DEIM Forum 2022 K23-5

Android 端末における輻輳予測モデルのパフォーマンス評価

佐藤　里香† 　山口　実靖†† 　神山　剛††† 　小口　正人†

† お茶の水女子大学　〒 112–8610 東京都文京区大塚 2–1–1
†† 工学院大学　〒 163–8677 東京都新宿区西新宿 1–24–2
††† 長崎大学　〒 852–8521 長崎県長崎市文教町 1–14
E-mail: †{rika,oguchi}@ogl.is.ocha.ac.jp, ††sane@cc.kogakuin.ac.jp, †††kami@nagasaki-u.ac.jp

あらまし　近年, 機械学習のモデルをスマートフォン等の端末に組み込んで推定処理を端末内で完結させる環境が整備されつつあり, 推定処理にリアルタイム性が求められるアプリケーションへの活用が期待されている. 本稿では, Android 端末上でトラフィックの輻輳を予測して輻輳制御を行うシステムを想定し, このようなアプリ実装形態の実現可能性を検証すべく, 予めサーバ側で学習した輻輳状況予測モデルを TensorFlow Lite により検証用アプリケーションに組み込み, その性能をサーバ上での予測精度や処理速度と比較し評価する.

キーワード　Android, 無線 LAN, 深層学習, LSTM, TensorFlow Lite, 輻輳制御

1 はじめに

近年, TensorFlow Lite などのように機械学習や深層学習のモデルを, スマートフォン等の端末側に組み込んで, 推定処理を端末内で完結させる環境が整備されつつあり, 推定処理にリアルタイム性が求められるアプリケーションへの活用が期待されている. 例えば端末のカメラで映した物体をリアルタイムに分類する画像分類アプリケーションや, チャットにおいて返信メッセージの候補を提案するスマートリプライアプリケーションなど [1] が注目を集めている.

中でも, 本研究では Android 端末上でトラフィックの輻輳を予測し制御を行うシステムに着目した. 無線環境下でのトラフィックの輻輳は突発的に生じ, 一度起こると制御が難しい上にコントロールしようとしてさらに輻輳が悪化してしまうことがあるため, 輻輳が起こる前にそれを予測していくことが望ましいと考えられる. また, 輻輳の予知に関して, データを端末外に出すセキュリティ上の問題やデータ転送に要する時間等の課題から, 端末内での処理が好ましいと言える.

そこで本稿では, Android 端末上でトラフィックの輻輳を予測して輻輳制御を行うシステムを想定し, そのようなアプリケーション実装形態の実現可能性を検証する. 予め高性能なサーバで端末におけるトラフィックデータを学習したモデルを, TensorFlow Lite により検証用の Android アプリケーションに組み込み, その性能をサーバ上での予測精度や処理速度と比較し実現可能性を評価する.

2 関連研究

2.1 TensorFlow Lite

TensorFlow Lite [2] は Google の機械学習向けソフトウェアライブラリ TensorFlow のモバイル環境向けのライブラリである. TensorFlow Lite では, TensorFlow により学習されたモデルを TFLite Converter を使って TensorFlow Lite 形式に変換し, デバイスに組み込むことによりデバイス上で推論を行うことができる.

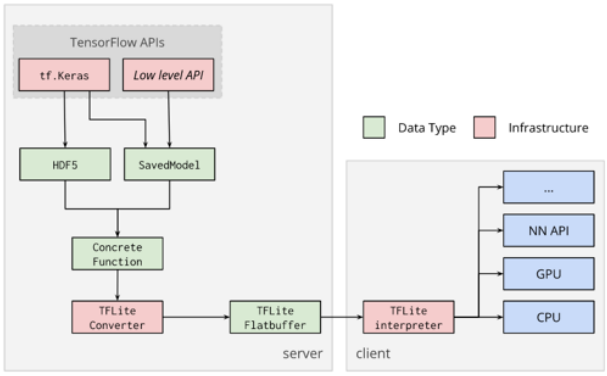

図 1　学習モデル変換の流れ ([3] より引用)

2.1.1 TFLite Converter

TFLite Converter は機械学習モデル形式の変換を行うコンバータ (変換器) である. 図 1 の左側にあるように, サーバ側で TensorFlow により機械学習を行ったのち, その学習モデルを TFLite Converter により TFLite FlatBuffer file に変換する.

TensorFlow Lite のモデル出力形式である TFLite FlatBuffers はデータにアクセスする際にパースを要さずメモリの占有容量が小さいという特徴があり, モバイル端末や組み込みデバイスに適している.

2.1.2 TFLite Interpreter

図 1 の下側にある通り, サーバで生成された TFLite FlatBuffer file をクライアント側で TFLite Interpreter で変換する

ことによりデバイス上で利用することができる．本研究ではこの TensorFlow Lite を用いて輻輳予測モデルを Android 端末に組み込み利用する．

2.2 輻輳制御ミドルウェア

本研究で取り扱うアプリケーション実行形態が対象とする事例の中でも，本稿ではスマートフォン端末上における輻輳制御システムを考える．Android OS ではカーネルに Linux カーネルを用いており，輻輳制御アルゴリズムに CUBIC TCP [4] を用いている．また，近年は新しい TCP 輻輳制御アルゴリズムとして TCP BBR [5] が提案され今後の普及が期待されている．ロスベース TCP の一種である CUBIC TCP では高いスループットを確保するためにアグレッシブな輻輳制御手法を用いているが，特に帯域の狭さやノイズの影響など障害が多く，有線接続に比べ脆弱な無線接続環境においては，膨大な ACK パケットの蓄積によりパケットロスや輻輳が発生してしまう．また，CUBIC TCP と TCP BBR が共存して通信を行った場合，両 TCP のスループットの公平性が低くなることが示されており [6] [7] [8]，特にモバイル環境における不公平性 [9] が示されている．

先行研究において開発された輻輳制御システムである輻輳制御ミドルウェア [10] [11](図 2) は，複数台の Android 端末が同一の無線 LAN アクセスポイントに接続する際に，カーネルモニタ [12] というツールによって取得した情報により端末間で輻輳の状態を把握し，協調して輻輳を制御することを可能にしている．

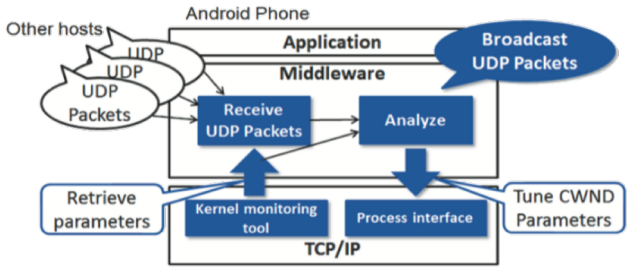

図 2　輻輳制御ミドルウェア

このミドルウェアを用いて輻輳制御を行うには，輻輳の予兆を端末側で検知することが望ましい．トラフィックの予測を行う手法は，筆者らの過去検討では深層学習を用いた手法 [13] や，Trinh らによる LSTM を用いた LTE トラフィックを予測する手法 [14] などが提案されているが，端末側での具体的な実装方法や性能を示すものではなく，サーバなどと比べ計算資源が限られる端末での実現性に懸念がある．

そこで本稿では，この輻輳制御ミドルウェアで想定されるトラフィック予測処理の端末上での実現可能性を検証する．具体的には，TensorFlow Lite を適用した学習モデルを検証用アプリケーションの形で端末に組み込み，端末上で予測処理を実行させ，予測精度や CPU 使用率などの指標で実性能を確認する．

また，このような輻輳制御システムを実際に運用するためには，Wi-Fi 接続環境ごとに構築した学習モデルを端末側に保持させるだけでなく，後になって再構築したモデルを差し替える仕組みも必要になる．たとえば，まず端末が繋がることのできる各 Wi-Fi 環境にあるエッジサーバ等でその環境におけるデータを蓄積して学習しモデルを作成，そして端末は，繋がった Wi-Fi 環境におけるモデルをその都度ダウンロードして利用する，という方法が考えられる．

端末上で TensorFlow Lite を適用した学習モデルを使用する方法は 2 種類あり，1 つは端末内のフォルダにモデルを置いておき，実行時にそのフォルダからモデルを起動させる手法 (ローカルモデルを用いた手法)，そしてもう 1 つはモデルを端末内ではなくサーバ上に上げておき，システムを起動させたタイミングでモデルをサーバ上からダウンロードして使用する手法 (リモートモデルを用いた手法) である．

上述したように，輻輳制御システムの実運用を考慮すると，モデルの差し替えが可能なリモートモデルを用いた手法が適切であると考えられるが，使用する Wi-Fi 環境を移動する度にモデルをダウンロードして利用することがどれほどオーバーヘッドになるかも確認するため，両モデルを用いた手法を比較する形で性能を評価する．

3　学習モデルの作成

本章において，端末内予測処理検証用アプリケーションに組み込む学習モデルを作成する．まず 3.2 においてサーバ上でトラフィックデータを LSTM により学習させる．続いて 3.3 でそのモデルを TensorFlow Lite により端末に組み込める形式に変換する．

3.1 トラフィック予測の概要

2.2 で述べた輻輳制御システムでは，同一アクセスポイントに接続された端末数と RTT の増減比率という 2 つのパラメータから，適切な輻輳ウィンドウの補正値を設定している．本研究が想定するシステムでは，輻輳を検知して制御するのではなく，輻輳が起こる前にその予兆を捉える必要があるため，輻輳を示す指標として接続端末台数を採用し，その変化を予測する．また，端末の接続台数が増えるにつれて合計スループットが大幅に減少することが先述の先行研究で明らかになっていることから，スループット値をもとに接続端末台数を予測するモデルを作成する．

3.2 TensorFlow モデルの作成

本実験では，過去 10 秒間のスループットを入力として，同一アクセスポイント内の接続端末台数を予測し出力するモデルを構築する．そのモデルを作るための学習データを，Android 端末の台数を 1 台から 10 台まで変化させてパケット通信を行わせることで取得し，サーバ上において LSTM を用いて学習させた．また，各 500 秒間の通信のうち，400 秒分を学習データ，100 秒分をテストデータとして学習を行った．

表 1 は学習データ，テストデータのそれぞれを入力データとして予測させた際の台数ごとの正解率である．台数が増えると端末 1 台あたりのスループットは減少していくが，台数が増え

れば増えるほどその減分が小さくなっていくため, 分類が難しくなり精度が下がる傾向にあることがわかる. さらに詳細に予測結果を分析した結果, 特に 7〜8 台の時には 5〜6 台もしくは 9〜10 台との判別が難しくなっていることがわかり, テストデータでの精度が 3〜4 割程度にまで下がってしまっている.

表 1　10 択予測モデルにおける正解率

	学習データ	テストデータ
1 台のとき	99.2%	96.5%
2 台のとき	98.5%	94.7%
3 台のとき	96.7%	93.0%
4 台のとき	75.0%	66.2%
5 台のとき	74.9%	65.3%
6 台のとき	55.2%	52.7%
7 台のとき	66.9%	43.1%
8 台のとき	69.4%	29.5%
9 台のとき	92.5%	44.6%
10 台のとき	90.5%	40.4%
全体	81.9%	62.6%

しかしながら, 本研究では輻輳が起こることを事前に予測することを目的としているため, 必ずしも接続端末台数を完璧に的中させる必要はなく, 端末台数の多寡を把握できれば制御に役立てることができると考えられる.

そこで次は, 過去 10 秒間のスループットを入力として, 出力する予測端末台数を 1 台〜10 台の 10 択ではなく, 1 台/3 台/5 台/7 台/9 台の 5 択に減らしモデルを作成した. モデルを作るための学習データを, Android 端末の台数を 1 台/3 台/5 台/7 台/9 台と 5 段階に変化させてパケット通信を行わせることで取得し, サーバ上において LSTM を用いて学習させた. また, 各 500 秒間の通信のうち, 400 秒分を学習データ, 100 秒分をテストデータとして学習を行った. 表 2 は学習データ, テストデータのそれぞれを入力データとして予測させた際の台数ごとの正解率である. 台数が増えるにつれて精度が少し下がる傾向にあるが, 全体の正解率は学習データにおいて 92%, テストデータにおいて 84%を超えており, 高い精度で予測ができていることがわかる.

表 2　5 択予測モデルにおける正解率

	学習データ	テストデータ
1 台のとき	100%	100%
3 台のとき	99.2%	94.7%
5 台のとき	92.6%	82.0%
7 台のとき	77.7%	62.2%
9 台のとき	93.0%	80.9%
全体	92.5%	84.4%

さらに, より実際の輻輳制御環境を想定し, 過去 10 秒間のスループットを入力として, 予測される端末台数が設定した閾値に達していれば 1(制御を行う), 下回っていれば 0(制御を行わない) を出力するモデルを構築する. また, 各 500 秒の通信のうち, 400 秒分を学習データ, 100 秒分をテストデータとして学習を行い, 閾値を変化させて 4 パターンのモデルを作成した.

表 3 は学習データ, テストデータのそれぞれを入力データとし, 閾値を変化させて予測させた際の正解率である. 全てのケースで正解率が 9 割を超えており, 非常に高い精度で予測ができていることがわかる. この実験において作成したモデル (TF モデル) のうちの 1 つを 3.3 で端末に組み込める形式に変換する.

表 3　制御判断モデルにおける正解率

閾値	学習データ	テストデータ
3 台	100%	100%
5 台	99.2%	98.4%
7 台	94.9%	93.0%
9 台	98.6%	93.5%

3.3　TensorFlow Lite モデルの作成

3.2 にて作成した TF モデルを 2.1 の流れに沿って TFLite Converter を用いて TensorFlow Lite 対応形式に変換した. この変換前後のモデルについて, モデルサイズを比較した結果が表 4 である. モデル変換によりサイズが 10 分の 1 程度まで圧縮されていることがわかる.

表 4　モデル変換前後のモデルサイズの比較

	変換前 (TF モデル)	変換後 (TFLite モデル)
10 択予測	876KB	81.6KB
5 択予測	873KB	80.3KB
制御判断	870KB	79.6KB

続いてこの変換後のモデル (TFLite モデル) により, TFLite Interpreter を用いてサーバ上で予測を行い, その結果を変換前のモデルである TF モデルによる予測結果と比較したところ, 両モデルによる予測結果は完全に一致し, モデル変換によって予測結果が変わることはないことが確認できた. また, 変換前後のモデルによりサーバ上で予測を行った際の予測時間を比較した結果が表 5 である. 変換によりモデルが軽量化され, 予測にかかる時間も 30 分の 1 以上と大幅に短縮される結果となった.

表 5　モデル変換前後の予測時間の比較

	変換前 (TF モデル)	変換後 (TFLite モデル)
10 択予測	35.9ms	1.00ms
5 択予測	37.6ms	1.00ms
制御判断	39.6ms	1.00ms

この TFLite モデルをアプリケーションとして端末に組み込む.

4　TFLite モデルの性能評価

4.1　学習モデルを組み込んだアプリケーション

3.3 にて作成した TFLite モデルを組み込む図 3 のような Android アプリケーション (検証用アプリケーション) を, Firebase の ML Kit [15] を利用し Android Studio [16] で実装した. 画面右下の ”predict” ボタンを押すと, 端末内のストレージにあるスループットデータのファイルから 10 秒間分の入力値を

取得し, その時点での予測接続端末台数から制御が必要か否かを出力するという実装になっている.

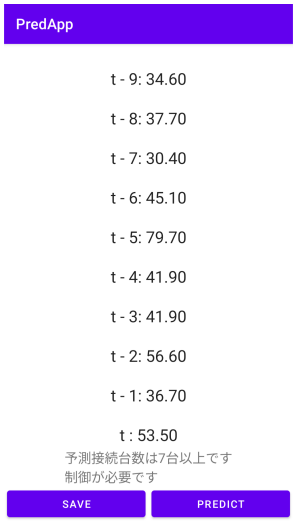

図 3　TFLite モデルを組み込んだ輻輳予測アプリケーション

TFLite モデルをアプリケーションに組み込む方法は, 2.2 で述べた通りローカルモデル, リモートモデルの 2 種類存在する. 前述の通りローカルモデルの場合は TF Lite モデルを端末内のフォルダに置いておき, リモートモデルの場合は TFLite モデルをサーバ上 (Firebase コンソール) に置いておく代わりに端末内には置かず, アプリケーション起動時に端末にダウンロードする. 次節以降でこのサーバ, 端末の両環境での TFLite モデルの性能を比較し評価する. なお, 検証に使用した機器の性能は表 6 の通りである.

表 6　使用機器の性能

Android 端末	Model number	Google Pixel 3
	Firmware version	12
	CPU	Octa-core (2.5GHz × 4 +1.6GHz × 4)Kryo
	Memory(Internal)	64GB, 4GB RAM
サーバ	OS	Windows 10 Home
	CPU	Inter(R) Core(TM) i7-6700T
	Main Memory	16.0GB

4.2　予測精度

サーバと端末の実行環境の違いによる予測精度の変化の有無を確認するため, 両環境での TFLite モデルによる予測結果を比較したところ, 結果は完全に一致し, 端末上においてサーバと同じ精度での予測が可能であることがわかった.

4.3　予測時間

続いてサーバ上と検証用アプリケーションで予測を行う際にかかる時間をそれぞれ計測したところ, 表 7 のような結果になった. リモートモデルでは, 表に示されている予測時間に加え, アプリケーション初回起動時に限りサーバからモデルをダウンロードするのに約 35 ミリ秒を要しているが, ローカルモデルとリモートモデルで予測そのものに掛かる時間はほぼ変わらない結果となった. これらの検証アプリケーション上における予測時間 (3〜4 ミリ秒) をサーバでの TFLite モデルの予測時間と比較すると, 端末上での予測時間は 2〜4 倍程度の時間を要しているが, 今回想定する輻輳制御によれば, 輻輳制御の周期が 0.3 秒 (300 ミリ秒) 程度であるため, この速度差は許容範囲内であるといえる.

表 7　端末上での予測時間の比較

	ローカルモデル	リモートモデル
10 択予測	4.03ms	3.22ms
5 択予測	3.63ms	3.64ms
制御判断	3.15ms	3.14ms

4.4　CPU 使用率

最後に, 検証用アプリケーション使用時の Android 端末の CPU 使用率を Android Studio の Android Profiler [17] で計測した (表 8). 検証用アプリケーションにおける CPU 使用率は, ローカルモデルとリモートモデルではほとんど差は見られず, 予測処理時において 7〜8%となった. TensorFlow Lite の公式サイトが紹介しているサンプルの機械学習アプリケーションを用いて Android 端末上で同様の検証を行ったところ, 予測処理時における CPU 使用率は 15%という結果となり, この結果と比較しても十分低い値であるといえる.

表 8　端末上での予測時における CPU 使用率

	ローカルモデル	リモートモデル
10 択予測	8%	7%
5 択予測	8%	8%
制御判断	8%	7%

5　まとめと今後の課題

本稿では, Android 端末上機械学習処理の実現可能性を検証すべく, Android 端末上でトラフィックの輻輳を予測して輻輳制御を行うシステムを想定し, サーバで学習させた輻輳予測モデルを, TensorFlow Lite により検証用の Android アプリケーションに組み込み, ローカルモデル, リモートモデルの 2 つの手法において, その性能をサーバ上でのパフォーマンスと比較し評価した. 予測精度に関しては, サーバと同等の精度での予測が可能であること, また処理速度に関しては, サーバ上と比較すると僅かに劣るものの十分有用なものであることがわかり, 端末上輻輳予測処理の実現可能性が示された.

今後の課題としては, 本研究で目指している輻輳の事前検知を実現させるため, より早い段階での高精度な予測を実現させること, そして最終的には端末上で完結する輻輳制御システムの実現を目指していきたいと考えている.

profile/android-profiler?hl=ja (参照 2020-11)

謝　　辞

本研究は一部, JST CREST JPMJCR1503 の支援を受けたものである.

文　　献

[1] TensorFlow Lite を使用しているアプリの例, https://www.tensorflow.org/lite/examples?hl=ja (参照 2019-10)
[2] TensorFlow Lite, https://www.tensorflow.org/lite?hl=ja (参照 2019-10)
[3] TensorFlow Lite コンバータ, https://www.tensorflow.org/lite/convert?hl=ja (参照 2019-10)
[4] Sangtae Ha, Injong Rhee, and Lisong Xu. 2008. CUBIC: a new TCP-friendly high-speed TCP variant. SIGOPS Oper. Syst. Rev. 42, 5 (July 2008), 64–74. DOI: https://doi.org/10.1145/1400097.1400105
[5] Neal Cardwell, Yuchung Cheng, C Stephen Gunn, Soheil Hassas Yeganeh, Van Jacobson, “BBR: Congestion-based congestion control,” ACM Queue, 14(5):50, 2016.
[6] M. Hock, R. Bless and M. Zitterbart, ”Experimental evaluation of BBR congestion control,” 2017 IEEE 25th International Conference on Network Protocols (ICNP), Toronto, ON, 2017, pp. 1-10. DOI: 10.1109/ICNP.2017.8117540
[7] K. Miyazawa, K. Sasaki, N. Oda and S. Yamaguchi, ”Cycle and Divergence of Performance on TCP BBR,” 2018 IEEE 7th International Conference on Cloud Networking (CloudNet), Tokyo, 2018, pp. 1-6, DOI: 10.1109/CloudNet.2018.8549411
[8] K. Sasaki, M. Hanai, K. Miyazawa, A. Kobayashi, N. Oda and S. Yamaguchi, ”TCP Fairness Among Modern TCP Congestion Control Algorithms Including TCP BBR,” 2018 IEEE 7th International Conference on Cloud Networking (CloudNet), Tokyo, 2018, pp. 1-4, DOI: 10.1109/CloudNet.2018.8549505
[9] F. Li, J. W Chung, X. Jiang, and M. Claypool, “TCP CUBIC versus BBR on the Highway,” Proc. Passiv. Activ. Meas. Conf. (PAM), March 26-27, 2018. DOI: 10.1007/978-3-319-76481-8_20
[10] Hiromi Hirai, Saneyasu Yamaguchi, and Masato Oguchi: ”A Proposal on Cooperative Transmission Control Middleware on a Smartphone in a WLAN Environment”, Proc. IEEE WiMob2013, pp.710-717, October 2013
[11] Ai Hayakawa, Saneyasu Yamaguchi, Masato Oguchi:”Reducing the TCP ACK Packet Backlog at the WLAN Access Point” Proc. ACM IMCOM2015, 5-4, January 2015.
[12] Kaori Miki, Saneyasu Yamaguchi, and Masato Oguchi: “Kernel Monitor of Transport Layer Developed for Android Working on Mobile Phone Terminals”, Proc. ICN2011, pp.297-302, January 2011
[13] Yamamoto A. et al. (2019) Prediction of Traffic Congestion on Wired and Wireless Networks Using RNN. Proceedings of the 13th International Conference on Ubiquitous Information Management and Communication (IMCOM) 2019. IMCOM 2019. Advances in Intelligent Systems and Computing, vol 935. Springer, Cham
[14] H. D. Trinh, L. Giupponi and P. Dini, ”Mobile Traffic Prediction from Raw Data Using LSTM Networks,” 2018 IEEE 29th Annual International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC), Bologna, 2018, pp. 1827-1832, DOI: 10.1109/PIMRC.2018.8581000
[15] MLKit, https://firebase.google.com/docs/ml-kit?hl=ja (参照 2020-05)
[16] Android Studio, https://developer.android.com/studio/intro?hl=ja (参照 2019-07)
[17] Android Profiler, https://developer.android.com/studio/