アプリケーションから機能拡張された通信ライブラリを利用する方法に関する研究

173426012　清水 一輝
渡邊研究室

1. はじめに

ライブラリは処理速度や移植性，他言語との連携といった観点から，C 言語で実装される．しかし，アプリケーションは C 言語よりも抽象度の高い高級言語にて開発されることが一般的である．そのため，C 言語ライブラリを使用したい場合は，一般的に高級言語に応じたラッパーを経由し，C 言語ライブラリを意識したプログラミングが必要である．しかし，通信ライブラリのようなプログラミング言語の標準 API として備わっている機能を拡張するライブラリの場合は，高級言語の提供する標準 API と同じ使用方法を踏襲できることが望ましい．

本研究では，スマートフォン向けアプリケーションに通信ライブラリを適用する方法をラッパー方式と VPN 方式といった 2 通りの提案と一部実装を行った．ラッパー方式はアプリケーションに組み込むことで，機能拡張された通信を行うための方式である．また，VPN 方式は Android OS/iOS の VPN 機能を利用することで，アプリケーションに機能拡張された通信を提供する方式である．動作検証を行った結果，実装したラッパーと VPN アプリケーションが実現可能であることを確認した．

2. 既存技術

ラッパーを生成する既存技術に SWIG (Simplified Wrapper and Interface Generator) がある．SWIG とは，C 言語や C++にて書かれたプログラムやソフトウェアを高級言語と連携させることで，高級言語から使用可能にするソフトウェア開発用ツールである [1]．SWIG は，SWIG の独自記述ファイルである i ファイルを用いて，C 言語もしくは C++と高級言語を連携させるグルーコードを生成する．その後，グルーコードから連携したい高級言語用のラッパーを生成する．このラッパーを高級言語が使用することで，高級言語から C 言語もしくは C++のプログラムやライブラリを使用できるようになる．

SWIG によって自動生成されたラッパーは，C 言語もしくは C++のプログラムやライブラリの実装に依存するという課題がある．この課題は，画像の特徴点抽出のような高級言語には存在しない新機能の場合であれば問題はない．しかし，暗号通信のように高級言語に存在する既存の送信/受信機能を拡張したような機能であれば，これらの機能の使い方は高級言語と同じ使用方法であることが望ましい．

3. 提案方式

スマートフォン向けアプリケーションに通信ライブラリを適用する方法をラッパー方式と VPN 方式といった 2 通りの提案をする．ラッパー方式はアプリケーションに組み込むことで，VPN 方式は Android OS/iOS の VPN 機能を利用することで，機能拡張された通信を行う方式である．

3.1 ラッパー方式

ラッパー方式では，C 言語や C++ライブラリ (以下，C ライブラリと呼ぶ．) の機能を 2 重にラップすることにより，高級言語と同じ使用方法で使用できるようにする．図 1 にラッパー方式の動作を示す．C ライブラリの機能をラップして高級言語から使用可能するラッパーを呼び出し用ラッパーとし，呼び出し用ラッパーを更にラップして高級言語と同じ使用方法で使用できるようにするラッパーを API 用ラッパーとする．

呼び出し用ラッパーでは，FFI (Foreign Function Interface) の仕組みを用いて C ライブラリが提供する API を呼び出し元の高級言語から使用可能にする．この際，ラッパーでは C ライブラリと呼び出し元の高級言語の関数や引数の型のマッピングを行う．これにより，高級言語から C ライブラリの API を呼び出し，使用することが可能になる．

API 用ラッパーでは，呼び出し用ラッパーにて使用可能になった C ライブラリの API を高級言語の使用方法と同じ使用方法にする．高級言語の通信用クラスを継承し，継承したクラスに C ライブラリの API を実装する．これにより，この通信用クラスの使用方法は継承元のクラスと同じ使用方法，すなわち高級言語の標準ライブラリと同じ使用方法になる．継承したクラスに API を実装する上で高級言語の標準 API の引数では調整することのできないパラメータが存在する場合がある．この場合は，高級言語の仕様に基づいた処理を行うようにパラメータをラッパー内で指定する．

アプリケーションは API 用ラッパーに実装された API を使用すると，これまでと同じ使用方法で，C ライブラリの API を呼び出し，実行することができる．

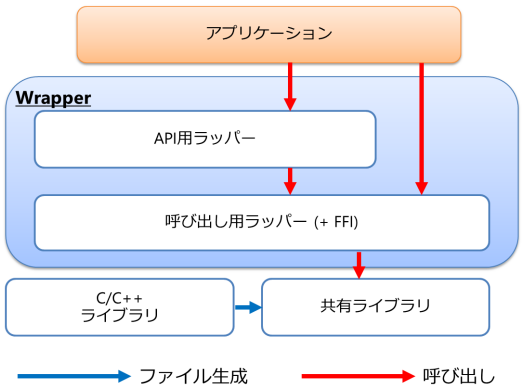

図 1: ラッパー方式の動作

3.2 VPN 方式

図 2 に VPN 方式の動作を示す．VPN 方式では，カーネルをフックし，通信ライブラリに処理をさせることにより実現する．

アプリケーションはパケットを生成し送信すると，VPN アプリケーションにて事前に作成した Virtual I/F (Virtual Interface) がパケットをフックする．パケットをフックした後は VPN アプリケーション内でパケットを解析する．パケットを解析後は，パケットの種類に応じて Android OS や iOS にて適用したい通信ライブラリの処理を行うようにパケットに変更を加える．パケットに変更を加えた後は，

Real I/F (Real Interface) を経由して実ネットワーク上にパケットを送信する．アプリケーションがパケットを受信する際は，上記と逆の動作をする．

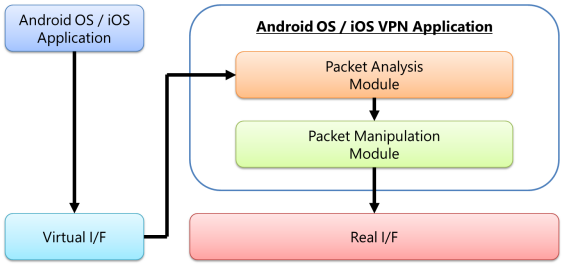

図 2: VPN 方式の動作

4. NTMobile framework library

実装を NTMfw (NTMobile framework library) を対象として行った [2]．NTMfw は暗号化トンネル通信機能をアプリケーションに提供する通信ライブラリである．NTMfw の利用モデルにはフレームワーク組込型と TUN/TAP 利用型が存在する．フレームワーク組込型では，標準ソケット API を用いる代わりに，NTMfw の提供する NTM ソケット API を用いることで暗号化トンネル通信を実現する方式である．TUN/TAP 利用型では，アプリケーションによって生成されたパケットをネットワーク上に送信する前にフックして別アプリケーションに渡すという TUN/TAP インタフェースの仕組みを用いて暗号化トンネル通信を実現する方式である．

提案方式が対象とする利用モデルは，ラッパー方式ではフレームワーク組込型，VPN 方式では TUN/TAP 利用型とする．

5. 実装と評価

5.1 実装と性能評価

提案方式のラッパー方式を実装し，VPN 方式の一部実装を行った．仮想環境においてシステムを構築し，ラッパー方式の処理時間と VPN 方式で利用する TUN の処理時間を測定した．表 1 にラッパー適用時と TUN 適用時の 100 回測定した結果の平均処理時間を示す．

表 1: 平均処理時間

	送信時 [μs]	受信時 [μs]
ラッパー	573.86	959.31
TUN	52.19	136.58

5.2 考察

ラッパー適用時と TUN 適用時に送信時の合計処理時間では約 520[μs]，受信時では約 820[μs] の差が生じていた．これには 2 つの原因がある．

1 つ目は，カプセル化/デカプセル化の処理時間である．TUN 適用時の処理時間にはカーネル空間で行われたカプセル化/デカプセル化の処理時間が含まれていない．これに対して，ラッパーでは lwIP を使用してユーザ空間で行われているカプセル化/デカプセル化の処理時間が含まれている．lwIP を用いたカプセル化/デカプセル化の処理時間は，送信時では約 280[μs]，受信時では約 720[μs] である．

2 つ目は，ラッパーという他言語の経由である．ラッパー適用時には C 言語と Java というプログラミング言語間でのデータの違いの除去を行っている．しかし，TUN 適用時にはパケット自体を書き換えているため，プログラミング言語間の違いを除去する必要がない．プログラミング言語間の違いの除去に要した処理時間は，送信時では約 215[μs]，受信時では約 75[μs] である．

以上の 2 つの原因が，ラッパー適用時と TUN 適用時における処理時間の差を生じさせている．

5.3 比較

提案したラッパー方式と VPN 方式を比較した結果を以下の表 2 に示す．

表 2: ラッパー方式と VPN 方式の比較

	ラッパー方式	VPN 方式
(i)	△	○
(ii)	×	○
(iii)	○	×

(i) アプリケーション開発時における容易性
- ラッパー方式では使用するクラス名やモジュール名を標準クラスやモジュールとは異なるのに対して，VPN 方式では標準クラスやモジュールを使用して問題ないため従来と変更点は一切ない

(ii) 既存アプリケーションに対する拡張性
- クラス名やモジュール名を全て変更する必要があるのに対して，提案方式では一切変更を加えることなく既存アプリケーションに適用可能である

(iii) 他の VPN との同時使用
- ラッパー方式では Android OS/iOS の VPN 機能を使用しないため他の VPN との同時使用が可能であるのに対して，VPN 方式では Android OS/iOS の VPN 機能を使用する必要があるため他の VPN アプリケーションとの同時使用が不可能である

以上より，既に他の VPN アプリケーションを使用している場合にはラッパー方式，他の VPN アプリケーションを使用していない場合は VPN 方式の方が利便性が高いと考えられる．

6. まとめ

アプリケーションから機能拡張された通信ライブラリを利用する方法について提案と実装を行った．

ラッパー方式では，C 言語や C++といった低級言語にて実装された通信ライブラリを高級言語から使用する際に，高級言語と同じ使用方法にて通信ライブラリを使用可能にした．VPN 方式では，スマートフォン上のアプリケーションが生成するパケットを Virtual I/F 経由させることで，アプリケーションに一切変更を加えることなく通信ライブラリを適用する方法を提案した．

Linux 上で Java 用ラッパーと VPN 方式の土台となる TUN/TAP 利用型の動作検証と性能評価を行い提案方式が動作することを確認した．

参考文献

[1] SWIG developers : Simplified Wrapper and Interface Generator，http://www.swig.org/.

[2] K.Naito，et al.，”End-to-end IP mobility platform in application layer for iOS and Android OS”，IEEE CCNC 2014，pp.276-281，2014.