呼ぶ）よりコンパクトな表（マルチランデブ表と呼ぶ）として表す．ここでは省略するが，並行同期 EFSM からマルチランデブ表を自動的に生成することができる[10]．

2.2.2 導出される回路のアーキテクチャ

与えられた並行同期 EFSM 群から，各 EFSM に対応する順序回路と，2.2.1 項の情報に基づくマルチランデブ制御部で構成される回路を生成する．我々が文献 10)，11) で提案している回路合成手法では，各 EFSM に対応する順序回路とマルチランデブ制御部で構成される回路を生成する．

表 1 図 1 に対する同期アクションの組
Table 1 Tuples of synchronized actions for Fig. 1.

	E_1	E_2	E_3
p_1	$(a!0,$	$a?x$	$a?y)$
p_2	$(a!0,$	$a!f(x)$	$a?y)$
p_3	$(a!1,$	$a?x$	$a?y)$
p_4	$(a!1,$	$a!f(x)$	$a?y)$
p_5	$(b!0$	$b?x[x \geq 0])$	
p_6	$(b!0$		$b?z)$

表 2 図 1 に対するマルチランデブ表
Table 2 Rendezvous table for Fig. 1.

	E_1	E_2	E_3
r_1	$(\{a!0\},$	$\{a?x,a!f(x)\},$	$\{a?y\})$
r_2	$(\{a!1\},$	$\{a?x,a!f(x)\},$	$\{a?y\})$
r_3	$(\{b!0\},$	$\{b?x[x \geq 0]\})$	
r_4	$(\{b!0\},$		$\{b?z\})$

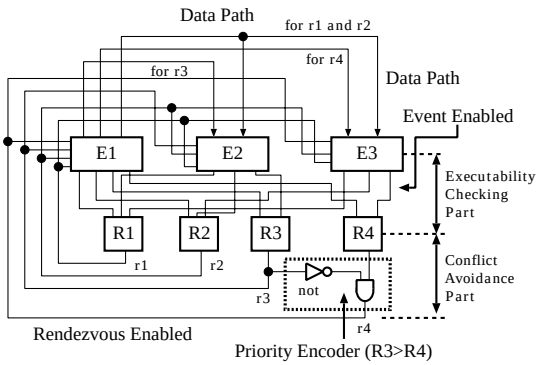

図 2 生成される回路の構成
Fig. 2 Architecture of the derived circuit.

2.2.2.1 EFSM に対する順序回路

図 1 の仕様に対して**図 2** のような回路が生成される．各 EFSM に対する順序回路は，状態を保持するレジスタ（E_1, E_2, E_3）を持つ．また，各順序回路は同じクロックを参照して動作する．EFSM は各クロックごとに，現状態から実行可能なアクションのうち 1 つを実行し，次の状態に遷移する．一方，実行可能なアクションが存在しないときには現状態で待機する．EFSM 中で用いられている各変数は，レジスタとして実現され，変数への代入を行うアクションの実行によって，各レジスタの値は更新される．

2.2.2.2 マルチランデブ制御部

マルチランデブ制御部は，同期判定部と競合回避部で構成される．

同期判定部では，各クロックごとに実行可能な同期アクションの組を判定する．同期判定部では，ランデブ情報ごとに同期判定回路を設ける（図 2 の R_1 から R_4）．同期アクションは，その同期に関わるすべての

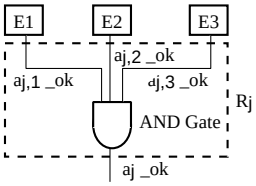

図 3 同期判定回路
Fig. 3 Synchronization checking circuit.

アクションが実行可能であるときに実行することができるので n 入力の AND 回路で実現する．ここで，n は同期を行う EFSM の個数である．

図 1 と表 2 に対する同期判定部は**図 3** のようになる．E_1 のイベント $a!1$ が実行可能になると E_1 の順序回路から信号 $a_{j,1}\_ok$ が 1 になり各 R_j（$j = 1 \ldots 4$）に送られる．同様に E_2 の $a?x$，E_3 の $a?y$ が実行可能になると信号 $a_{j,2}\_ok$，$a_{j.3}\_ok$ が 1 になり R_j へ送られる．R_j への入力がすべて 1 になれば，ゲート a に対するアクションの実行許可信号 a_j_ok を各 EFSM に送る．信号を受け取った EFSM はゲート a に対するアクションを実行する．

競合回避部は，競合する複数の同期アクションの組が同時に実行可能となったときに，いずれか 1 つを選択する回路である．選択の方法はいくつか考えられるが，ここでは簡単のために，あらかじめ設定した優先度に従って選択を行うものとする．

図 1 では表 2 より，ゲート b に対する同期アクションの組合せとして r_3 と r_4 がある．たとえば，r_3 が r_4 よりも優先度が高い場合，競合回避部は図 2 の破線内のようになる．

同期アクションの実行可能性判定に，他の EFSM の出力値が必要な場合（ガード式の判定など）がある．そこで，各同期アクションの組に属する EFSM 間にデータ転送パスを設置する．データ転送パスは，複数の同期アクション組で同時に利用される可能性があるので，ランデブ情報ごとに設置する．なお，ランデブ情報をグループ化し，パスを共用させることでパスの効率化を図ることができる．マルチランデブ制御部の詳細については文献 11) 参照されたい．

3. 提案するネットワークモニタ

本論文では，以下のような手順でネットワークモニタを開発する．

(1) ネットワークモニタを構成する各モジュールを並行同期 EFSM で記述する．
(2) 検出項目およびフラッディング検出時のアドレスの分割数やカウンタの閾値などのパラメータ値を与え，目的とするネットワークモニタの仕様を導出する．
(3) 文献 10), 11) の手法を用いて，並行同期 EFSM から VHDL 記述を生成する．
(4) SYNOPSYS 社[13] の論理合成ツール（FPGA Express）を用いて，VHDL 記述から FPGA 回路を合成する．

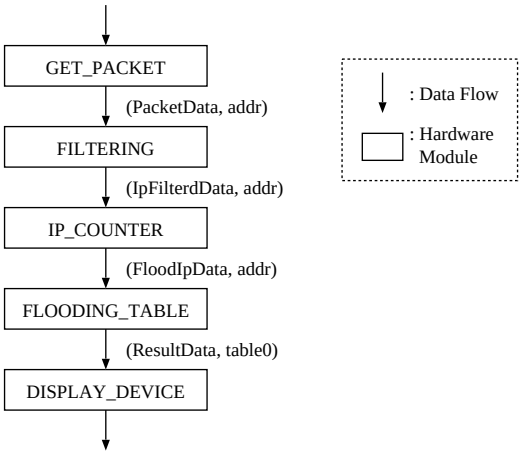

図 4 IP フラッディング検出モジュール
Fig. 4 IP flooding detection module.

本論文で対象とするネットワークモニタはヘッダ情報取得モジュール（Header Information Capturing Module）と，ネットワークモニタモジュール（Network Monitoring Module）で構成され，ヘッダ情報取得モジュールは既存のモジュール[8),9)] を使うと仮定する．ヘッダ情報取得モジュールでは,「送信元 IP アドレス」,「送信先 IP アドレス」,「パケットの種類」などを表すビット列が得られる．

提案手法では構成される回路の規模を小さくし，クロック周波数を大きくするため，FPGA に格納できるレジスタのみを使用して，外部メモリを使用しない構成法を考える．

3.1 IP フラッディング検出モジュール

3.1.1 IP フラッディング検出モジュールの構成

IP フラッディング検出モジュールの構成を**図 4** に示す．図中の矢印はマルチランデブによるデータの受渡しを表す．括弧内は，マルチランデブのゲート名および渡されるデータ名を表す．この回路への入力はヘッダ情報取得モジュールで得られたヘッダ情報である．

まず GET_PACKET モジュールがヘッダ情報からそのパケットの送信元（送信先）IP アドレスを取得する．FILTERING モジュールでは，すでにフラッディングを起こしていると判定された一定個数の IP

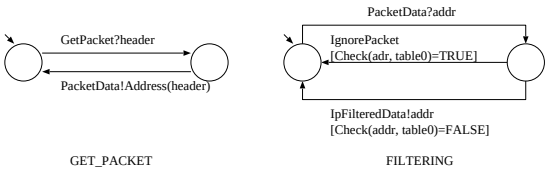

図 5 GET_PACKET と FILTERING に対する EFSM
Fig. 5 EFSMs for GET_PACKET and FILTERING.

アドレスを保持しておき，保持しているアドレスと同じアドレスを持つパケットを検出候補から外す．GET_PACKET モジュールと FILTERING モジュールに対する EFSM を**図 5** に示す．GET_PACKET モジュールは，ヘッダ情報取得モジュールからパケットのヘッダ情報 *header* を受け取り，それに含まれるアドレス情報（$Address(header)$）を送出する動作を繰り返す．一方，FILTERING モジュールは受け取ったアドレス情報 *addr* を $Check(addr, table0)$ で判定し，すでに保持しているアドレスであればその情報を無視し，まだ保持していないアドレスであれば，IP_COUNTER モジュールにその情報を渡す．ここで，$Address()$ や $Check()$ などの関数は，プリミティブとして使用できるものと仮定している．実際の回路合成時には，これらの関数を組合せ回路として実現する VHDL 記述を与える必要がある．

IP_COUNTER モジュールでは，FILTERING モジュールを通過した 32 ビットの IP アドレスをいくつかのビット列（パーティション）に分割し，それぞれのビット列ごとにフラッディングしているビットパターンを並列処理によって特定する．ここで，各パーティションが含むビット数をパーティションサイズと呼ぶ．パーティションサイズを 4 にした場合，パーティション数は 8（$= 32 \div 4$）となり，各パーティションのビットパターンは 16（$= 2^4$）種類となる．各パーティションに対して，ビットパターンと同数のカウンタを用意し，各ビットパターンごとにパケット数を数える．ある一定時間にカウンタの値が閾値を超えると，対応する IP アドレスがフラッディングを起こしている IP アドレスの一部であると判断する．すべてのパーティションに対してそのパーティションに対するビットパターンが特定されるまで同じ処理を繰り返す．このようにして，フラッディングを起こしている IP アドレスを検出する．上記の処理で検出された IP アドレスは FLOODING_TABLE モジュールに記録しておく．検出した IP アドレスを FILTERING モジュールに送り，その IP アドレスをフィルタリングする．

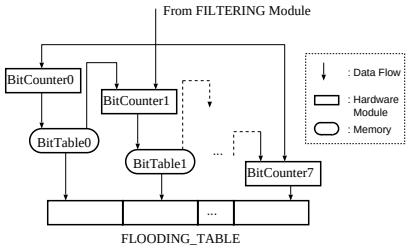

図 6 IP_COUNTER モジュール
Fig. 6 IP_COUNTER module.

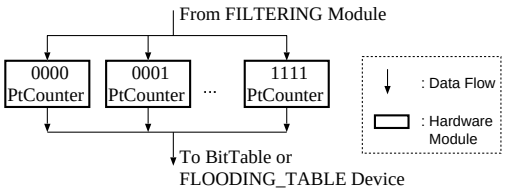

図 7 BitCounter モジュール
Fig. 7 BitCounter module.

3.1.1.1 IP_COUNTER モジュール

IP_COUNTER モジュールの詳細を**図 6** に示す．図 6 ではパーティションサイズが 4 であるので，8 個のビットカウンタ（BitCounter0–BitCounter7）と 8 個のビットテーブル（BitTable0–BitTable7）でモジュールが構成されている．まず，最上位の 4 ビットを担当している BitCounter0 が，ある一定時間，ビットパターンごとに出現した数を数える．出現数が最初に閾値を超えたビットパターンをフラッディングを引き起こしている IP アドレスの一部であると判定し，BitTable0 に保存する．この閾値はネットワークモニタが監視するネットワークの環境に依存する．また，与えられた閾値を超えるビットパターンが出現しない場合，フラッディングが発生していないと判断し，BitCounter0 をリセットする．次に BitCounter1 が，上位 4bit が BitTable0 の値に一致したアドレスに対して，BitCounter0 と同様の処理を行う．IP_COUNTER モジュールは以上の処理を順に BitCouter7 まで行う．このように上位から順番に 4bit ずつアドレスを特定することによって，最終的にフラッディングしているアドレスを特定する．BitCounter と BitTable を図 6 のように配置することにより，パイプライン処理を行うことができる．

3.1.1.2 BitCounter モジュール

BitCounter は**図 7** のような構成をしている．パーティションサイズが 4 である場合，各 BitCounter には 0000 から 1111 の 2^4 種類のビットパターンが入力される．そこで，各ビットパターンを担当する 2^4 個

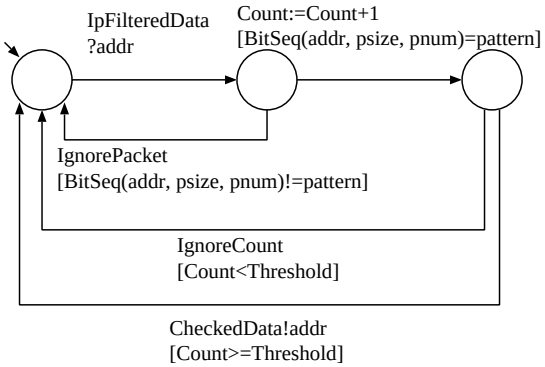

図 8 PtCounter に対する EFSM
Fig. 8 EFSM for PtCounter.

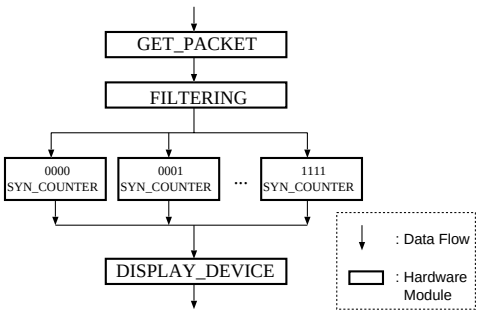

図 9 SYN フラッド検出回路
Fig. 9 SYN flood detection module.

の PtCounter を用意する．PtCounter は，入力されたビットパターンが自分の担当するビットパターンと一致すればカウンタの値を 1 増やす．この処理をある一定時間繰り返し，閾値に一番最初に到達したビットパターンを BitCounter の出力とする．PtCounter に対する EFSM を**図 8** に示す．EFSM では受け取った IP アドレス $addr$ から指定したパーティションのビット列（パーティション長および何番目のパーティションかを $psize, pnum$ で指定）を取り出し，担当ビットパターン $pattern$ と一致するかどうかを判定する（$BitSeq(addr, psize, pnum) = pattern$）．一致すれば，カウンタ $Counter$ の値を増やす．その後，条件式 $Count < Threshold$ および $Count \geq Threshold$ で閾値を超えたかどうかの判定を行う．ここで，関数 $BitSeq()$ は組合せ回路として利用できることを仮定している．

3.2 SYN フラッド検出モジュール

SYN フラッドは 2 端末間の TCP ベース通信において SYN+ACK パケットに対応する ACK パケットが返信されないことに起因して生じる．そこで，SYN フラッド検出モジュールは，SYN+ACK パケットに対応する ACK パケットの有無を監視し，対応する ACK パケットが返ってこない場合，SYN フラッドが起こっていると見なし，被害を受けているホストの IP アドレスを特定する．本検出モジュールでは，IP アドレスの 32 ビットのうち環境に応じて特定のビット列のみを監視して，そのビットパターンを特定する．たとえば，ドメインの特定などを目的にする場合，上位 8 ビットを監視することなどが考えられる．

3.2.1 SYN フラッド検出モジュールの構成

SYN フラッド検出回路の構成を**図 9** に示す．先に述べた IP フラッディング検出モジュールと同様，ヘッダ情報取得モジュールの出力から GET_PACKET モジュールが必要なヘッダ情報を抽出する．ここで抽出する情報は，送信元 IP アドレス，送信先 IP アドレスと TCP ヘッダ制御用フラグビット列である．フラグビット列は 6 ビットで構成され，パケットの種類を表している．この 6 ビットのなかに SYN，ACK に対応するビットがあり，ビットの組（SYN, ACK）が（1, 0）なら SYN パケット，（1, 1）なら SYN+ACK パケット，（0, 1）なら ACK パケットであることを表している．

3.2.1.1 FILTERING モジュール

FILTERING モジュールは，GET_PACKET モジュールによって抽出されたビット列のうち（SYN, ACK）ビット組を調べ，SYN+ACK パケットと ACK パケットのみを抽出する．さらに，FILTERING モジュールは，SYN フラッドで攻撃されているホストの IP アドレスを検出するために，パケットが SYN+ACK パケットならば送信先アドレスを，ACK パケットなら送信元アドレスを抽出する．たとえば，攻撃されているホストの IP アドレスのうち 4 ビットを検出するのであれば，2^4 個の SYN_COUNTER を準備し各カウンタが対応するビット列の数を数えればよい．

3.2.1.2 SYN_COUNTER モジュール

図 10 は SYN_COUNTER の構成を表している．まず，PT_CHECKER モジュールがヘッダ情報から必要なビット列を抽出し，担当しているビット列であるかどうかを判定する．PT_CHECKER モジュールに対する EFSM を**図 11** に示す．担当ビット列であれば，PT_CHECKER は SA_CHECKER にビット列を渡す．SA_CHECKER は，ビット組（SYN, ACK）から SYN+ACK パケットか ACK パケットかを調べる．SYN+ACK パケットなら変数 MEM の値を 1 増やす．また，ACK パケットなら MEM の値を 1 減らす．これにより，MEM は，SYN+ACK パケットの数と ACK パケットの数の差を保持する．SYN フラッドが発生し

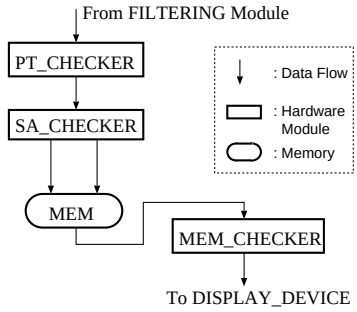

図 10 SYN カウンタ
Fig. 10 SYN_COUNTER.

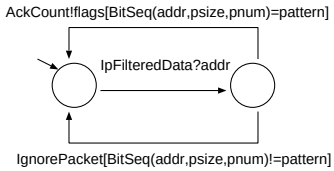

図 11 PT_CHECKER に対する EFSM
Fig. 11 PT_CHECKER.

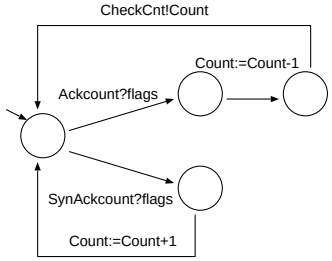

図 12 SA_CHECKER に対する EFSM
Fig. 12 SA_CHECKER.

た場合，ACK パケットが返信されないので MEM の値が増加する．SA_CHECKER モジュールに対する EFSM を**図 12** に示す．そこで，MEM_CHECKER モジュールは MEM の値が閾値を超えるかどうかを調べる．この閾値は環境に依存する．閾値を超えた場合，対応するビット列を含む IP アドレスに対して攻撃が行われていると判定する．判定後，MEM_CHECKER は DISPLAY_DEVICE にビットパターンを送り，MEM をリセットする．MEM_CHECKER モジュールに対する EFSM を**図 13** に示す．

4. 評　　価

並行同期 EFSM の仕様から，EFSM とマルチランデブ表で構成される中間プログラムを導出するツールと，中間プログラムから VHDL 記述を生成するツールを実装した[10),11)]．VHDL 記述は，市販の論理合成ツール（SYNOPSYS 社の FPGA Express など）を

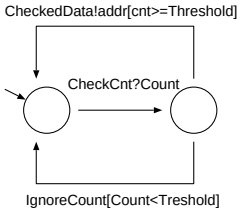

図 13 MEM_CHECKER に対する EFSM
Fig. 13 MEM_CHECKER.

表 3 各検出回路の論理合成結果
Table 3 Results for two modules.

回路名	IP-FLOOD	SYN-FLOOD
EFSM の数	156	67
論理合成時間（sec）	1,048	123
クロック周波数（MHz）	12.45	12.45
回路ゲート数（gate）	14,181	3,392
ラッチ数	15,169	482
FF 数	385	328

表 4 bit 抽出数による評価
Table 4 Evaluations depending on the number of bits.

Name	Bit	EFSMs	Area (gates)	Clocks (MHz)
SYN2	2	19	843	13.37
SYN3	3	35	1,700	13.37
SYN4	4	67	3,392	12.45
SYN5	5	131	7,062	11.51
SYN6	6	259	14,680	10.55
SYN7	7	515	30,848	9.63
SYN8	8	1027	64,598	8.67

用いて FPGA 回路に変換することができる．

4.1 回路合成の結果

前述のフラッディング検出モジュール（IP-FLOOD）と SYN フラッド検出モジュール（SYN-FLOOD）を論理合成した．どちらのモジュールについてもパーティションサイズは 4 とした．また，論理合成を行う際，FPGA チップセットとして ALTERA 社の FLEX10K シリーズを指定した．合成した結果，**表 3** のようになった．

IP-FLOOD の論理合成にかかった時間は約 17 分で，得られた回路のゲート数は約 14,000，クロック周波数は約 12.5 MHz であった．また，SYN-FLOOD の論理合成にかかった時間は約 2 分で，回路のゲート数は約 3,300，クロック周波数は約 12.5 MHz であった．

4.2 SYN フラッド 検出モジュールに対する考察

SYN-FLOOD において抽出するビット数に対する回路面積とクロック周波数の変化を**表 4** に示す．抽出するビット数が 1 つ増えるたびにカウンタ数が 2 倍になるので，回路面積が 2 倍程度に増加している．一方，カウンタの演算はすべて並列に処理しているため，ク

表 5 5bit 抽出での段数と分岐数による評価
Table 5 Evaluation depending on the numbers of vertical steps and branches when using 5 bit partition.

Name	Steps	Dummies	Branches	Area (gates)	Clocks (MHz)
SYN0-64	1	0	64	7062	11.51
SYN2-32	2	2	32	7171	12.45
SYN4-16	2	4	16	7410	14.90

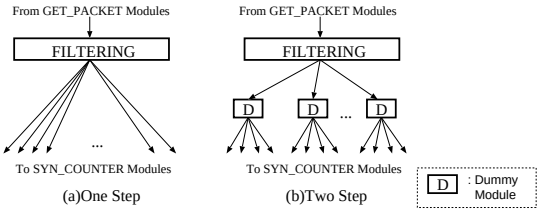

図 14 カウンタ処理の分散
Fig. 14 Distribution of counter processing.

ロック周波数はそれほど低下していない．

クロック周波数が低下しているのはマルチランデブ制御部で制御する EFSM（SYN_COUNTER）の数が増加しているためと考えられる．SYN-FLOOD では**図 14** (a) のように FILTERING モジュールがすべての SYN_COUNTER と同期をとることでパケット情報を送信している．fan-out の制限から，同期する SYN_COUNTER の数が増加すると，一度にすべての SYN_COUNTER にデータを転送できない．そこで，論理合成ツールが自動で fan-out の制限を満たすように中継用の論理ゲートを挿入し，1 クロックで処理しようとするため，FILTERING モジュールのクロック周波数が低下してしまう．

そこで図 14 (b) のように同期データ転送を 2 段（2 クロック）に分割し，FILTERING モジュールは fan-out 制限を満たす範囲で中継用のダミーモジュール D に 1 クロック目でデータ転送を行い，各ダミーモジュール D が 2 クロック目で SYN_COUNTER にデータ転送を行うようにすることで，クロック周波数の低下を防ぐことができる．このように変更した回路の回路面積とクロック周波数を**表 5** に示す．項目 Steps は図 14 における段数，Dummies はダミーモジュールの数，つまり FILTERING モジュールが直接同期するモジュールの数を指す．また，Branches は各ダミーモジュール（1 段の場合は FILTERING モジュール）から SYN_COUNTER への分岐，つまり同期すべきカウンタの数を表している．回路面積は，段数の増加，ダミーモジュールの数に応じて若干増加している．また，ダミーモジュールを追加し，2 段で処理を行うと 1 段で処理するよりもクロック周波数が 1〜3 MHz 向上している．

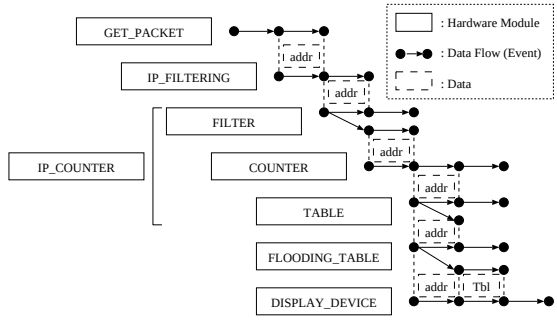

図 15 IP フラッディング検出モジュールの一連の処理
Fig. 15 Processing stages of IP flooding detecting module.

4.3 IP フラッディング検出モジュールの処理能力

各モジュールが 1 秒間にどれくらいのパケットを処理できるかを考察する．今回作成した IP フラッディング検出モジュールでのパケットの流れと各サブモジュール（EFSM）の関係を**図 15** に示す．各サブモジュールは左側の実線で囲んだものであり，右側は EFSM を表す．EFSM 間の点線で囲んだものは転送するデータを表し，Pct はパケット情報，Cnt はカウンタ情報，Ptn はビットパターンを表すビット列，Req は外部からのリクエスト信号，Tbl はフラッディング IP アドレスの集合を表す．図 15 より，それぞれのサブモジュールで一連の処理を行うアクション系列はたかだか 3 アクションで構成されている．各サブモジュールはそのアクション系列を実行したあと初期状態に戻る．各サブモジュールは独立に動作しており，パイプライン処理が可能である．原則 1 つのアクションを実行するために必要なクロック数は 1 である．よってクロック周波数が 12.45 MHz で，アクション数が最大 3 なので，毎秒約 415 万パケット（12.45（MHz）÷ 3）を処理することができる．ただし，実際にはパケット分割処理や，カウント処理などにおいて付加的な作業が追加される場合もある．しかし，アクション系列が 10 アクションで構成されていても，毎秒 100 万パケット以上を処理することができる．

4.4 ネットワーク環境に対応した閾値の決定

IP フラッディング検出モジュールも SYN フラッド

検出モジュールも取得したIPアドレスを分割し，各々のカウンタモジュールに渡してカウント処理を行っている．カウンタモジュールは担当するビットパターンの出現数がある閾値を超えた段階でフラッディングと見なす．

ここでは，毎秒100万パケットが流れるような環境を考える．このような環境では，毎秒1,000パケット増加するといった小規模なフラッディングは，IPフラッディングとして扱わない．一方，トラフィックの5%から10%程度（毎秒5万から10万パケット）のフラッディングが発生すれば，ネットワークが混雑し障害の原因となる．ここでは，このような量のIPフラッディングを検出することを考える．

あるIPアドレスに対して毎秒10万パケットのフラッディングパケットが送られるとする．パーティションサイズを4とし，8個のIP_COUNTERモジュールを使うとすると，それぞれのIP_COUNTERに毎秒約6.3万（100万 ÷ 16）のパケットが入力される．パケットが持つIPアドレスには偏りがあると考えられる．いま，IPアドレスのビットパターンに最大で50%の変動があると仮定すると，各IP_COUNTERには毎秒3.1〜9.5万（6.3万 ±50%）のパケットが入力される．このとき，IP_COUNTER中の各BitCounterの閾値を毎秒9.8万に設定する．このようにすると，ある特定のIPアドレスに対するパケットが毎秒10万パケットを超えた場合，そのIPアドレスのパケット総数は，IPアドレスのビットパターンの偏りにかかわらず毎秒9.8万を超える．このことから，パーティションサイズが4の8個のIP_COUNTERでこのIPフラッディングを検出することができる．

なお，同じモジュールで毎秒6,000パケットのフラッディングを検出したい場合，パーティションサイズが4のIP_COUNTERモジュールでは各カウンタに最低でも毎秒3.1万以上のパケットが入力されるのでこの構成では適当な閾値を設定できない．そこで，パーティションサイズを8とし，4個のIP_COUNTERを利用すると，1個のIP_COUNTERあたり毎秒入力されるパケット数は，3,906（100万 ÷ 256）となり，ビットパターンに50%の偏りがあったとしても，$3{,}906 + 50\% = 5{,}940$ となり，6,000を越えないので，閾値を6,000程度に設定すると，ビットパターンの偏りにかかわらずIPフラッディングを検出することができる．ただしこの場合，IP_COUNTER中のBitCounterの総数は $256 \times 4 = 1{,}024$ 個となり，回路サイズがBitCounterの総数に応じて増大する．

監視を行いたいネットワークに対して，N を1秒間に流れるパケット数，D をIPアドレスのビットパターンの偏り，F を検出したい1秒あたりのパケット数とする．N，D，F から自動的にパーティションサイズとIP_COUNTERの数を決めることができる．また，パーティションサイズとIP_COUNTERの数から，IP_COUNTERモジュールに対するFPGA回路を導出することができる．一方，文献3)ではインターネットバックボーンを流れるSYN+ACK，ACKパケットの量は最大で5%程度であると述べられている．このことから，SYN-FLOODモジュールは我々のツールを用いることで自動的に作成することができる．

5. 考　　察

ネットワークモニタをソフトウェアで実現した場合，数百Mbpsのネットワークに適用できる[19),20)]．本論文では，バックボーンなど10 Gbps程度の高速ネットワークを流れるパケットをリアルタイムに監視することを目標としている．文献15)から平均のパケットサイズは400 byte程度であるので，約毎秒300万パケットを処理できる性能がネットワークモニタに必要となる．提案手法で合成したIPフラッディングモジュール，SYNフラッドモジュールは，ともに12.45 MHzで動作し，処理のステップ数も3であるので，毎秒約415万パケットを処理することができ，目標性能を満たしている．さらに，動作速度を向上させる方法として，一度に多くのモジュール間でデータの受渡しが行われる部分にダミーモジュールを挟むことによって，FPGA回路が動作するクロック周波数を向上させることができる．

Snortなどソフトウェアで実現されたネットワークモニタの場合，規則を追加することにより，様々な項目を検出することがでる．提案手法では，各項目に対する規則を1つのEFSMとして記述し，他の項目を検出するEFSMと並列動作させることで，様々な項目を同時に検出することができる．

たとえば，Snortでは Land 攻撃を検出することができる．Land攻撃とは，プロトコルがtcpでSYNフラグが1になっており，パケットの送信元IPアドレスと送信先IPアドレスが等しいパケットを送出し，システムを誤動作させる攻撃である．Snortでは，これに対応する規則に基づいて，規則に一致するパケットを受け取った場合に警告メッセージを出力する．提案手法では，3章で述べたGET_PACKETモジュールからSYNフラグおよびIPアドレスをマルチランデブにより受け取り，比較した結果が等しければ報告するようなEFSMを記述することで，Land攻撃に対

応することができる．

Snort などのソフトウェアベースのネットワークモニタでは，規則の数が増加すると，各パケットにそれらの規則を順番に適用するため，扱うことができる最大トラフィックが減少することが予想される．一方，提案手法では，各規則を並列に適用できるため，速度低下を最小限に抑えることができる．

6. あとがき

本論文では，ネットワークモニタのハードウェア化について提案・実装を行った．SYNOPSYS 社の論理合成ツール FPGA Express を用いて，FPGA 回路の合成を行い，回路の面積とその速度を計測した．その結果，面積，速度ともに実用上問題ないことが確認できた．提案手法では，ネットワークモニタが持つパラメータ（パーティションサイズ，閾値）に基づいて，自動的に回路を合成する．

本手法ではネットワークモニタをモデル化する際に，同期通信を容易に表現可能であり，並行モデルを記述するのに適している並行同期 EFSM を適用した．これは，提案するネットワークモニタは上述の並行同期 EFSM の利点に特化して記述しているためである．したがって，他のモデルで仕様を記述した場合，原理的には類似の仕様が記述可能であるが，並行同期 EFSM を適用した場合と比較してかなり記述が煩雑になると考えられる．

今後は，様々なモニタ項目に対応する基本モジュールを作成し，ネットワークモニタ作成の支援ツールとして利用できるようにする予定である．

謝辞 本研究の一部は，（株）半導体理工学研究センター（STARC）との共同研究によるものである．本研究を進めるにあたり，適切な御助言をいただいた小澤時典（STARC），伊藤雅樹（日立製作所），東明浩（富士通），吉田久人（松下電器産業）の各氏に感謝する．

参考文献

1) Tanenbaum, A.S.: *Computer Networks*, Third Edition, Prentice-Hall Inc. (1996).
2) Garber, L.: Denial-of-Service Attacks Rip the Internet, *Proc. IEEE Computer*, pp.12–17 (2000).
3) Moore, D., Voelker, G.M. and Savage, S.: Inferring Internet Denial-of-Service Activity, *USENIX Security Symposium* (2001).
4) Paxson, V.: Bro: A System for Detecting Network Intruders in Real-Time, *Computer Networks*, Vol.31, No.23–24, pp.2435–2463 (1999).
5) Park, K. and Kee, H.: On the Effectiveness of Route-Based Packet Filtering for Distributed DoS Attack Prevention in Power-Law Internets, *Proc. ACM SIGCOMM2001*, pp.15–26 (2001).
6) Mansfield, G., et al.: Towards Trapping Wily Intruders in the Large, *Computer Networks*, Vol.34, pp.659–670 (2000).
7) Claffy, K., Miller, G.J. and Thompson, K.: The nature of the beast: recent traffic measurements from an Internet backbone, *Proc. INET'98* (1998). http://www.caida.org/outreach/papers/1998/Inet98/
8) 八木 哲，小倉 毅，川野哲生，丸山 充，高橋直久：メタモニタ：適応型ネットワークトラヒック観測機構，情報処理学会論文誌，Vol.41, No.2, pp.444–451 (2000).
9) Ditta, Z.D., Cox Jr, J.R. and Parulkar, G.M.: Design of the APIC: A High Performance ATM Host-Network Interface Chip, *Proc. IEEE INFOCOM'95*, pp.179–187 (1995).
10) Yasumoto, K., Kitajima, A., Higashino, T. and Taniguchi, K.: Hardware Synthesis from Protocol Specifications in LOTOS, *Proc. Joint Int. Conf. on 11th Formal Description Techniques and 18th Protocol Specification, Testing, and Verification* (*FORTE/PSTV'98*), pp.405–420 (1998).
11) Katagiri, H., Yasumoto, K., Kitajima, A., Higashino, T. and Taniguchi, K.: Hardware Implementation of Communication Protocols Modeled by Concurrent EFSMs with Multi-Way Synchronization, *37th IEEE/ACM Design Automation Conference* (*DAC-2000*), pp.762–767 (2000).
12) Apisdorf, J., Claffy, K. and Thompson, K.: OC3MON: Flexible, Affordable, High-Performance Statistics Collection, *Proc. INET'97* (1997). http://www.isoc.org/isoc/whatis/conferences/inet/97/proceedings/F1/F1_2.HTM
13) SYNOPSYS Inc. http://www.synopsys.com/
14) ISO: Information Processing System, Open Systems Interconnection, LOTOS — A Formal Description Technique Based on the Temporal Ordering of Observational Behavior, ISO 8807 (1989).
15) WIDE Project: Packet traces from WIDE backbone. http://tracer.csl.sony.co.jp/mawi/
16) Sekar, R., Guang, Y., Verma, S. and Shanbhag, T.: A High-Performance Network Intrusion Detection System, *ACM Conference on Computer and Communications Security*,

pp.8–17 (1999).
17) Kato, T., Ogishi, T., Idoue, A. and Suzuki, K.: Design of Protocol Monitor Emulating Behaviors of TCP/IP Protocols, *Proc. IFIP 10th Int. Workshop on Testing of Communicating Systems* (*IWTCS'97*), pp.416–431 (1997).
18) Snort. http://www.snort.org/
19) Gokhale, M., Dubois, D., Dubois, A., Boorman, M., Poole, S. and Hogsett, V.: Granidt: Towards Gigabit Rate Network Intrusion Detection Technology, *Proc. 12th Int. Conf. on Field Programmable Logic and Applications* (*FPL2002*), LNCS2438, pp.393–403 (2002).
20) Sekar, R., Guang, Y., Verma, S. and Shanbhag, T.: A High-Performance Network Intrusion Detection System, *ACM Conference on Computer and Communications Security*, pp.8–17 (1999).

(平成 14 年 8 月 23 日受付)
(平成 15 年 4 月 3 日採録)

桐村　昌行（正会員）

平成 12 年大阪大学基礎工学部情報工学科卒業．平成 14 年同大学大学院博士前期課程修了．現在，三菱電機株式会社勤務．在学中，実時間システムや分散システムの仕様記述・実装に関する研究に従事．

高本　佳史

平成 13 年大阪大学基礎工学部情報科学科卒業．現在，同大学大学院博士前期課程在学中．実時間システムや分散システムの仕様記述・実装に関する研究に従事．

森　　亮憲（正会員）

平成 10 年大阪大学基礎工学部情報工学科卒業．平成 15 年同大学大学院博士後期課程修了．現在，独立行政法人通信総合研究所．博士（工学）．通信プロトコルの設計，検証および適合性試験手法等の研究に従事．

安本　慶一（正会員）

平成 3 年大阪大学基礎工学部情報工学科卒業．平成 7 年同大学大学院博士後期課程退学後，滋賀大学経済学部助手．現在，奈良先端科学技術大学院大学情報科学研究科助教授．博士（工学）．平成 9 年モントリオール大学客員研究員．分散システムおよびマルチメディア通信システムの実装法およびミドルウェアに関する研究に従事．

中田　明夫（正会員）

平成 4 年大阪大学基礎工学部情報工学科卒業．平成 9 年同大学大学院博士後期課程修了．現在，大阪大学大学院情報科学研究科助教授．博士（工学）．実時間システムや分散システムの仕様記述と検証法，プロセス代数，時相論理等の研究に従事．

東野　輝夫（正会員）

昭和 54 年大阪大学基礎工学部情報工学科卒業．昭和 59 年同大学大学院博士後期課程修了．現在，大阪大学大学院情報科学研究科教授．工学博士．分散システム，通信プロトコル等の研究に従事．電子情報通信学会，ACM 各会員．IEEE Senior Member．