Vol.2015-SLDM-170 No.19
Vol.2015-EMB-36 No.19
2015/3/6

組込み向けセキュアモニタとTUNデバイスを用いたセキュアなネットワーク通信機構

金井 遵[1]　橋本 幹生[1]　磯崎 宏[1,2]

概要：本稿ではIoT機器向けの軽量なセキュア仮想化モニタLiSTEE(TM)とLinux向けのTUN/TAPデバイスを用いて，セキュアなVPN（Virtual Private Network）システムを構築する方式を提案する．従来のLinux向けVPNソフトウェアでは，マルウェア等によりLinuxへ侵入され，不正操作によりVPN通信が迂回される可能性がある．これにより意図しない通信先への情報のアップロードによる情報漏洩が起きる危険があった．本方式ではプロトコルスタックとネットワークインタフェースのデバイスドライバからなるネットワーク機能をLiSTEEモニタ上に実装し，LinuxのVPNソフトウェアは本ネットワーク機能を経由して通信するようにした．これによりLinuxへの不正侵入時にも確実に通信のフィルタリングを行うことができ，IoT機器からの情報漏洩を防げる．またVPNパケットの暗号化や認証処理をLiSTEE側で行っており，通信鍵の改竄・漏洩からも保護できる．さらに本方式ではLinuxのネットワーク通信の捕捉をTUN/TAPデバイス等の広く利用されるソフトウェアにより行うなど，Linuxアプリへの透過性を維持しつつも特権モードで動作する独自ソフトウェアの導入が最小限に抑えられ，信頼性が高い構成となる．本機能の実装と評価を行った結果，多くのIoT機器において必要な通信性能が得られることが解った．

1. はじめに

近年，ネットワークに接続される組込み機器（IoT機器）の増加や高機能化に伴い，組込み機器においてもセキュリティが重要な課題となっている．しかし，IoT機器のOSとしても広く利用されるようになったLinuxはOSの規模が非常に大きく，攻撃の糸口となる脆弱性が混入しやすい．中でもLinuxの管理者権限が奪取されるとLinux元来のファイルやプロセスへのアクセス制御が効かなくなるため情報の不正な取得を目的とした攻撃の糸口となりうるが，実際に管理者権限への権限昇格が可能なセキュリティホールが毎月のように発見されている[1]．これを背景として，我々はARM社Cortex-Aプロセッサ搭載のIoT機器向けにTrustZone[3]と呼ばれるハードウェア機能を用いて，Linuxと独立した環境でセキュアにソフトウェアを実行することができる仮想化環境“セキュアモニタLiSTEE(TM)”を開発している[2]．

一方，インターネットを跨がってプライベートネットワークを拡張する技術としてVPN（Virtual Private Network）技術が広く用いられている．IoT機器においても特定のネットワークからIoT機器のリモートメンテナンスを行う場合や，IoT機器が収集したセンサデータをセキュアにアップロードする場合にVPNが有効である．VPN通信を実現する機器には，ルータのようにハードウェア自体がVPNサーバやクライアントとして振る舞うものと，PC向けVPNクライアントのようにVPNソフトウェアを導入し，そのVPNソフトウェアがネットワーク通信をVPN通信に変換するものがある．ハードウェアVPN機器は主に拠点間のVPN接続などで利用され，クライアント機器を修正せずにVPNを導入できるメリットがある．一方，ソフトウェアVPNはモバイルPCなどで利用され，専用機器のコストが不要である．モバイルPCなど直接インターネット接続されるクライアント機器の場合，ハードウェアVPN機器を設置したとしても，PCからハードウェアVPN機器までの通信経路は保護されないが，直接VPNクライアント機能をPC内にソフトウェアとして導入すれば，通信路上で平文が流れない利点もある．この場合，VPNサーバをハードウェアVPNで，VPNクライアントをソフトウェアVPNで構築する例も多い．IoT機器は宅外設置される場合も多く，ソフトウェアVPNの導入が好ましい．

しかしLinux上で動作する従来のソフトウェアVPNの主な問題点として，VPN通信を強制することが困難な点が挙げられる．VPNシステムの構成例として，プライベート

[1] (株) 東芝 研究開発センター
Corporate R&D Center, Toshiba Corporation, Japan
[2] 慶應義塾大学大学院政策・メディア研究科
Graduate School of Media and Governance, Keio University

ネットワークへのゲートウェイを VPN サーバ，インターネット上にある IoT 機器やモバイル PC を VPN クライアントとする構成を考える．このとき，不正な VPN クライアントからプライベートネットワークへの接続は VPN サーバ側での認証で弾くことができる．一方で VPN クライアント側では，VPN サーバに限らず任意のサーバへの接続が可能である．すなわち，不正な VPN クライアントへの VPN 内の情報漏洩は防げるが，正規の VPN クライアントから不正なサーバへの情報漏洩は防げない．重要情報を扱う IoT 機器において，これは大きな問題である．

そこで我々は，Linux 用の VPN クライアントソフトウェアに手を加え，セキュアモニタ LiSTEE と連携して通信を行うことで VPN 通信が迂回困難な仕組みを設計，実装した．また本方式は Linux 側の通信先が特定ネットワークに限定されるため用途は限定されるものの，単に複数 OS でネットワークデバイスを共有するためのネットワークデバイス仮想化の仕組みとしても利用できる．本稿では本機能の設計，実装，評価結果について述べる．

2. 既存技術の問題点とセキュアな VPN クライアントソフトウェアの要件

本節では既存 VPN クライアントソフトウェアとネットワークデバイス仮想化の問題を説明し，問題を解決するセキュアな VPN クライアントソフトウェアの要件を述べる．

2.1 一般的な VPN システムの構成

既存 VPN クライアントソフトウェアの問題点を議論するために，まず一般的な Linux 向け VPN クライアントソフトウェアを用いた VPN システムの構成を図 1 に示す．

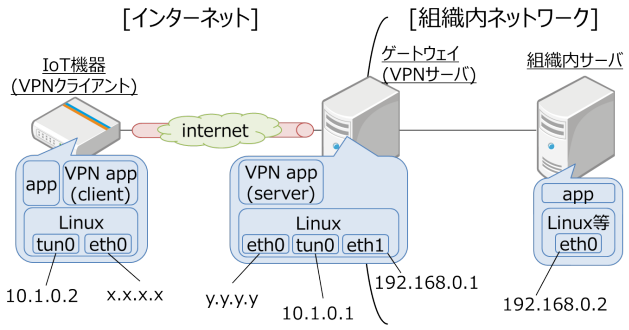

図 1 一般的な VPN システムの構成

一般的な VPN システムでは，VPN クライアントと VPN サーバ（ゲートウェイ）間で仮想的なトンネルが張られ，組織内ネットワークなどのプライベートネットワーク内のサーバへアクセスが可能になる．

このとき，多くの Linux 向け VPN クライアント [7][8] では Universal TUN/TAP[6] デバイスと呼ばれる仮想的なネットワークデバイス（図中では VPN クライアントの tun0）が機器内に作られる．TUN/TAP デバイスを用いるとユーザモードで動作する VPN ソフトウェアが，Linux 上で動作するアプリケーションからのネットワーク通信要求を捕捉することができる．ネットワーク通信を行うプログラムが OS に対してネットワーク通信要求を出したとき，TUN/TAP デバイスはユーザモードで動作する VPN ソフトウェアに対して通信要求を転送する．この要求内容を元に VPN ソフトウェアは VPN パケットへのカプセル化を行い，今度は物理ネットワークデバイス（図中では VPN クライアントの eth0）を通じて VPN サーバへの通信要求を出すことで，レイヤ 3（TUN）もしくはレイヤ 2（TAP）の VPN 通信を実現する仕組みとなっている．

2.2 VPN クライアントソフトウェアにおける問題点

VPN のクライアント端末に導入する既存の VPN ソフトウェアの主な問題点として，(1)VPN 通信の迂回が可能，(2) 通信暗号路や認証に用いる鍵の改竄や漏洩危険性が高い，以上の 2 点が挙げられる．

(1) については，物理ネットワークデバイスのデバイスドライバを Linux や Windows などの VPN ソフトを導入する OS が持つため，VPN ソフトを経由せずに，直接物理ネットワークデバイスを指定して通信できることによる問題である．この対策として，Linux ではルーティング設定により VPN 接続優先の通信や，iptables によるパケットのフィルタリングが可能である．しかし，管理者権限の不正奪取によりこれらの設定も変更される可能性がある．さらに OS の強制アクセス制御機能 [4] を用いた設定変更の制限や，マルウェア対策ソフトやホスト型侵入検知ソフトを用いた対策を行うこともできるが，OS 自体の脆弱性を突いて機器に侵入された場合には，これらの対策自体が無効化される恐れがある．すなわち OS の脆弱性の存在を想定すると，VPN を経由したネットワーク通信を強制することは困難で，情報漏洩を招く可能性がある．

(2) については，ユーザ入力のパスワードから通信路暗号化用の共通鍵を生成するような場合には，キーロガー等によりパスワードが漏れると通信路暗号化用の共通鍵が解ってしまう．また，鍵生成は乱数に依存することも多く，DH（Diffie-Hellman）法による鍵交換を例に取ると，DH 法では鍵の生成に使う乱数が漏洩すると通信路の暗号化鍵が容易に求められてしまう．Linux で鍵生成を行う場合には，Linux に侵入したのちに乱数生成器が返す値を固定する攻撃も考え得る．一度乱数生成器の返す値を固定してしまえば，あとは通信路上に流れる情報を盗聴するだけで暗号化鍵が解ってしまい，もう一度 Linux に潜入せずとも通信路上に流れる情報の復号が可能になってしまう．

2.3 ネットワークデバイス仮想化における問題点

一方，LiSTEE のような仮想化環境においては，VMM

上で動作する複数のゲスト OS でのネットワークデバイス共有や，ゲスト OS の通信のフィルタリングを行うために，ゲスト OS 内に専用ネットワークドライバを導入したり，仮想化環境内にネットワークデバイスへのアクセスをトラップする機構を導入したりすることも多い．しかしながら，これらのソフトウェアは大規模であることが多く [5]，脆弱性が混入するリスクが高い．また特定用途に向けて独自開発されたものも多く [9]，汎用ソフトウェアを利用した構成のほうが信頼性で勝る．セキュリティ上，カーネル内に導入するソフトウェアは小規模で枯れているソフトウェアのみで構成されることが望ましい．

2.4 セキュアな VPN クライアントソフトウェアの要件

以上の問題点を解決する，セキュアな VPN システムを構築する上での要件として以下の四点を挙げる．

(1) VPN クライアントソフトウェアが動作する Linux への侵入時にも VPN 通信の迂回が困難であること
(2) VPN クライアントソフトウェアが動作する Linux への侵入時にも通信路暗号化や認証に用いる鍵の漏洩を防止でき，改竄が困難であること
(3) 上記 2 要件を実現する際に，特権モード（TrustZone の場合には後述のセキュアワールドで動作するユーザモードプログラムも含む）で動作する独自ソフトウェアの導入が最小限ですむこと
(4) ネットワーク通信を行う一般の Linux アプリに対する変更や再コンパイルが不要であること（透過性の維持）

以降では，これらの要件を満たすセキュアな VPN クライアントソフトウェア（以下，セキュア VPN クライアントと呼ぶ）の仕組みについて説明する．

3. LiSTEE と TUN/TAP デバイスを用いたセキュア VPN クライアント

本節ではセキュア VPN クライアントの仕組みとソフトウェアの設計について説明する．

3.1 システム構成

我々は ARM Cortex-A コア搭載の IoT 機器において，Linux と Linux と独立した環境でセキュアにソフトウェアを並行実行することができる仮想化環境“LiSTEE モニタ”を開発している．LiSTEE モニタでは TrustZone と呼ばれる仮想化支援機能を活用して OS の並行実行を実現している．TrustZone ではプログラムを実行するワールドがセキュアワールドとノーマルワールドに分離され，一般的にノーマルワールドでは Linux などの汎用 OS を，セキュアワールドではセキュリティを担保すべき処理のみを実行する．両ワールドはソフトウェア例外命令（Secure Monitor Call : SMC 命令）で遷移でき，連係動作が可能である．なお，セキュアワールドのメモリ領域は TZASC（TrustZone Address Space Controller）と呼ばれる機構で保護され，ノーマルワールドからセキュアワールドのプログラムが扱うメモリ領域にアクセスすることはできない．また，ペリフェラルについても TZPC（TrustZone Protection Controller）と呼ばれる機構でデバイス毎に保護することができ，ノーマルワールドからの操作を禁止できる．

これらの仕組みを用いてセキュアワールド側のみで物理ネットワークデバイスを扱うようにすることで，ノーマルワールド側からは直接物理ネットワークデバイスにアクセスすることができなくなる．ノーマルワールドからもネットワーク通信を行いたい場合，セキュアワールド側にノーマルワールドからの通信要求を受け付け，代理でネットワークアクセスを行う機能を設けることとなる．つまり，ノーマルワールド側からは必ずセキュアワールド側を介して通信を行う必要があるため，セキュアワールド側で確実に通信のフィルタリングを行うことが可能となる．VPN 通信についてもセキュアワールド側を経由して行う仕組みとすれば，例え Linux に侵入されたとしても VPN 通信を迂回して，意図しないサーバに情報を送信することは困難となる．この場合の VPN システムの構成図を図 2 に示す．

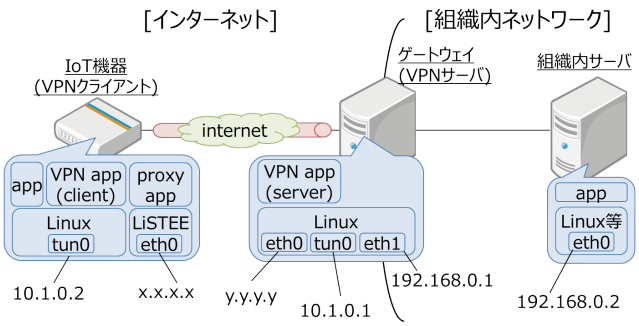

図 2 セキュアな VPN システムの構成

単に物理ネットワークデバイスアクセスをセキュアワールドで行うだけでなく，鍵管理や鍵を扱う認証処理，VPN パケット暗号化をセキュアワールドで行うことにより，Linux 侵入時にも鍵の漏洩や改竄を防止でき，2.4 節の要件 (1) に加えて (2) を充足するシステムを実現できる．

3.2 ソフトウェア構成と通信の流れ

3.1 節で述べたシステム構成を実現するために今回設計したセキュア VPN ソフト（VPN クライアントソフトウェア）のソフトウェア構成を図 3 に示す．今回導入するソフトウェアは大別して，Linux 側に導入するセキュア VPN モジュールとワールド間通信用のドライバ，LiSTEE のセキュアワールドで動作するセキュア VPN モジュールからなる [*1]．各機能について以下に述べる．

[*1] これ以外に TUN/TAP デバイスドライバや LiSTEE モニタも導入するが，これらは既存ソフトウェアをそのまま利用するため，本稿では説明を省略する．

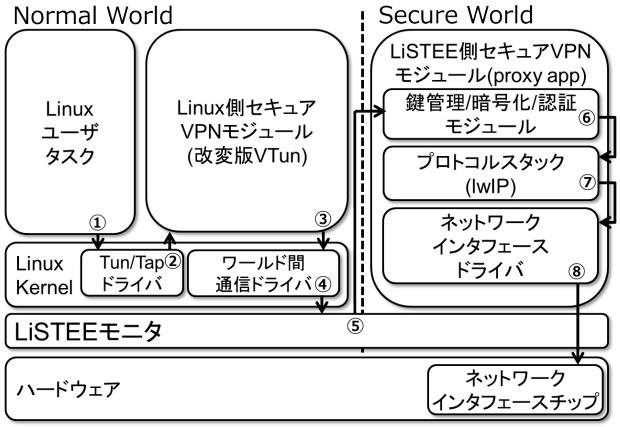

図 3 ソフトウェアスタックと通信の流れ

(1) Linux 側セキュア VPN モジュール

TUN/TAP デバイスからパケット送受信要求を受け取り，データ圧縮や VPN ヘッダ付与などの後，セキュアワールドで動作する LiSTEE 側のセキュア VPN モジュールに通信要求を渡すための機能を持つ．本機能はオープンソースの VPN ソフトウェア VTun[8] を流用して実装した．オリジナルの VTun は UDP もしくは TCP を用いた独自方式で VPN サーバと通信を行うが，今回は UDP のみに対応することとした．VTun への変更点として，オリジナルの VTun は物理ネットワークへの通信に socket で生成したディスクリプタに対する readv/writev システムコールを利用しているが，これをワールド間通信用ドライバを使った通信要求内容の共有メモリへの read/write とワールド間遷移命令に置き換えた．また，VPN サーバとのハンドシェイク処理は VPN パケットの送受信に UDP を使うか TCP を使うかの設定によらず TCP で行われるが，この部分を UDP により行うように設計を変更し，UDP 通信のみで通信が完結するようにした．一方，セキュリティ強度に関わらないデータ圧縮やヘッダの付与などの処理については極力 Linux 側に残してセキュアワールドの処理を最小限に留め，セキュアワールドへの脆弱性混入リスクを小さくしている．また，VPN サーバとの認証処理も LiSTEE 側の VPN モジュールを経由して行うように変更した．

(2) Linux 用ワールド間通信用ドライバ

Linux 用デバイスドライバであり，ワールド間で共有される共有メモリに対する読み書き機能と，SMC 命令によるワールド間遷移機能を提供する．本ドライバはキャラクタデバイスとして実装され，Linux 側のセキュア VPN モジュールから呼び出される．

(3) LiSTEE 側セキュア VPN モジュール

共有メモリに置かれた Linux 側セキュア VPN モジュールからの通信要求を解釈し，パケットの暗号化および VPN パケットへのカプセル化（UDP/IP パケットの生成）を行い，物理ネットワークに送信する機能を持つ．また，パケットの受信時には VPN パケットのカプセル化解除と復号と共有メモリへの配置を行う．このため，ネットワーク通信を行うためのプロトコルスタックとイーサネットのデバイスドライバについても実装した．プロトコルスタックに関してはオープンソースプロトコルスタックの lwIP を流用している．lwIP は TCP/IP など多くのプロトコルに対応しているが，今回はセキュリティの観点から UDP や ARP など必要最低限の機能のみを導入している．また，VPN サーバへの接続時の認証（チャレンジアンドレスポンス）でのレスポンスの生成や，経路暗号化鍵の生成にも対応している．これらの鍵はセキュアワールドでのみ扱われ，Linux 側から読むことはできない．

以上のソフトウェアを用いた通信の流れ（送信の場合）は次のようになる．Linux 上のユーザタスクが Linux カーネルに対して send システムコール等により送信要求を出すと（図 3 の 1），TUN/TAP ドライバは Linux 側セキュア VPN モジュールに要求を転送する（同 2）．続いて，Linux 側セキュア VPN モジュールは VPN 関連のヘッダを付加して LiSTEE OS 間通信ドライバを呼び出して共有メモリにパケットを書き込み，さらにワールド間遷移要求を出す（同 3）．ワールド間通信ドライバはワールド間遷移命令を発行し（同 4），LiSTEE モニタはコンテキスト待避・復帰によりワールドを切り替える（同 5）．続いて，LiSTEE 側セキュア VPN モジュール内の暗号化モジュールは，自身で管理する暗号化鍵を用いてパケットの暗号化を行い，UDP パケットとして VPN サーバへの送信を要求する（同 6）．プロトコルスタックは UDP および IP ヘッダの付与ののち，イーサネットフレームを生成してネットワークインタフェースドライバを呼び出し（同 7），ネットワークインタフェースドライバは物理ネットワークインタフェースを操作してイーサネットフレームを送信する（同 8）．

6 の段階での通信先のフィルタリングにより VPN 通信の迂回は困難である．また，通信路暗号化鍵の生成や認証に用いるレスポンス生成もセキュア側で行い，鍵の扱いはセキュアワールドに閉じるので Linux 侵入時にも漏洩や改竄の心配がない．また，Linux アプリからは通常の socket 関連のシステムコールを利用してネットワークアクセスができ，2.4 節に示した要件 (4) の透過性も維持される．

4. 実装

セキュア VPN クライアントの実装対象として，ARM 社製リファレンスボード Versatile Express（Coretile A9x4 搭載）を今回利用した．CPU コアとしては 400MHz 動作の Cortex-A9 を搭載し，メモリ容量は DRAM が 1GB，SRAM が 40MB となっている．また，ノーマルワールドで動作させる Linux としては Linux 3.6 を利用している．

なお，メモリ空間としては Linux に 256MB(DRAM)，LiS-

TEE モニタに 1MB(SRAM)，LiSTEE 側セキュア VPN モジュールを含むセキュアワールドのプログラム動作用に残りの 39MB の SRAM を割り当てている．ワールド間の共有メモリは DRAM の末尾に置かれる．SRAM に関しては，TZASC によってノンセキュアワールドからの読み書きを禁止している．

Versatile Express には，イーサネットデバイスとして SMSC LAN 9118 チップが搭載されており，デバイスドライバとして u-boot から LAN 911x ドライバを移植した．今回導入・開発した各ソフトウェアのコード量と，元々のソースコードがあるソフトウェアについては改変したコード量，さらにソフトウェアの動作モードを表 1 に示す [*2]．

セキュアワールドで動作するプログラムは LiSTEE モニタを除き，プロトコルスタックやイーサネットドライバを含めてユーザモードで動作する．このため，イーサネットデバイスのレジスタやバッファはユーザモードからもアクセス可能な領域（仮想アドレス空間）にマップしている．

5. 考察・評価

本セキュア VPN クライアントのセキュリティ強度に関する考察および，性能評価としてレイテンシとスループットの計測を行った結果について述べる．

5.1 セキュリティ強度に関する考察

本手法では，Linux への侵入時にもアクセス先の変更などによる VPN 通信の迂回が困難である．また，通信路暗号化に用いる鍵の生成・管理や認証時のレスポンス生成，暗号化処理も LiSTEE 側で行うため，Linux への侵入時にも各種鍵の保護が可能である．これらの特徴により一般的なソフトウェア VPN を利用する場合に比べて，端末からの情報漏洩の危険性は低いと言える．

一方，セキュアワールド側に VPN 機能を導入することにより，セキュアワールド側に導入するソフトウェアの量が増え，セキュアワールドへの脆弱性の混入リスクが増える欠点がある．特に，プロトコルスタックやネットワークインタフェースドライバをセキュア側に導入することにより攻撃界面が増えることとなる．しかし本構成では，セキュアワールドの特権モードで動作するコードの総行数は 5000 行以下に抑え，脆弱性混入リスクを小さくしている．

また，セキュアワールドのユーザモードで動作するプログラムもプロトコルスタックとしてサポートする機能を UDP/IP 通信を行うために必要な最小機能に抑えたほか，VPN 本体の機能も 2.4 節で述べた (1)，(2) の要件を実現するために必要最低限の機能のみ導入し，多くの機能は Linux 側に残している．これらの総行数は 2 万行程度，そのうち独自部分は 2000 行程度であり，残りは広く利用され「枯れている」コードである．加えて，Linux 側に導入する独自ドライバも 200 行程度と小さい．

以上より，本構成は 2.4 節の要件 (3) を満たし，VPN 機能を全てセキュア OS 側で持つ構成に比べて脆弱性混入リスクを低減するのに適した構成であることは明らかである．

5.2 性能評価

セキュア VPN クライアントの性能計測として，レイテンシとスループットの測定を行った．レイテンシについては，100Mbps 接続の LAN 環境において ping プログラムを用いて VPN クライアントとサーバ間のラウンドトリップタイム (RTT) を計測した．また，スループットについては同じく LAN 環境の http サーバから wget コマンドを用いて zip 圧縮された 10MB のファイルをダウンロードする時間を計測した．VPN 機能を利用せずに直接サーバと通信した場合，オリジナルの VTun による通信，本 VPN 機能を使った通信の三種類で計測した結果を表 2 に示す．

表 2 レイテンシおよびスループット評価の結果

通信方法	RTT[ms]	スループット [Mbps]
直接通信	1.13	16.5
オリジナル VTun	3.13	9.3
セキュア VPN クライアント	4.67	3.0

結果より，直接接続に比べオリジナルの VTun では RTT が 3 倍弱に増加している．さらにセキュアソフト VPN では暗号化処理や OS 遷移処理がオーバヘッドとして加わり，オリジナルの VTun に対して RTT が 1.5 倍程度になっている．ただし，本機能の利用が想定されるインターネット接続では，日本国内から国内への RTT は数十 ms 程度，米国へは 100ms 程度，欧州へは 200ms 程度の RTT があると言われている．本評価は LAN 環境での測定結果であり，インターネット環境で本機能を利用する場合には，オーバヘッドは相対的に小さいと言える．

スループットについては，オリジナルの VTun において直接接続の約半分のスループットに低下している．さらに，セキュア VPN クライアントによって 1/3〜1/4 程度の性能（直接接続の 1/5〜1/6 程度の性能）となった．しかし，3.0Mbps 程度の通信速度は確保できるため，センサネットワークからのセンサ情報のアップロード，ファームウェアアップデート，機器診断情報のやりとりなどといった，小さなデータのやり取りや大きなデータであっても送受信頻度が低い用途には利用可能と考えられる．

また，LiSTEE 内でのプロトコルスタックと暗号化モジュール間，プロトコルスタックとネットワークインタフェースドライバ間におけるメモリコピー回数の削減などの余地があり，これらの最適化により性能向上が見込める．そのほかにも，より根本的にはワールド間遷移回数を減らすことが有効と考えられる．現状 1 パケットの送信毎

[*2] プロトコルスタックについてはコンパイル時にリンクされない TCP/IP 等のモジュールを除く行数を概算した

表 1　ソースコード行数と動作モード

ソフトウェア	総行数	改変/追加行数 (内数)	動作ワールド	動作モード
Linux 側セキュア VPN モジュール	12066	418	ノーマル	ユーザモード
Linux 側ワールド間通信ドライバ	212	同左	ノーマル	特権モード
LiSTEE モニタ (初期化コード等含む)	4814	同左	セキュア	特権モード
LiSTEE 側セキュア VPN モジュール（鍵管理/暗号化/認証機能）	1813	1200	セキュア	ユーザモード
LiSTEE 側セキュア VPN モジュール（プロトコルスタック）	16954	228	セキュア	ユーザモード
LiSTEE 側セキュア VPN モジュール（イーサネットドライバ）	948	167	セキュア	ユーザモード

にワールド間の遷移を行っているが，例えば複数の送信パケットがある際にはまとめて処理するといった工夫によって性能を向上させることができる可能性がある．

6. 関連研究

VPN 接続をセキュアに行う方法として，VPN 接続時に 2 要素認証を行う方法が広く用いられている．また，2 要素認証におけるワンタイムパスワードを専用ハードウェアトークンではなく，一つの端末内で仮想化機構や専用チップを用いて生成するようにした研究や製品も存在する [11]．これらは不正な端末やユーザの VPN への接続を防止することができ，不正な VPN クライアントへの情報漏洩は防止できる．しかし，VPN クライアント内での VPN 接続の迂回までは防止できない．よって，VPN クライアント機器へのマルウェア侵入まで考慮すると，VPN クライアントからの情報漏洩の防止は困難である．提案方式では，不正な VPN クライアントへの情報漏洩だけでなく，VPN 接続の迂回や暗号化鍵の改竄などによる VPN クライアントからのネットワーク経由での情報漏洩を防止できる．

仮想化技術を用いてネットワーク通信を VPN 通信化する研究として BitVisor[9] の VPN 機能 [10] がある．BitVisor の VPN 機能では，VMM 内にネットワーク通信をトラップする機能を設けて BitVisor 内の VPN モジュールがゲスト OS のネットワーク通信を VPN 通信に変換する．この方式にはゲスト OS 側に対し，VPN 通信が行われていることを隠蔽できるという利点がある一方で，VMM 内に組み込む機能が多くなるため，脆弱性が混入するリスクが増えるという欠点がある．提案方式では，Linux 側から VPN 通信が行われていることは分かってしまうが，セキュアワールド (VMM) に組み込む機能はプロトコルスタックとネットワークドライバの他は小規模なプログラムのみで良く，脆弱性の混入リスクを減らすことができる利点がある．

また，Cooperative Linux（coLinux）[12] では Windows 上で動作する Linux から Windows のネットワークデバイスドライバを介して通信を行うために，TUN/TAP デバイスを用いる．ただしゲスト OS となる Linux ではなく，ホスト OS の Windows に TUN/TAP デバイスを導入する．Xen や VMWare Player などでも同じく，ネットワーク共有のためにホスト OS やドメイン 0 に仮想的なネットワークデバイスを生成する．つまり，Linux 側（ゲスト OS 側）に TUN/TAP デバイスを導入する本方式とはネットワーク仮想化の実現方法が異なる．OS 間でのネットワークデバイスの共有は両者とも可能であるが，coLinux などではゲスト OS 上の VPN 接続のセキュア化は想定されていない．

7. おわりに

本稿では IoT 機器向けの軽量なセキュアモニタ LiSTEE と Linux 向けの TUN/TAP デバイスを用いて，セキュアな VPN システムを構築する方式を提案した．設計・実装および評価を行った結果，多くの IoT 機器において必要となる通信性能が得られることが解った．今後の課題としては性能の最適化などが挙げられる．

参考文献

[1] MITRE: Common Vulnerabilities and Exposures (CVE), http://cve.mitre.org/
[2] 金井ほか: 高速な OS 切替え機構を有する組込み機器向けセキュアモニタ LiSTEE, 情報処理学会 研究報告 2013-OS-126(19), pp.1–8(2013).
[3] ARM: TrustZone, http://www.arm.com/ja/products/processors/technologies/trustzone.php
[4] P. Loscocco et al.: Integrating Flexible Support for Security Policies into the Linux Operating System, Proc of the FREENIX Track: 2001 USENIX Annual Technical Conference (FREENIX'01), pp 29-42 (2001).
[5] Barham, P. et al. : Xen and the Art of Virtualization, 19th ACM Symposium on Operating Systems Principles (SOSP'03), pp. 164–177 (2003).
[6] Universal TUN/TAP device driver, https://www.kernel.org/pub/linux/kernel/people/marcelo/linux-2.4/Documentation/networking/tuntap.txt
[7] SoftEther: SoftEther, http://www.softether.jp/
[8] VTun(Virtual Tunnel), http://vtun.sourceforge.net/
[9] 品川ほか: 準パススルー型仮想マシンモニタ BitVisor の設計と実装, 2008 年並列／分散／協調処理に関する『佐賀』サマー・ワークショップ（SWoPP 佐賀 2008）(2008).
[10] 登：BitVisor における透過的な VPN 処理の設計と実装, セキュア VM ワークショップ (2008).
[11] van Rijswijk-Deij, R.M. et al.: Using Trusted Execution Environments in Two-Factor Authentication: comparing approaches, Proc of the Open Identity Summit 2013 (OID 2013), pp. 9–11(2013).
[12] Dan Aloni: Cooperative Linux, Proc of Linux Symposium 2004, Vol.1, pp.3-31(2004).