平成29年度　卒 業 論 文

和文題目

不正パケットの高速な検出を実現する簡易認証方式の提案と評価

英文題目

Proposal and Evaluation of Simple Authentication Method that Detects Invalid Packets Fast

情報工学科　渡邊研究室

(学籍番号: 140441043)

鴨下　友馬

提出日: 平成30年2月9日

名城大学理工学部

概要

DoS 攻撃対策の一例として，共通鍵を事前に共有している場合は HMAC (Hash-based Message Authentication Code) を用いたパケット検証を利用することができる．DoS 攻撃を防御するためには不正パケットを高速に廃棄する必要があるが，このパケット検証にはパケット長が長いとパケット検証に要する時間が大きいという課題がある．

そこで，通信に用いる共通鍵とシーケンス番号のみを用いた簡易認証方式を追加することにより，パケット検証を高速に行う．送信側は共通鍵とシーケンス番号から簡易ハッシュ値を生成し，パケット内に付加する．受信側は付加情報を最初に検証することにより，不正パケットのほとんどを高速に検出することが可能となる．

本論文では，実験とシミュレーションによる評価を行い，僅かな処理を追加するだけでパケット検証時間を大幅に検証できることを示した．

Abstract

As an example of measures to a DoS attack, a packet verification method using a HMAC (Hash-based Message Authentication Code) can be used when a common key is shared in advance. In order to prevent a DoS attacks, it is necessary to discard invalid packets fast. However, this verification method has a problem that the time for packet verification becomes long if the packet's length is long.

Therefore, I include a simple authentication method using only a common key for communication and a sequence number, and packet verification can be performed fast. The sender generates a simple hash value from the common key and the sequence number and adds it to the packet. By verifying this additional information first, the receiver can detect most of the invalid packet fast.

In this paper, I evaluate by experiment and simulation, and show that the time for packet verification can be significantly reduced by only including a little processing.

目 次

第1章　はじめに

携帯電話やスマートフォンなどの移動通信端末の普及や，IoT (Internet of Things) の発展に伴い，ネットワークを利用する機会がますます増加している．これに伴い，ネットワークセキュリティを脅かす様々なサイバー攻撃が問題となっている．中でも，リプレイ攻撃 (Replay Attack) と DoS 攻撃 (Denial of Service Attack) は大きな脅威である [1]．リプレイ攻撃は，受信側からすると正規のパケットとの見分けがつかないため，特にログイン情報に関わるパケットの場合は不正ログインを許可することになり，大きな問題となる．IPsec (Security Architecture for the Internet Protocol) [2] では，パケット内に付与されたシーケンス番号を用いて検証を行う．具体的には，受信側がリプレイ防御ウィンドウと呼ばれるビットマスクを用いて受信を許可するシーケンス番号の範囲を決定することで，リプレイ攻撃パケットを検出する（リプレイ攻撃チェック）．

一方，DoS 攻撃は，大量のデータを処理するサーバ類では特に脅威となる攻撃である．DoS 攻撃では，攻撃者の追跡ができないようにするため，送信元を偽造する場合が多い．そのため DoS 攻撃対策の例としては，最初に軽いやり取りを行うことで，通信相手が確かに存在することを確認してから重いやり取りに移行するという方法がある．暗号化通信のように，既に共通鍵を共有している場合は，HMAC (Hash-based Message Authentication Code) を用いてパケットを検証するのが一般的である（MAC 認証）．IPsec においても，リプレイ攻撃チェックに続いて MAC 認証を行うように定められている．しかし，MAC 認証にはパケット長が長いと，不正パケットの検出にかかる処理時間が長くなるという問題がある．特に，多くのパケットを高速に処理する必要があるサーバには DoS 攻撃耐性が求められるため，少しでも早くパケット検証を行えることが望ましい．

そこで，パケット内に 8bit 分の新しいチェック用フィールドを定義し，これを最初に検証する簡易認証方式を提案する．提案方式では，送信側は共通鍵とシーケンス番号から 8bit の簡易ハッシュ値を生成してパケット内に付加し，受信側はこの付加情報を最初に検証する．ほとんどの不正パケットは簡易認証で破棄され，簡易認証を通過した場合も MAC 認証で最終的に必ず破棄される．簡易認証は処理が軽いので，正規の受信処理にほとんど影響を与えることはない．本論文では，提案方式を用いたパケット検証の実験を行い，パケット検証に要する処理時間を計測した結果を示す．また，この実測値を用いたシミュレーションを行い，パケット検証に要する処理時間の総合的な評価を行った結果を示す．最後に，IPsec と NTMobile (Network Traversal with Mobility) に提案方式を適用した場合のフォーマット例を示す．

以後，2 章では IPsec と NTMobile のパケット検証について説明する．3 章では提案方式を用いたパケット検証について説明し，4 章では提案方式を実装したテストプログラムの概要について示す．5 章ではテストプログラムの動作検証および処理時間の測定，シミュレーションから評価を行い，最後に 6 章でまとめる．

第2章　既存技術

本章では，IPsec (Security Architecture for the Internet Protocol) のセキュリティプロトコルである ESP (Encapsulating Security Payload) [3] と，移動透過性と通信接続性の両者を同時に実現する技術である NTMobile (Network Traversal with Mobility) について，概要およびパケット検証処理を説明する．

2.1　ESP

2.1.1　概要

ESP は，パケットの機密性 *1 および完全性 *2 を保証するセキュリティプロトコルである．機密性の確保にはパケットの暗号化を利用し，完全性の確保には ICV (Integrity Check Value) による完全性チェックを利用する．ICV はその役割という観点では MAC (Message Authentication Code) と同義である．したがって，以下では ICV をチェックする処理を「MAC 認証」と呼ぶ．

図 1 に，ESP トランスポートモードのパケットフォーマットを示す．encrypted は暗号化範囲である．authenticated は認証範囲であり，MAC 認証の対象となる．ESP header には 32bit のシーケンス番号が含まれている．ICV は 128bit であり，パケットの認証範囲と共通鍵を用いて HMAC-MD5 により生成し，その結果を ICV フィールドに付与する．

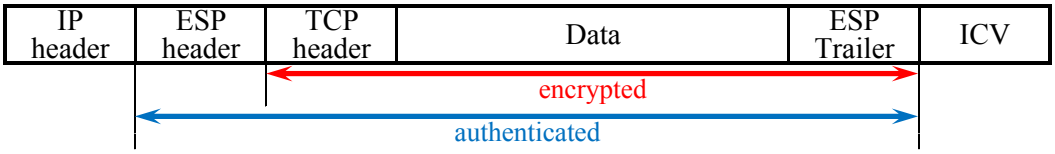

図 1　ESP のパケットフォーマット

パケット検証は，リプレイ攻撃チェック，MAC 認証の順に行う．不正パケットであると判定した場合にはその時点で破棄を決定し，以降の処理は行わない．MAC 認証まで成功した場合は正規のパケットとみなし，リプレイ防御ウィンドウの更新（受信許可範囲の変更処理）を行う．最後に，パケットを復号して受信処理に入る．

*1 アクセスを認可された者だけが情報にアクセスできることを確実にする性質 [4]
*2 情報および処理方法が正確であること，および完全であることを保護する性質 [4]

2.1.2　パケット検証処理

リプレイ攻撃チェックでは，リプレイ防御ウィンドウと呼ばれる 32bit 以上のビットマスクを用いて受信を許可するシーケンス番号の範囲を決定することで，リプレイ攻撃パケットを検出する．リプレイ攻撃では正規のパケットを攻撃に利用するため MAC 認証では検出できず，必須の処理である．

図 2 に，リプレイ攻撃チェックの概要を示す．ここで，リプレイ防御ウィンドウのサイズは 32bit である．シーケンス番号が 116 のパケットまで受信済みであるとすると（図 2 (a)），リプレイ防御ウィンドウ内の未受信のパケット（シーケンス番号 87），およびシーケンス番号 117 以降のパケットは受信を許可する．つまり，受信済みのシーケンス番号 110 のパケットを再度受信した場合は不正パケットと判定される（図 2 (b)）．また，シーケンス番号 120 のパケットを受信した場合は受信許可範囲が変わる（図 2 (c)）．このとき，未受信のシーケンス番号 117 のパケットは受信可能（図 2 (d)）だが，シーケンス番号 87 のパケットはリプレイ防御ウィンドウから外れるため不正パケット扱いとなる（図 2 (e)）[5]．これは，未受信のパケットとはいえ，最新の受信済みシーケンス番号 120 と比較して古すぎるシーケンス番号のパケットをこの段階で受信するのは不自然であり，不正パケットと考えるべきであるからである．

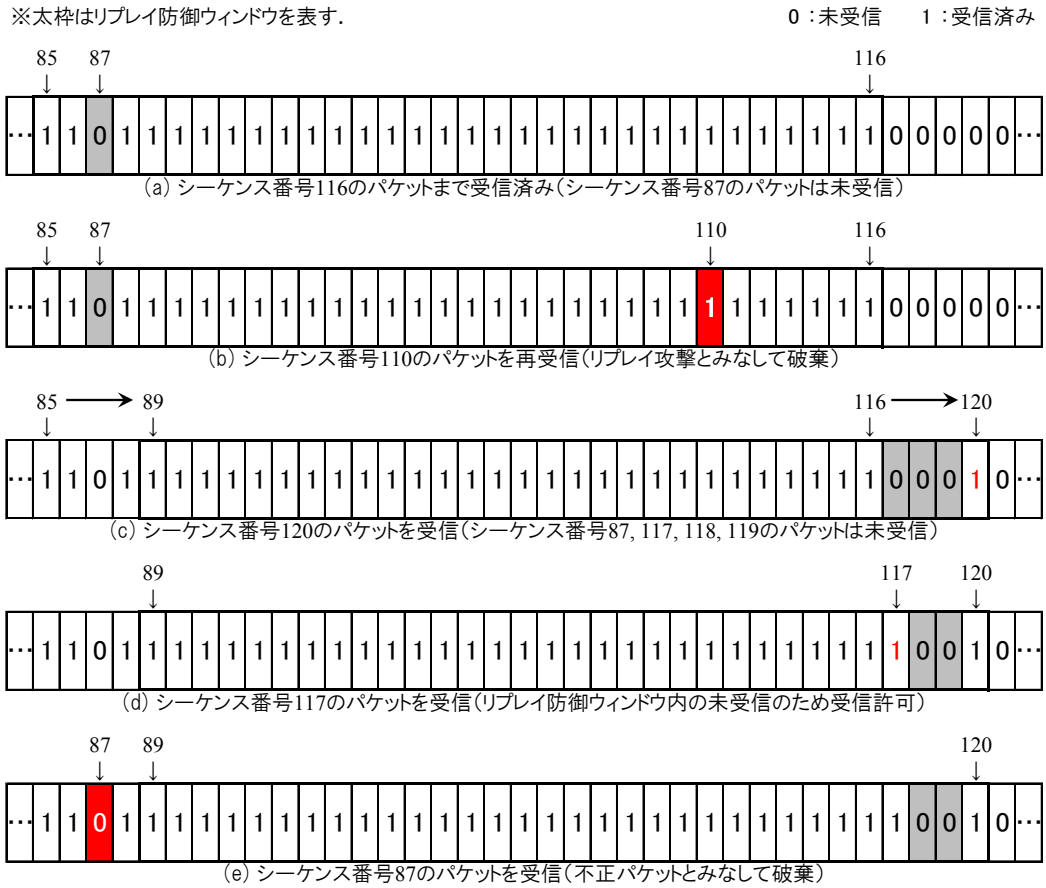

図 2　リプレイ攻撃チェックの概要

リプレイ攻撃チェックの段階では，受信許可するか破棄するかを決定する部分のみを行う．これは，この時点では正規のパケットであるか否かが確定していないためである．

リプレイ攻撃チェックを通過した受信パケットは，続けて MAC 認証を行う．MAC 認証では，受信パケットの認証範囲と共通鍵を入力として，暗号学的ハッシュ関数 HMAC-MD5 により 128bit の認証コードを生成する．これと受信パケットの ICV を比較し，一致していれば受信を許可する．MAC 認証を通過した受信パケットは，正規のパケットとみなしてリプレイ防御ウィンドウ更新処理を行う．

リプレイ防御ウィンドウ更新処理では，次の受信パケットに対してリプレイ攻撃チェックを行うために，リプレイ防御ウィンドウを更新する処理である．具体的には，図 2 の (c) や (d) のように受信を許可した場合に，そのパケットを受信したことを表すフラグを立てたり，ウィンドウをスライドさせたりといった処理を行う．

2.2 NTMobile

2.2.1 概要

NTMobile は，通信中にネットワークが切り替わっても通信を継続できる移動透過性と，ネットワーク環境に関わらず通信を開始することができる通信接続性の両者を同時に実現する技術である [6] [7]．NTMobile 端末は，通信開始時に DC (Direction Coordinator) からの経路生成指示によってトンネル経路を生成し，以後のすべての通信は仮想アドレスのパケットを実アドレスでカプセル化する．

NTMobile には ESP と同様に，パケットの機密性および完全性を確保し，送信元の認証を行う機能が備わっている．機密性の確保にはパケットの暗号化を利用し，完全性の確保および送信元の認証にはメッセージ認証コード MAC (Message Authentication Code) を利用する．この他，ESP と同様のリプレイ攻撃チェックも備わっている．これにより，エンドツーエンドのセキュリティを実現することができる．

2.2.2 NTMobile のパケット検証

図 3 に NTMobile のパケットフォーマットを示す．encrypted は暗号化範囲である．authenticated は認証範囲であり，MAC 認証の対象となる．NTM header には NTMobile の通信に関わるデータが含まれており，32bit のシーケンス番号もここに含まれる．MAC は 128bit であり，パケットの認証範囲と共通鍵を用いて HMAC-MD5 により生成し，その結果を MAC フィールドに付与する．

パケット検証は基本的に ESP と同様で，リプレイ攻撃チェック，MAC 認証の順に行う．不正パケットであると判定した場合にはその時点で破棄を決定し，以降の処理は行わない．MAC 認証まで成功した場合は正規のパケットとみなし，リプレイ防御ウィンドウの更新（受信許可範囲の変更処理）を行う．最後に，パケットを復号して受信処理に入る．なお，現状の NTMobile では，リプレイ攻撃チェックは検討段階であるため未実装である．

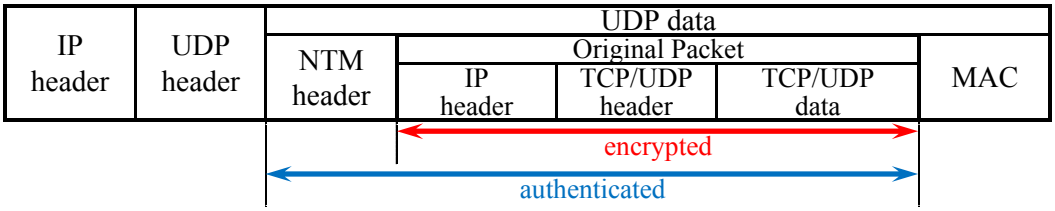

図 3　NTMobile のパケットフォーマット

NTMobile には，RS (Relay Server) と呼ばれるパケット中継装置がある．RS では大量のパケットを中継するため，不正パケットを可能な限り短時間で検出する機能が求められる．しかし，MAC 認証はパケット長が長いほど処理時間が長くなるため，現状のパケット検証では不正パケットを高速に検出することができないという問題がある．

第3章　提案方式

本章では，提案する簡易認証の概要，および簡易認証に用いるハッシュ関数について説明する．

3.1　簡易認証とパケット検証処理の概要

簡易認証は，不正パケットを高速に検出するための処理である．処理に要する時間が MAC 認証と比較して短いため，パケット検証の最初に行うことで MAC 認証の補助的役割を担うことができる．提案方式を適用する際は，簡易認証に係るフィールド（8bit）をパケット内に追加する必要がある．ただし，フィールドを追加する場所は，暗号化範囲外かつ認証範囲内である必要がある．

送信側は，通信に用いる共通鍵とシーケンス番号を入力として 8bit のハッシュ値（以下，簡易ハッシュ値）を生成し，パケットに付与する．このときのハッシュ関数は，演算時間の短い FNV-1(32bit) を使用する [8]．受信側は，最初に簡易ハッシュ値を計算し，受信パケットに格納された値と比較する．これらの値が不一致であれば不正パケットであると判定して破棄し，一致していればリプレイ攻撃チェックに進む．以後の処理は既存のパケット検証処理と同様で，リプレイ攻撃チェック，MAC 認証の順に行い，正規のパケットであればリプレイ防御ウィンドウの更新（受信許可範囲の変更処理）を行う．ここまでがパケット検証に関わる処理であり，その後は復号処理や正規の受信処理を行う．

以上を踏まえて，図 4 にパケット検証処理の順序を示す．

3.2　簡易認証に用いるハッシュ関数

簡易ハッシュ値の生成に用いるハッシュ関数 FNV-1(32bit) について説明する．

入力を $M = \{m_1, m_2, \cdots, m_n\}$ とすると，出力 $h(M)$ は式 (3.1) のようになる．

$$h(M) = [[\cdots\{\{((Offset \times Prime) \oplus m_1) \times Prime\} \oplus m_2\}\cdots] \times Prime] \oplus m_n \tag{3.1}$$

ここで，$Offset$, $Prime$ は定数であり，それぞれ $Offset$ =2,166,136,261, $Prime$ =16,777,619 である．ハッシュ関数 FNV-1(32bit) には，以下のような特徴がある．

性質 1　入力 M は 8bit 整数のベクトルを想定している．すなわち，$m_1, m_2, \cdots, m_n$ はいずれも 8bit 整数である．

性質 2　入力 M において m_n が近接した値のとき，$h(M)$ の値が近くなることがある．

性質 3　出力は 32bit 整数である．

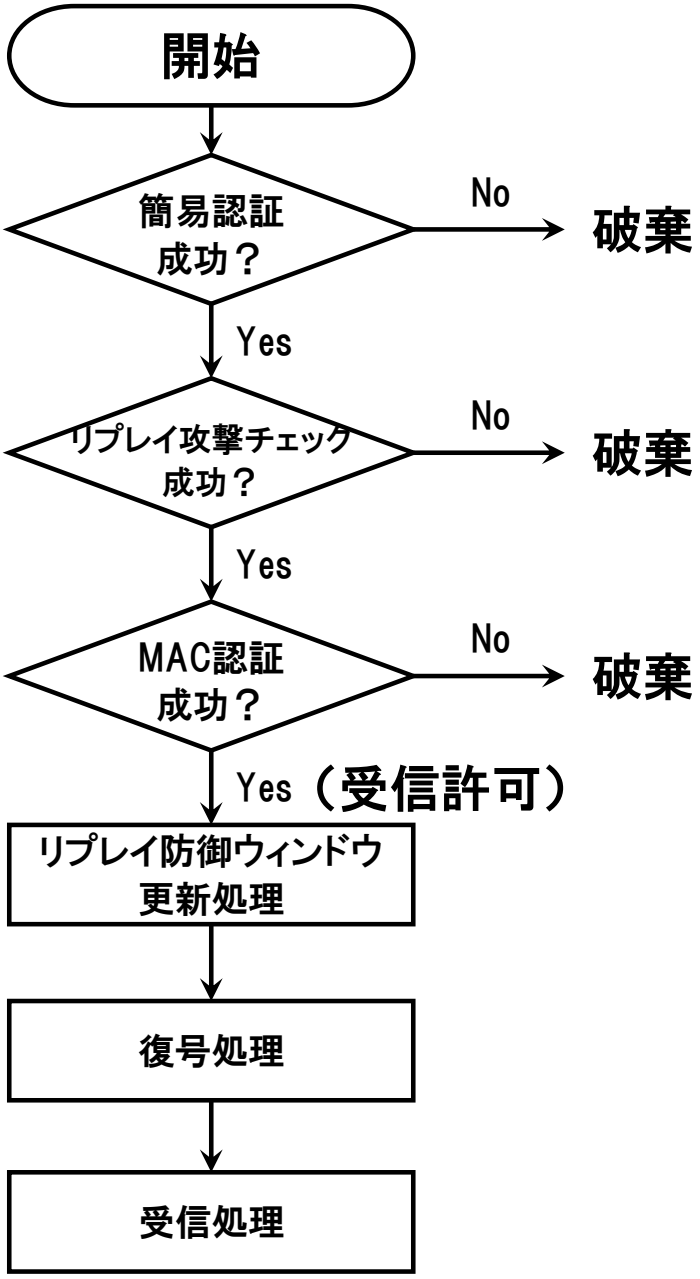

図 **4**　パケット検証処理の順序

第4章　実装

本章では，提案方式を実装したテストプログラムの概要について示す．今回の実験は NTMobile に適用した場合を想定して行うため，プログラミング言語は C を用いるなど，基本的な仕様は NTMobile に合わせる．図 3 の NTM header 以降の部分を生成して「簡易認証」,「リプレイチェック」,「MAC 検証」,「リプレイ防御ウィンドウ更新処理」を行い，それらの処理時間を測定する．NTMobile では，簡易認証に係るフィールド（8bit）を図 3 の NTM header に追加する．

4.1　簡易ハッシュ値の生成

ハッシュ関数 FNV-1(32bit) の入力は，共通鍵とシーケンス番号である．NTMobile では，共通鍵は 128bit 整数であり，シーケンス番号は 32bit 整数である．したがって，入力値の合計は 160bit であり，FNV-1(32bit) の性質 1 より $M = \{m_1, m_2, \cdots, m_{20}\}$ となる．

ここで，共通鍵を $K = \{k_1, k_2, \cdots, k_{16}\}$，シーケンス番号を $N = \{n_1, n_2, n_3, n_4\}$ と表す（$k_i\,(i = 1, 2, \cdots, 16), n_j\,(j = 1, 2, 3, 4)$ はいずれも 8bit 整数）と，セッション継続中は K は変化せず，N はパケットを送信する度に 1 ずつ増加する．このとき，m_n に n_4 を配置すると，FNV-1(32bit) の性質 2 より，隣接するシーケンス番号のパケットの $h(M)$ は近接することがあり，シーケンス番号の増分から $h(M)$ の予測が容易となる．したがって，$M = \{N, K\} = \{n_1, \cdots, n_4, k_1, \cdots, k_{16}\}$ とする．このときのハッシュ関数は，式 (3.1) より，

$$h(M) = \{\{\{\{((\cdots((Offset \times Prime) \oplus n_1)\cdots) \times Prime) \oplus n_4\} \times Prime\} \oplus k_1\} \cdots\} \oplus k_{16} \qquad (4.1)$$

となる．$h(M)$ は FNV-1(32bit) の性質 3 より 32bit となるが，NTM header の簡易認証に係るフィールドは 8bit であるため，$h(M)$ の下位 8bit を簡易ハッシュ値とする．

4.2　簡易認証の手順

受信側は，受信パケットの NTM header からシーケンス番号を取得し，これと共通鍵を入力として簡易ハッシュ値を生成する．生成した簡易ハッシュ値と，受信パケットの NTM header に付与された簡易ハッシュ値を比較し，一致していれば簡易認証に成功したものとする．一致していなければ簡易認証に失敗したものとし，不正パケットと判定する．

4.3 リプレイ攻撃チェックの手順

現状の NTMobile では，リプレイ攻撃チェックは検討段階であるため未実装である．しかし，パケット検証実験においてリプレイ攻撃チェックは必須となるため，リプレイ攻撃チェックの実装も行った．2.1.2 節に示したリプレイ攻撃チェックは，RFC2401（IPsec の旧バージョンに関する RFC）にサンプルプログラムが記述されている [9]．ただし,「リプレイ攻撃チェック」と「リプレイ防御ウィンドウ更新処理」を合わせたものとなっているため，これらの処理を分割して実装した．

リプレイ攻撃チェックでは，受信側は，受信パケットの NTM header からシーケンス番号を取得して 2.1.2 節のような処理を行い，受信許可と判定した場合はリプレイ攻撃チェックに成功したものとする．受信拒否と判定した場合はリプレイ攻撃チェックに失敗したものとし，不正パケットと判定する．

4.4 MAC の生成と MAC 認証の手順

MAC は HMAC-MD5 により生成して図 3 の MAC フィールドに付与する．受信側も同様に MAC を生成し，受信パケットの MAC と比較する．一致していれば MAC 認証に成功したものとする．一致していなければ MAC 認証に失敗したものとし，不正パケットと判定する．

この処理は現状の NTMobile に実装済みであり，テストプログラムではこれを引用して使用した．

4.5 リプレイ防御ウィンドウ更新処理の手順

4.3 節にて説明した通り，RFC2401 のサンプルプログラムから「リプレイ防御ウィンドウ更新処理」の部分を抽出して実装した．

第5章　評価

本章では，提案方式を実装したテストプログラムの動作検証および処理時間の測定結果を示し，その結果を用いたシミュレーションから提案方式の性能評価を行う．

5.1　動作検証

表 1 に実験環境を示す．この環境において，図 3 の NTM header の長さを 36Byte，Original Packet の長さを 1,000Byte としてパケット検証処理を行い，正常に動作することを確認した．

表 1　実験環境

	ホストマシン	仮想マシン
OS	Windows 7 64bit	Ubuntu 14.04 32bit
Linux カーネル	–	3.13.0-116-generic
CPU	Intel Core i7-2600 3.40GHz	1Core 割り当て
Memory	8.00GB	1.00GB 割り当て

5.2　パケット検証処理時間

簡易認証に要する時間を t_s，リプレイ攻撃チェックに要する時間を t_r，MAC 認証に要する時間を t_m，リプレイ防御ウィンドウ更新処理に要する時間を t_u とする．5.1 節の条件でパケット検証処理を 100,000 回実行した際の，処理時間の平均値を表 2 に示す．

表 2　パケット検証処理時間

$t_s[\mu s]$	$t_r[\mu s]$	$t_m[\mu s]$	$t_u[\mu s]$
0.536	0.414	3.835	0.561

5.3 不正パケットの検証処理時間の評価

確率の観点からシミュレーションを行い，提案方式を用いた場合の不正パケット検証処理時間を評価する．ある不正パケットが簡易認証で破棄される確率を $\overline{P_s}$，リプレイ攻撃チェックで破棄される確率を $\overline{P_r}$，MAC 認証で破棄される確率を $\overline{P_m}$ とすると，

$$\overline{P_s}+\overline{P_r}+\overline{P_m}=1 \tag{5.1}$$

が成り立つ．図 5 に，不正パケット検出処理の評価図を示す．このとき，不正パケットの検証処理時間の平均値 E は式 (5.2) のようになる．

$$E=t_s\overline{P_s}+(t_s+t_r)\,\overline{P_r}+(t_s+t_r+t_m)\,\overline{P_m} \tag{5.2}$$

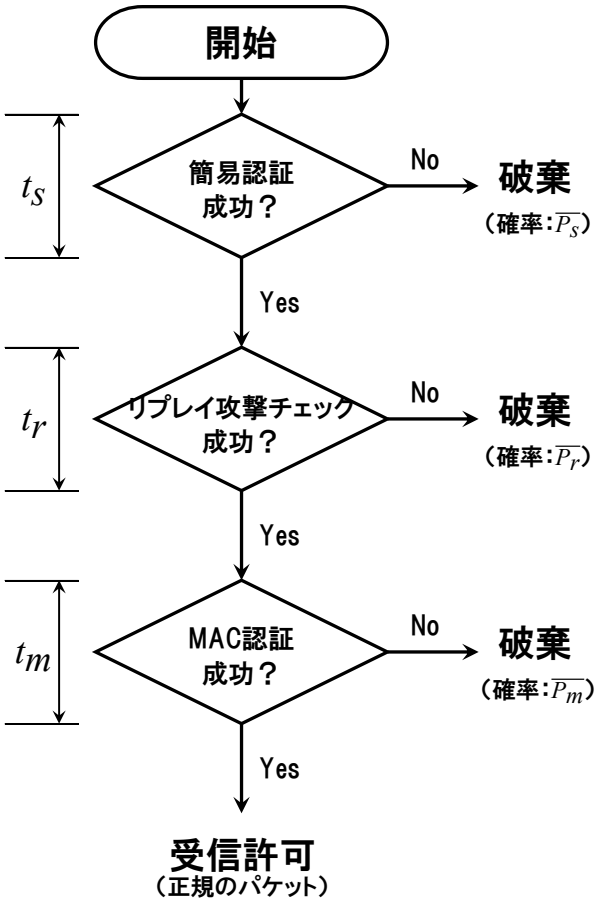

図 5　不正パケット検出処理の評価図

5.3.1 不正パケット検出率

ある不正パケットが簡易認証に成功する確率 P_s を算出する．前提として，不正パケットの簡易ハッシュ値の分布は一様であると仮定する．このとき，簡易ハッシュ値の長さを l_h[bit] とすると，

$$P_s = \frac{1}{2^{l_h}} \tag{5.3}$$

となる．よって，ある不正パケットが簡易認証で破棄される確率 $\overline{P_s}$ は，

$$\overline{P_s} = 1 - P_s \tag{5.4}$$

となる．

次に，ある不正パケットがリプレイ攻撃チェックに成功する確率 P_r を算出する．リプレイ攻撃チェックにおいて受信許可と判定されるシーケンス番号は，2.1.2 節で説明した通り，以下のいずれかの条件を満たす必要がある．

条件 1 最新のシーケンス番号より大きく，シーケンス番号の最大値以下であるもの

条件 2 リプレイ防御ウィンドウの範囲内のうち，未受信のシーケンス番号であるもの

ここで，シーケンス番号の長さを l_n[bit] とすると，シーケンス番号の最大値は $2^{l_n} - 1$ となる．したがって，検証処理開始時点での最新の受信済みシーケンス番号を n_l（図 2 で (c) から (d) に遷移する時点での n_l は 120）とすると，条件 1 を満たすシーケンス番号の数は，

$$\left(2^{l_n} - 1\right) - n_l \tag{5.5}$$

となる．一方，リプレイ防御ウィンドウのサイズを s_w，検証処理開始時点でのリプレイ防御ウィンドウ内の受信済みシーケンス番号の個数を $r\,(1 \leq r \leq \min(s_w, n_l))$（図 2 で (c) から (d) に遷移する時点での r は 29）とすると，条件 2 を満たすシーケンス番号の数は，

$$\min(s_w, n_l) - r \tag{5.6}$$

となる．受信許可と判定されるシーケンス番号の数は式 (5.5) と式 (5.6) の和であるから，

$$\left\{\left(2^{l_n} - 1\right) - n_l\right\} + \{\min(s_w, n_l) - r\} \tag{5.7}$$

となる．シーケンス番号の総数は 2^{l_n} であるから，P_r は，

$$P_r = \frac{\left\{\left(2^{l_n} - 1\right) - n_l\right\} + \{\min(s_w, n_l) - r\}}{2^{l_n}} \tag{5.8}$$

となる．ある不正パケットがリプレイ攻撃チェックで破棄されるとき，その不正パケットは簡易認証に成功している必要がある．したがって，ある不正パケットがリプレイ攻撃チェックで破棄される確率 $\overline{P_r}$ は，

$$\overline{P_r} = P_s\,(1 - P_r) \tag{5.9}$$

となる．

以上から，ある不正パケットがMAC認証で破棄されるとき，その不正パケットは簡易認証とリプレイ攻撃チェックの両方に成功している必要がある．したがって，ある不正パケットがMAC認証で破棄される確率を $\overline{P_m}$ とすると，

$$\overline{P_m} = P_s P_r \tag{5.10}$$

となる．なお，式(5.4)，式(5.9)，式(5.10)は，式(5.1)の条件を満たしている．

簡易ハッシュ値の長さ l_h，シーケンス番号の長さ l_n，リプレイ防御ウィンドウのサイズ s_w は定数である．したがって，$\overline{P_s}$ は定数である．一方，検証処理開始時点での最新の受信済みシーケンス番号 n_l，リプレイ防御ウィンドウ内の受信済みシーケンス番号の個数 r は，受信状況により変化する値である．したがって，$\overline{P_r}, \overline{P_m}$ は変数である．なお，式(5.9)，式(5.10)から，$\overline{P_r}, \overline{P_m}$ は P_r に依存し，$\overline{P_r}$ が小さいほど $\overline{P_m}$ は大きくなることがわかる．

5.3.2 実験結果に基づくシミュレーション結果

表 2の実測値を得た際の各種パラメータは，$l_h = 8, l_n = 32, s_w = 32$ である．また，n_l, r は受信状況により変化するため厳密に定義することはできないが，以下では $n_l = 1, r = 1$ としてシミュレーションを行う．なお，$n_l = 1, r = 1$ は「シーケンス番号1のパケットのみを受信した状態」であり，リプレイ攻撃チェックでは図 6に示した状態となっている．このとき，リプレイ攻撃チェックで破棄される確率 $\overline{P_r}$ は最小となるため，MAC認証で破棄される確率 $\overline{P_m}$ が最大となる．

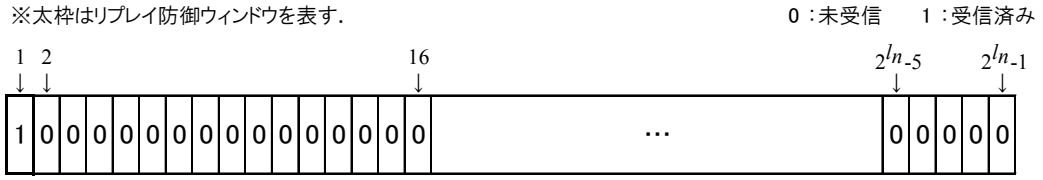

図 6　$n_l = 1, r = 1$ のときのリプレイ攻撃チェック

以上のパラメータと表 2の実測値 t_s, t_r, t_m を用いたとき，不正パケットの検証処理時間の平均値 E は，

$$E = 0.553\,[\mu\mathrm{s}] \tag{5.11}$$

となった．

一方，簡易認証を用いていない既存のパケット検証処理では，$l_h = 0, t_s = 0$ となる．その他のパラメータと実測値は同じものを用いたとき，不正パケットの検証処理時間の平均値 E は，

$$E|_{l_h=0, t_s=0} = 4.249\,[\mu\mathrm{s}] \tag{5.12}$$

となる．この結果から，提案手法により不正パケットの検証処理時間を最大1/8程度に短縮することができる．したがって，不正パケットによるDoS攻撃耐性が大きく向上することが期待できる．

5.4 正規のパケットの検証処理時間の評価

攻撃がない状態においては，パケット検証に常に簡易認証処理が加わるため負荷が増えることになる．そこで，この増加した負荷が全体の検証処理時間に与える影響を調査した．図 7 に，正規のパケット検証処理の評価図を示す．

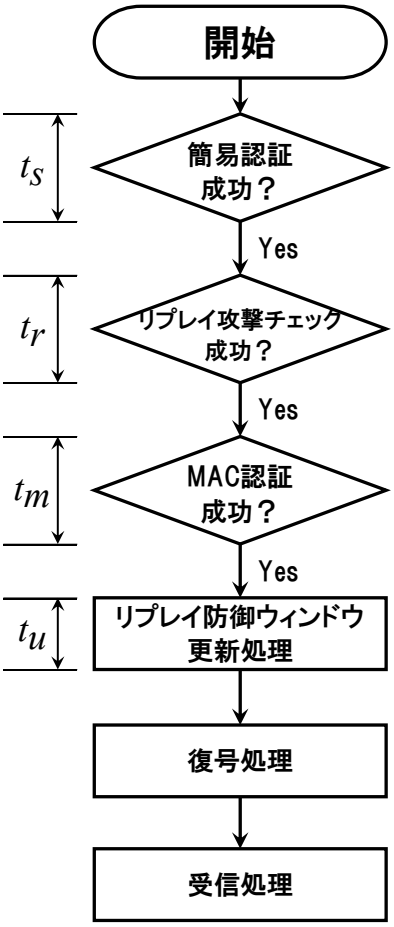

図 7　正規のパケット検証処理の評価図

簡易認証を用いていない既存のパケット検証処理における，正規のパケットの検証処理時間の理論値は，

$$t_r + t_m + t_u = 4.810[\mu\mathrm{s}] \tag{5.13}$$

となる．これに対して，提案手法における，正規のパケットの検証処理時間の理論値は，

$$t_s + t_r + t_m + t_u = 5.346[\mu\mathrm{s}] \tag{5.14}$$

となる．提案手法では t_s が加わるため，全体の検証処理時間は約 11%長くなることがわかる．しかし，この後の復号処理時間（MAC 認証の約 10 倍），さらには正規の受信処理時間を考慮すると，t_s の影響は無視できるほど小さいと言える．

5.5 簡易ハッシュ値の長さによる処理時間の比較

3.1 節に示した通り，提案手法における簡易ハッシュ値の長さ l_h は 8bit としている．本節では，この正当性を調査した結果を示す．

プログラムの制約上，l_h の最小値は 8bit である．l_h を 8bit, 16bit, 32bit のように変化させる．このときの $\overline{P_s}, \overline{P_r}, \overline{P_m}$ を，5.3.1 項に示した計算式からそれぞれ計算した結果を表 3 に示す．

表 3　簡易ハッシュ値の長さを変化させた場合の不正パケット検出率

l_h[bit]	$\overline{P_s}$	$\overline{P_r}$	$\overline{P_m}$
8	9.9609×10^{-1}	1.8190×10^{-12}	3.9063×10^{-3}
16	9.9998×10^{-1}	7.1054×10^{-15}	1.5259×10^{-5}
32	9.9999×10^{-1}	1.0842×10^{-19}	2.3283×10^{-10}

l_h が大きくなるほど，$\overline{P_s}$ は 1 に近付き，$\overline{P_r}, \overline{P_m}$ は 0 に極めて近づくため，不正パケットの検証処理時間の平均値 E は式 (5.2) より，

$$E \approx t_s \cdot 1 + (t_s + t_r) \cdot 0 + (t_s + t_r + t_m) \cdot 0 = t_s \tag{5.15}$$

となる．すなわち，E は簡易認証に要する時間 t_s の値にほぼ一致することになる．

5.1 節の条件で簡易認証を 100,000 回実行した際の，処理時間の平均値 t_s と，算出した E を表 4 に示す．ただし，$l_h = 8$ の場合については 5.3.2 項の結果を引用した．

表 4　パケット検証処理時間の比較

l_h[bit]	$t_s[\mu s]$	$E[\mu s]$
8	0.536	0.553
16	0.675	0.675
32	0.823	0.823

表 4 から，l_h が長くなると t_s が増加し，その結果 E も増加していく．このとき，表 3 から，t_s の増加量に対して $\overline{P_s}$ の変化は大差ないことがわかる．これは，簡易認証の効果が大きく上昇するわけではないということを意味する．したがって，簡易ハッシュ値の長さは 8bit が最適である．

第6章　まとめ

本稿では，共通鍵とシーケンス番号のみを用いた簡易認証方式を提案した．この提案方式を用いたパケット検証テストプログラムを作成し，Linux において正常に動作することを確認した．また，実験とシミュレーションにより，既存方式と比較して不正パケットの検証処理時間を最大 1/8 程度に短縮することが可能であるという点で，提案方式の有用性を示した．さらに，簡易認証に用いる簡易ハッシュ値の長さは 8bit が最適であることも示した．

今回はテストプログラムを用いた実験を行ったが，NTMobile に適用することでセキュリティの向上が見込めることがわかった．したがって，今後は提案方式を NTMobile に正式仕様として組み込む予定である．

謝辞

本研究を進めるにあたり，多大なるご指導を賜りました，指導教官である名城大学理工学部情報工学科　渡邊晃教授に心から感謝致します.

本研究を進めるにあたり，様々なご指導を賜りました，名城大学理工学部情報工学科　鈴木秀和准教授に深謝致します.

本研究を進めるにあたり，様々なご助言を賜りました，愛知工業大学情報科学部情報科学科　内藤克浩准教授に拝謝致します.

最後に，本研究を進めるにあたり，日頃から多くの有益なご意見を賜りました，渡邊研究室の皆様，鈴木研究室の皆様，NTMobile の研究に携わる皆様に感謝致します.

参考文献

[1] Rescorla, E. and Korver, B.: Guidelines for Writing RFC Text on Security Considerations, RFC 3552, IETF (2003).

[2] Kent, S. and Seo, K.: Security Architecture for the Internet Protocol, RFC 4301, IETF (2005).

[3] Kent, S.: IP Encapsulating Security Payload (ESP), RFC 4303, IETF (2005).

[4] 佐々木良一，手塚　悟，石井夏生利，稲葉宏幸，上原哲太郎，越前　功，岡崎美蘭，岡田仁志，岡本栄司，小松尚久，白勢政明，瀬戸洋一，高倉弘喜，土井　洋，村上康二郎：情報セキュリティの基礎，共立出版 (2011).

[5] 馬場達也：マスタリング IPsec，オライリー・ジャパン (2001).

[6] 上醉尾一真，鈴木秀和，内藤克浩，渡邊　晃：IPv4/IPv6 混在環境で移動透過性を実現する NTMobile の実装と評価，情報処理学会論文誌，Vol. 54, No. 10, pp. 2288–2299 (2013).

[7] 納堂博史，八里栄輔，鈴木秀和，内藤克浩，渡邊　晃：実用化に向けた NTMobile フレームワークの実装と評価，第 82 回 MBL・第 53 回 UBI 合同研究発表会，No. 46, pp. 1–8 (2017).

[8] : FNV Hash. `http://www.isthe.com/chongo/tech/comp/fnv/index.html`, 2018/01/23 閲覧.

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

[10] Knuth, D. E.: *THE ART OF COMPUTER PROGRAMMING SECOND EDITION: Volume 3 / SORTING AND SEARCHING*, Addison-Wesley Professional (1998).

研究業績

研究会・大会等（査読なし）

（1）鴨下 友馬，鈴木 秀和，内藤 克浩，渡邊 晃 : エンドツーエンド通信を実現する NTMobile のセキュリティ対策，平成 29 年度電気・電子・情報関係学会東海支部連合大会論文集，No. C3-4, Sep. 2017.

付 録A　ハッシュ関数の比較

当初，簡易認証に用いるハッシュ関数としては，除算法，乗算法，FNV-1(32bit) 以下を候補として挙げていた．以下，これらについて比較する．

除算法の計算式を式 (A.1) に示す [10]．ここで，$r \in \mathbb{N}$ であり，衝突の発生は r に依存する．また，$a \div b$ の剰余を求める演算を $\mathrm{mod}\,(a,b)$ としている．

$$h_d\,(m) = \mathrm{mod}\,(m,r) \tag{A.1}$$

これを基に，入力が $M = \{m_1, m_2, \cdots, m_n\}$ の場合の計算式を式 (A.2) のように定めた．ここで，B は文字列の基数（半角英数字の文字列の場合は $B = 256$）である．

$$h_d\,(M) = h_d\,[B\,h_d\,\{\cdots B\,h_d\,(B\,h_d\,(m_1) + m_2)\cdots\} + m_n] \tag{A.2}$$

乗算法の計算式を式 (A.3) に示す [10]．ここで，$0 < A < 1, q \in \mathbb{N}$ である．また，ある数 x の小数部分を取り出す演算を $\mathrm{dec}(x)$ と表すことにする．

$$h_m\,(m) = \lfloor q\,(mA - \lfloor mA \rfloor) \rfloor = \lfloor q\,\mathrm{dec}(mA) \rfloor \tag{A.3}$$

これを基に，入力が $M = \{m_1, m_2, \cdots, m_n\}$ の場合の計算式を式 (A.4) のように定めた．ここで，B は文字列の基数（半角英数字の文字列の場合は $B = 256$）である．

$$h_m\,(M) = h_m\,[B\,h_m\,\{\cdots B\,h_m\,(B\,h_m\,(m_1) + m_2)\cdots\} + m_n] \tag{A.4}$$

FNV-1(32bit) については 3.2 節を参照されたい．

以上のハッシュ関数の処理時間を比較する．入力 M を 160bit とし，除算法，乗算法，FNV-1(32bit) をそれぞれ 100,000 回実行した際の平均処理時間を表 5 に示す．ただし，実験環境は表 1 と同様とし，除算法のパラメータは $r = 701, B = 256$，乗算法のパラメータは $q = 256, A = (\sqrt{5}-1)/2, B = 256$ とした．

表 5　入力 160bit の場合の平均処理時間

アルゴリズム	平均処理時間 [μs]
除算法	0.544
乗算法	1.247
FNV-1(32bit)	0.410

この結果，最も演算時間の短い FNV-1(32bit) を採用した．

付 録B　ESP headerのフォーマット

図 8 に，ESP header を示す．

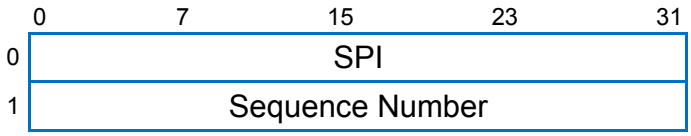

図 8　ESP ヘッダ

各フィールドの内容を以下に示す．

- SPI (32bit)：コネクション識別子
- Sequence Number (32bit)：シーケンス番号（拡張シーケンス番号の場合は下位 32bit）

現状は簡易ハッシュ値を含める余裕がない．したがって，提案方式を適用する際にはヘッダを拡張する必要がある．図 9 に，ヘッダの拡張の一例を示す．

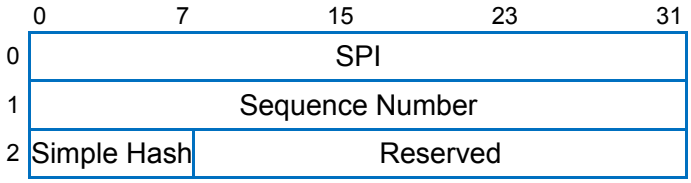

図 9　ESP ヘッダ（変更案）

追加したフィールドの内容を以下に示す．

- Simple Hash (8bit)：簡易ハッシュ値
- Reserved (24bit)：予約フィールド（パディング）

このとき，パケットフォーマットが変わるため，現在の ESP（バージョン 3）との互換性がなくなるという課題がある．したがって，仕様を変更しないと簡易認証を適用できない．

付 録C　NTM headerのフォーマット

図 10 に，2018 年 1 月現在の NTM header を示す．

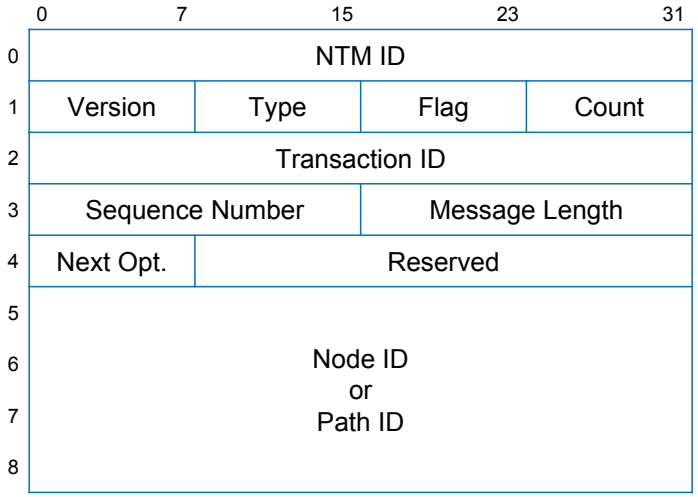

図 10　NTM ヘッダ（現行版）

各フィールドの内容を以下に示す．

- NTM ID (32bit)：NTMobile の ID（現段階では未使用）
- Version (8bit)：NTMobile のバージョン（現段階では Ver.3）
- Type (8bit)：メッセージタイプを表す識別番号
- Flag (8bit)：異常状態などを表すフラグ
- Count (8bit)：メッセージの連続数
- Transaction ID (32bit)：一連のシーケンスを表す ID
- Sequence Number (16bit)：シーケンス番号
- Message Length (16bit)：メッセージ長（対象範囲は図 3 の UDP data 部分）
- Next Opt. (8bit)：連結される拡張ヘッダの番号（拡張ヘッダを使用しない場合は 0）
- Reserved (24bit)：予約フィールド（パディング）
- Node ID or Path ID (128bit)：通信識別子

現在は，シーケンス番号フィールド（Sequence Number）が 16bit となっているが，IPsec と同等のセキュリティを確保するため，このフィールドを 32bit に拡張する必要がある．さらに，提案方式を用いるためのフィールドを付加することを考慮すると，図 11 のように NTM header を変更する必要がある．今回の実験では，この NTM header を採用している．

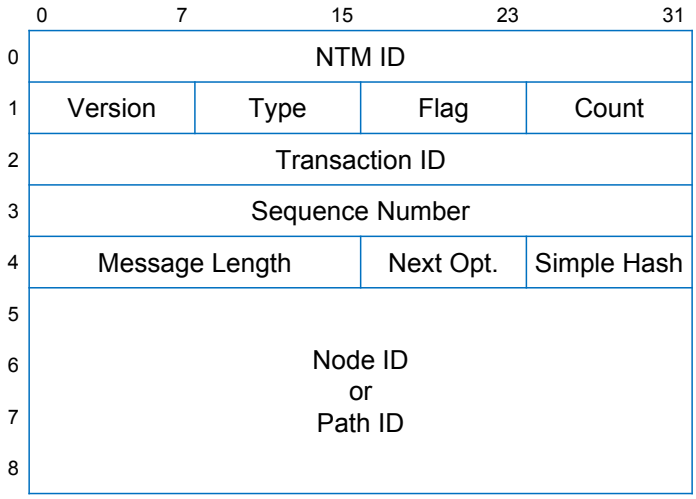

図 11　NTM ヘッダ（変更案）

変更を加えたフィールドの内容を以下に示す.

- Sequence Number (32bit)：シーケンス番号
- Simple Hash (8bit)：簡易ハッシュ値

この場合は予約フィールド（Reserved）が消滅するため，今後，新機能を追加する際には再改修を行う必要がある.

付 録D　テストプログラムの仕様と使用方法

テストプログラムは，以下の 6 つのファイルから成る．

- ntm.h・ntm.c：現行の NTMobile に実装されている関数などを記述したファイル
- check_function.h・check_function.c：新規実装する関数などを記述したファイル
- PacketCheckTest.c：パケット生成からパケット検証処理までを記述したファイル
- PacketCheckTest.sh：コンパイルを効率化するためのシェルスクリプト

簡易認証，リプレイ攻撃チェック，リプレイ防御ウィンドウ更新処理といった，現行の NTMobile に実装されていない部分は，check_function.h および check_function.c に記述した．表 6 から表 8 にかけて，新規実装した関数・定数・マクロの一覧を示す．

表 6　新規実装した関数の一覧

型	関数名	概要
int32_t	ntm_simple_auth	簡易認証を行う関数
uint8_t	ntm_cmpt_fnv1	簡易ハッシュ値を計算する関数
int32_t	ntm_replay_check	リプレイ攻撃チェックを行う関数
void	ntm_update_replay_window	リプレイ防御ウィンドウ更新処理を行う関数
char*	dec2bin32	10 進数を 32bit の 2 進数に変換する関数（デバッグ用）

表 7　新規実装した定数の一覧

型	定数名	値	概要
uint32_t	ntm_fnv_offset_basis	2166136261U	式 (4.1) における $Offset$
uint32_t	ntm_fnv_prime	16777619U	式 (4.1) における $Prime$

表 8　新規実装したマクロの一覧

マクロ名	置換先	概要
NTM_SIMPLE_HASH_DATA_LEN	20	ハッシュ関数の入力データのサイズ [Byte]
REPLAY_WINDOW_SIZE	32	リプレイ防御ウィンドウのサイズ
win_size_t	uint32_t	リプレイ防御ウィンドウのサイズを表す型

テストプログラムを使用するにあたり，ntm.h に定義した各種マクロの有効／無効を切り替えて動作モードを設定する必要がある．表 9 に，各種マクロの一覧を示す．ntm.h の ℓ 行目のマクロをコメントアウトすれば，そのマクロは無効化される．

表 9　動作モード設定用マクロの一覧

ℓ	マクロ名	概要
8	RUN_TIMES	パケット検証処理の実行回数を指定する．
14	LOG	有効時，処理開始・終了などのログを出力する．
15	DEBUG	有効かつ LOG 有効時，計算結果などを含めた全てのログを出力する．
19	WRITE_TO_FILE	有効時，処理時間の計測結果をファイル出力する．
22	SIMPLE_AUTH	有効時，簡易認証の処理時間を計測して終了する．
23	REPLAY_CHECK	有効時，リプレイ攻撃チェックの処理時間を計測して終了する．
24	MAC_AUTH	有効時，MAC 認証の処理時間を計測して終了する．
27	OLD_MODE	有効かつ SIMPLE_AUTH 無効時，簡易認証を行わない．

なお，SIMPLE_AUTH，REPLAY_CHECK，MAC_AUTH が複数有効な場合は，この順に優先される．SIMPLE_AUTH, REPLAY_CHECK, MAC_AUTH がいずれも無効の場合は，リプレイ防御ウィンドウ更新処理の処理時間を計測する．

動作モードを設定後，6 つのファイルを含むディレクトリで

```
sh ./PacketCheckTest.sh
```

と入力するとコンパイルでき，

```
./PacketCheckTest
```

で実行することができる．