Vol.2020-IOT-48 No.19
2020/3/3

IoTストリームデータ処理のためのソフトウェアライブラリSINETStreamの開発

竹房 あつ子[1,a)]　孫 静涛[1,b)]　藤原 一毅[1,c)]　吉田 浩[1,d)]　合田 憲人[1,e)]

概要：多様なIoTデバイスから生成されるビッグデータをクラウドで収集し，解析することで，自動運転やスマートホームから娯楽，医療，農林水産業への活用などより高度なサービスの創出が期待されている．しかしながら，このようなIoTアプリケーションを構築するには，ネットワークを介した大量ストリームデータの収集，データの安全性の確保，処理応答時間の削減のための性能チューニング等の課題があり，アプリケーションを専門とする開発者にとっては容易ではない．本研究では，高度なIoTアプリケーションの開発を容易にするためのソフトウェアライブラリSINETStreamを開発する．SINETStreamでは，IoTストリームデータ処理のためのメッセージングAPI，セキュリティ機能，および多様なメッセージング基盤ソフトウェアに対応するためのSPI (Service Provider Interface) を提供する．評価では，SINETStreamの基本性能を示すとともに，SINETStreamを用いて実際にIoTアプリケーションを構築し，その実用性を示す．

1. はじめに

多様なセンサデバイス（センサ）から生成されるビッグデータを大容量ストレージと高性能計算機が利用可能なクラウドで収集し，解析することで，自動運転，スマートホームから娯楽，医療，農林水産業への活用など，より高度なサービスの創出が期待されている．学術分野では，学術情報ネットワークSINET5のアクセス環境としてモバイル網を提供するSINET広域データ収集基盤 [1] の実証実験が進められており，センサから学術機関およびクラウドの計算資源まで，隔離された安全な環境で様々なIoT（Internet of Things）アプリケーションを構築することが可能になってきた．

しかしながら，IoTアプリケーションの構築，運用には，(1) 広域ネットワークを介した大量ストリームデータの収集，(2) データの安全性の確保，(3) 処理応答時間の削減のための性能チューニング等の課題があり，アプリケーションを専門とする研究者にとって容易ではない．

(1) ストリームデータの収集では，多数のセンサからの大量の小規模ストリームデータを欠損なくまた効率よく行うのは容易ではない．オブジェクトストレージは，そのような通信パターンに対して書き込み性能が低くなることが知られている [2]．また，サーバ計算機で直接そのようなストリームデータを受け取る場合は，サーバ計算機で行っている解析処理本体の性能が劣化する恐れがある．よって，MQTT (Message Queue Telemetry Transport)[3] ブローカのようなメッセージを中継するメッセージング基盤システムを利用したPublish-Subscribe（Pub-Sub）型非同期メッセージングを行う必要がある．しかしながら，様々なメッセージング基盤システムが開発されており，アプリケーション開発初期の段階でどのシステムを利用すればよいか判断するのは難しい．Amazon Kinesis[4] やGoogle Cloud Pub/Sub[5] のように，IoTストリームデータ収集のための商用クラウドPub-Subサービスも複数あるが，利用するクラウドへのロックインは避けられない．

(2) センサから収集されるデータを安全に収集することは，特に個人情報に関わるデータや新しい技術の創出等に関わる秘匿性の高いデータでは非常に重大な課題となる．SINET広域データ収集基盤は，ネットワークの隔離による安全な環境を提供することができるが，クラウドや学術機関での障害や人的操作ミスによる情報流出に備える必要がある．データへのアクセスに関する認証・認可やデータの暗号化等の仕組みを適用するなどの対応が必要となる．

IoTアプリケーションでは，アプリケーションごとに要求される応答時間までにデータの収集，解析・結果のフィードバックの処理を終わらせる必要があるが，通信状況や利

[1] 国立情報学研究所
[a)] takefusa@nii.ac.jp
[b)] sun@nii.ac.jp
[c)] ikki@nii.ac.jp
[d)] h-yoshida@nii.ac.jp
[e)] aida@nii.ac.jp

用する計算サーバの性能や所在により応答時間が変わってくるため，(3) 高度な性能チューニングが重要になる．自動運転ではミリ秒，見守り等のサービスでは数秒程度の応答時間が求められる一方，定期的な観測からシステム全体の効率化を図るようなサービスでは長い応答時間を許容するものもある．アプリケーションごとに設置・利用条件や応答時間，配備コストへの要求等が異なるため，エッジサーバの活用やクラウドでの高性能計算機の利用など，個々のケースに応じた高度なチューニングが求められる．よって，アプリケーションと計算基盤の専門家が協同して構築することを可能にする枠組みがあることが望ましい．商用クラウド Pub-Sub サービスの API の利用はその一つの解であると言えるが，同様にロックインの問題がある．

本研究では，高度な IoT アプリケーションの開発を容易にするためのソフトウェアライブラリ SINETStream を開発する．SINETStream では，IoT ストリームデータ処理のためのメッセージング API（Application Programming Interface），セキュリティ機能，および SPI（Service Provider Interface）の提供を行う．ストリーム処理のための基盤ソフトウェアやクラウドサービスを抽象化するメッセージング API を提供することで，アプリケーション研究者がメッセージング基盤システムを意識することなくセンサデータの収集・解析のためのデータの書き込み・読み出しを容易に行えるようにする．セキュリティ機能として，ユーザおよびホストの認証・認可，データ暗号化の機能を提供する．また，SPI により多様なミドルウェアやクラウドサービスを利用できるようにし，アプリケーションと計算基盤の専門家が協同して IoT アプリケーションを構築可能にする枠組みを提供する．

評価では SINETStream のオーバヘッドやセキュリティ機能を用いた場合の性能劣化を LAN 環境とモバイル環境で調査する．また，SINETStream を用いて IoT アプリケーションが構築できることを実証する．

2. IoT ストリームデータ処理

IoT アプリケーション構築事例として，SINET 広域データ収集基盤を用いたオンラインビデオ解析実証実験 [2] について紹介する．実験の概要と用いたソフトウェアの構成について説明する．

2.1 オンライン動画像解析実証実験の概要

学術情報ネットワーク SINET5 とモバイル通信を直結したサービス「広域データ収集基盤」では，SINET 接続用の SIM を装着したセンサ等が取得したデータを，SINET の L2VPN で接続されたクラウド等の計算機を用いて安全に収集，解析することができる．実証実験では，センサ端末のカメラで取得した動画像から複数の静止画を生成し，その静止画を順次 SINET のモバイル網経由でクラウド上の

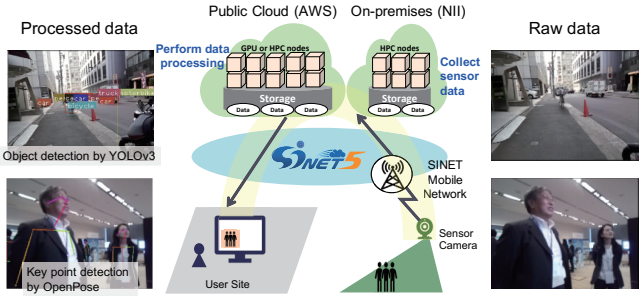

図 1　実証実験の概要．

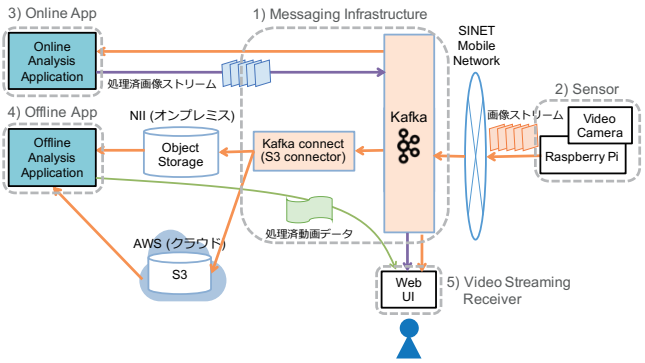

図 2　リアルタイムビデオ処理機構プロトタイプシステムのソフトウェア構成．

メッセージング基盤に送信・収集し，そのリアルタイム画像解析処理ができることを示した．

図 1 に実証実験実施時の利用者端末で取得した画像と実験の概要を示す．図 1 の右側上下に並んでいる画像は，センサ端末のカメラで取得した動画像をクラウド上のメッセージング基盤を介して利用者端末にストリーム配信された結果が表示されている．右側上には屋外を移動しているセンサ端末のカメラ画像，右側下には屋内のセンサ端末のカメラ画像が表示されている．また，左側上下に並んでいる画像は，処理前画像をメッセージング基盤から受け取り，クラウドで何らかの処理をした処理後画像を，メッセージング基盤を介して利用者端末に出力された結果となっている．ここで，左側上の画像は YOLO v3[6], [7] を用いてオブジェクト抽出を行った結果，左側下の画像は OpenPose[8], [9], [10], [11], [12] を用いて人のキーポイントを抽出した結果が表示されている．

図 2 は本実験で用いたプロトタイプシステムのソフトウェア構成であり，主に以下のソフトウェアで構成される．

1） メッセージング基盤ソフトウェア
2） センサ用プログラム
3） オンラインアプリケーションプログラム
4） オフラインアプリケーションプログラム
5） ストリーム画像出力プログラム

ここで，アプリケーション開発者がコーディングするソフトウェアをプログラムと区別して呼ぶことにする．1）メッセージング基盤ソフトウェアには，Apache Kafka（Kafka）

を用いた．Kafka は，Pub/Sub 形式のメッセージング基盤の一つであり，センサ等の Producer から送信されるメッセージを Broker で一時的に収集、永続化し、データ処理プログラム等の Consumer に提供する．Broker はクラスタ構成をとることで，スケールアウトが可能になっている．個々のストリーミングデータは Topic と呼ばれるカテゴリ単位で管理され，Topic 名，値，タイムスタンプからなるレコードとして送信，格納される．2）センサ用プログラムは，Kafka の Producer として動作する．センサ端末には Raspberry Pi とカメラモジュールを用い，USB 接続で SINET SIM を装着したモバイルルータを用いて SINET L2VPN 経由で Kafka Broker に静止画を繰り返し送信する．3）オンラインアプリケーションプログラムでは，YOLO v3 の PyTorch 実装と OpenPose を用いてそれぞれクラウドの GPU ノード上で画像を処理した．各オンラインアプリケーションプログラムでは，Kafka の Consumer および Producer の機能を利用している．また，オブジェクトストレージに格納された複数静止画をまとめて処理する 4）オフラインアプリケーションプログラムも用意した．利用者端末では，処理前および処理後静止画のストリームデータを動画像として表示させる 5）ストリーム画像出力プログラムを用意した．ストリーム画像出力プログラムもまた Kafka Consumer プログラムであり，Broker から受け取ったデータを順次利用者端末に出力することができる．

2.1.1 アプリケーションの開発における技術的課題

本実証実験では，1) のメッセージング基盤ソフトウェアとして Kafka を用いたが，他のメッセージング基盤が適切であると判断された場合には 1) を入れ替えるだけでなく，2)，3)，5) のプログラムの修正が必要となる．また，センサの認証・認可やデータの暗号化など，セキュリティ機能を高める場合も，各メッセージング基盤ソフトウェアやストリーム処理クラウドサービスの機能や利用方法を習得し，実装する必要がある．

さらに，より短い応答時間が要求されるようなアプリケーションでは，センサとクラウドの間の計算資源を活用するエッジコンピューティングやフォグコンピューティングを行う場合がある．そのような構成変更をアプリケーションを専門とする研究者が行うのは負担が大きいため，計算基盤研究者と協同して構築できる枠組みが必要となる．

3. SINETStream の開発

高度な IoT アプリケーションの開発を容易にするためのソフトウェアライブラリ SINETStream を開発する．SINETStream では，IoT ストリームデータ処理のための抽象化 API，セキュリティ機能，および SPI（Service Provider Interface）の提供を行う．

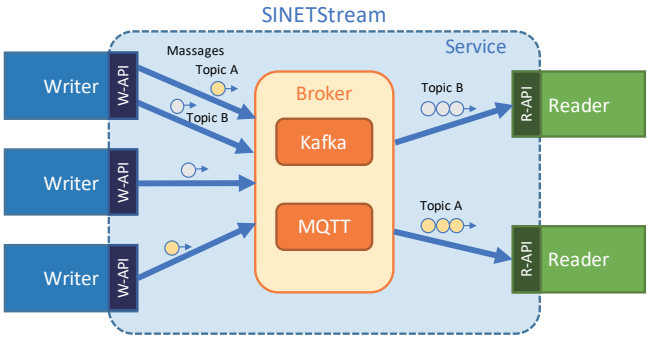

図 3 SINETStream の概念図.

```
from sinetstream import MessageWriter

writer = MessageWriter('service-1', 'topic-1')
with writer as f:
    f.publish(b'Hello! This is the 1st message.')
    f.publish(b'Hello! This is the 2nd message.')
```

図 4 Writer の Python プログラム例.

```
from sinetstream import MessageReader

reader = MessageReader('service-1', 'topic-1')
with reader as f:
    for msg in f:
        print(msg.value)
```

図 5 Reader の Python プログラム例.

3.1 SINETStream API

SINETStream では，トピックベースの Pub-Sub 型非同期メッセージングモデルを前提としている．トピックは，メッセージを送受信するときの論理的なチャネルを表している．図 3 に SINETStream の概念図を示す．SINETStream では，センサデータを送信する IoT デバイス側の Publisher プログラムを Writer，収集したデータを活用するサーバ計算機等で利用する Subscriber のプログラムを Reader と呼ぶ．Writer と Reader では，SINETStream が提供する API を実装することで図 3 の中央にあるメッセージブローカ（ブローカ）を介してセンサデータの送受信が可能となる．ブローカは，SINETStream に対応したものであれば任意のブローカソフトウェアまたはクラウド Pub-Sub サービスを利用することができる．また，プログラムの可搬性を高めるため，各メッセージブローカとの通信に必要となるパラメータは別途定義できるようにした．SINETStream では，このパラメータセットを Service と呼ぶことにする．以降で，Writer，Reader の API および Service の設定方法について説明する．

3.1.1 Writer/Reader API

SINETStream では，Python および Java の Writer，Reader API を提供しており，これによりブローカへの接続／切断，メッセージの送信／受信の 4 つの操作が可能

```
service-1:
  type: kafka
  brokers:
    - kafka-1:9092
    - kafka-2:9092
    - kafka-3:9092
    - kafka-4:9092
service-2:
  type: mqtt
  brokers: 192.168.2.105:1883
```

図 6　.sinetstream_config.yml でのブローカ関連パラメータの設定例.

となっている．図 4，図 5 に，SINETStream Python API を用いて Writer と Reader でテキストベースのメッセージを送受信する際のプログラム例を示す．SINETStream Java API もほぼ同様にプログラムを作成することができるため，本稿では説明を割愛する．各 API の詳細については，SINETStream のウェブサイト [13] を参照されたい．

図 4 は，Service service-1 に対応するブローカのトピック topic-1 の論理チャネルに対して Writer がメッセージを送信する例を示している．MessageWriter インスタンス生成時に Service 名，トピック名を指定し，with 文内で publish メソッドを用いてメッセージを送信する．メッセージはデフォルトではバイト列で送信することになっているが，インスタンス生成時にメッセージのデータタイプやシリアライザを指定することもできる．また，consistency パラメータによりメッセージ配信の信頼性のレベルを AT_MOST_ONCE，AT_LEAST_ONCE または EXACTLY_ONCE のいずれかで指定することができる．少なくとも 1 つは必ず送信したい場合は AT_LEAST_ONCE を指定すればよい．

図 5 は，Reader が service-1 に対応するブローカの topic-1 の論理チャネルからメッセージを受信する例を示す．同様に，MessageReader インスタンス生成時に Service 名，トピック名を指定し，with 文内でメッセージを受信する．メッセージは，イテレータで順次受信することができる．メッセージは Message オブジェクト内に格納されており，msg.value にメッセージ本体が保存されている．他にも，トピック名，メッセージ送信時のタイムスタンプ，利用しているブローカの生メッセージオブジェクト情報が取得できる．

図 4, 図 5 いずれの場合も with 文開始時に SINETStream でブローカへの接続，終了時に切断操作を行うため，プログラマは明示的にそれらの処理を書く必要はない．

3.1.2　Service の設定

SINETStream では，ブローカの実装を抽象化し，プログラムの可搬性を高めるため，YAML ベースの Service 設定ファイル .sinetstream_config.yml 内でブローカ等に関するパラメータセットを Service として指定する．

```
service-aes-1:
  type: mqtt
  brokers: 192.168.2.105:1883
  username_pw_set:
    username: user01
    password: pass01
  tls:
    ca_certs: /etc/sinetstream/ca.pem
    certfile: certs/client.pem
    keyfile: certs/client.key
  crypto:
    algorithm: AES
    key_length: 256
    mode: EAX
    key_derivation:
      algorithm: pbkdf2
      iteration: 10000
    password: secret-000
```

図 7　.sinetstream_config.yml での認証および暗号化の設定例.

図 6 に .sinetstream_config.yml の例を示す．ここでは，service-1 と service-2 の 2 つの Service を予め定義している．service-1 では Kafka，service-2 では MQTT ベースのブローカを利用する際に必要となるエンドポイントなどのパラメータを指定している．

Service 設定ファイルを変更するだけで，Writer や Reader のプログラムを修正することなく，容易に異なるブローカ実装を利用することができる．また，Python API，Java API いずれの場合も同じ Service 設定ファイルを利用することができる，SINETStream API を呼び出す際に指定するパラメータのデフォルト値も設定できるという利点もある．

3.2　SINETStream のセキュリティ機能

SINETStream では，セキュリティ機能として認証・認可とデータ暗号化を有効にすることができる．

3.2.1　認証・認可

認証・認可は，各ブローカ実装でサポートしている認証・認可機能を，SINETStream の Service 設定ファイルまたは API のパラメータで統一的に設定できるようにした．

認証は，事前に利用するブローカに登録されたユーザだけが接続を許可される機能であり，TLS および個々のブローカ実装のパスワード認証機能等が利用できる．TLS では，クライアントの認証，サーバの認証，通信内容の暗号化を行うことができる．SINETStream では，クライアントは Writer または Reader，サーバはブローカ実装に相当する．公開鍵証明書を用いて，クライアントおよびサーバの認証を行うことが前提とされているが，SINET L2VPN などの閉域網で利用するケースもあるため，事前に承認された自己署名証明書の利用も可能としている．図 7 に Service

設定ファイル.sinetstream_config.yml での認証機能の設定例を示す．tls のパラメータ設定で，CA 証明書，クライアント証明書，秘密鍵等を指定することができる．また，各ブローカ実装固有のパスワード認証も利用できる．図 7 では，MQTT ブローカの認証機能を username_pw_set パラメータで利用可能にしている．

認可は，利用するブローカに登録されたユーザが行える操作，具体的にはあるトピックに対するメッセージの読み書きを制限する機能であり，ブローカ側の認可機能をそのまま利用している．SINETStream 独自の認可機能は現状では提供していないが，各ブローカで認可処理の失敗による例外が発生した場合には，SINETStream API でも例外を発生させるようにした．

また，認証・認可機能を有効にする場合は SINETStream 側での設定だけでなく，利用するブローカ本体での設定も必要となる．ブローカ本体の設定については，SINETStream のウェブページ [13] で設定例を紹介している．

3.2.2 暗号化

SINETStream は，TLS による暗号化と SINETStream 自体が提供する「データ暗号化」の機能を提供する．Service 設定ファイルで tls パラメータが設定された場合は，送受信時にデータの暗号化がなされる．TLS による暗号化では，通信中のみ暗号化されるため，ブローカでメッセージを受け取ると復号化され，通常はブローカ内で一時的に保存される．しかしながら，個人情報等の秘匿性の高いデータを扱う場合は，Writer プログラムと Reader プログラム以外ではデータを暗号化して管理したいという要求も考えられる．よって，SINETStream がデータを直接暗号化するデータ暗号化機能も提供する．

図 7 では，.sinetstream_config.yml の crypto でデータ暗号化機能を利用する際のパラメータ設定例を示している．ここでは，利用する暗号アルゴリズム，鍵長，暗号利用モード，鍵導出関数に関するパラメータセット，パスワードをそれぞれ指定している．

3.3 SINETStream SPI

Kafka や MQTT ブローカ，クラウド Pub-Sub サービスのような従来のブローカソフトウェアだけでなく，エッジコンピューティング等のメッセージング基盤ソフトウェアにも対応可能とするため，SINETStream では SPI を提供する．SINETStream v1.0 では，Kafka と MQTT ブローカをサポートしているが，SPI を実装したプラグインを追加することで多様なメッセージング基盤ソフトウェアに対応することができる．

図 8 に SINETStream のモジュール構成を示す．青枠内に示した部分は SINETStream 本体であり，API および SPI，メッセージング基盤に必要とされる共通の機能とプラグイン管理機能を提供している．緑枠内で示した部分は

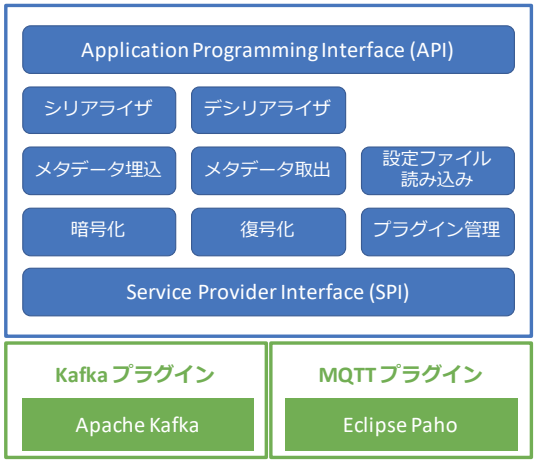

図 8 SINETStream のモジュール構成．

SINETStream のプラグインを表しており，SPI を介して独立したプラグインモジュールを組み合わせることができる．SINETStream の Python 実装では，Kafka プラグインと MQTT プラグインを提供している．Kafka プラグインでは kafka-python クライアント [14] を用いて Kafka ブローカとのメッセージ送受信機能を提供している．MQTT プラグインでは，Eclipse Paho[15] の機能により MQTT ブローカとの送受信を可能にしている．同様に，SINETStreamJava 実装でも Kafka プラグインと MQTT プラグインを提供している．

4. 評価

SINETStream の実用性を示すため，SINETStream の利用の有無と TLS およびデータ暗号化を用いた場合のオーバヘッドを SINET モバイル環境および LAN 環境で調査する．また，SINETStream を用いて IoT アプリケーションを構築できることを示す．

4.1 予備実験

評価では，Writer からブローカへデータを書き込んでから Reader でブローカに到着したデータの読み込みが完了するまでの時間を測定し，SINETStream 利用の有無，TLS またはデータ暗号化を行った場合のスループットを比較した．SINETStream の Java 実装を用い，ブローカには Apache Kafka v2.12-2.3.0 と Mosquitto v1.6.2 を用いた．SINETStream の Python 実装を用いた実験も行ったが，Python 自体の Global Interpreter Lock の影響でスループット性能が低かったため，Java 実装で比較する．

センサで集めた情報をサーバ計算機で処理する IoT アプリケーションシナリオを想定し，Writer には Raspberry Pi（raspi と表記）と AWS の m5.large インスタンス（aws と表記），ブローカと Reader には同じ仕様の AWS VM を用いた．ネットワーク環境は Writer とブローカ間では SINET 広域データ収集基盤の提供するモバイル環境（raspi を用

表 1　予備実験環境.

Writer 計算機	[raspi] Raspberry Pi 3 Model B Plus Rev 1.3 または [aws] Amazon EC2 m5.large (vCPU=2), AZ=ap-northeast-1a, CentOS7
ブローカおよび Reader 計算機	[aws] Amazon EC2 m5.large (vCPU=2), AZ=ap-northeast-1a, CentOS7
Writer-ブローカ間ネットワーク	SINET モバイル環境，または LAN 環境（最大 10Gbps）
ブローカ-Reader 間ネットワーク	LAN 環境（最大 10Gbps）

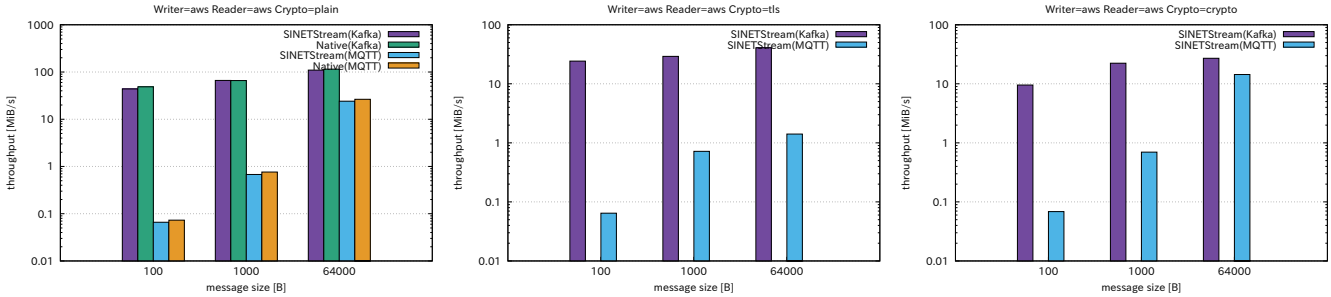

図 9　LAN 環境での測定結果．左は SINETStream の有無の比較，中央は TLS 通信暗号化の結果，右はデータ暗号化の結果を表す．

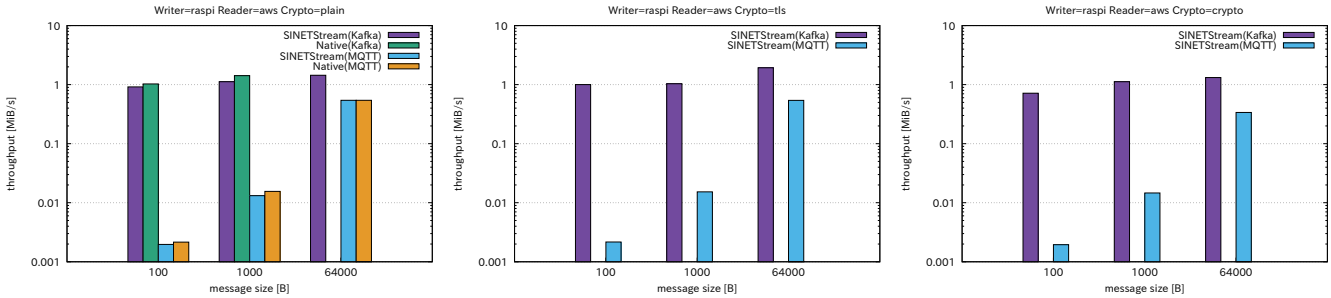

図 10　Writer-ブローカ間でモバイル環境を用いた場合の測定結果．左は SINETStream の有無の比較，中央は TLS 通信暗号化の結果，右はデータ暗号化の結果を表す．

表 2　SINET モバイル環境での ping（RTT）の結果.

src → dest	最小値	平均値	最大値	標準偏差
raspi → aws	28.049 ms	37.384 ms	47.254 ms	5.403 ms

表 3　SINET モバイル環境での iperf3（通信スループット）の結果.

src → dest	平均通信スループット
raspi → aws	12.1 Mbps
aws → raspi	9.90 Mbps

いる場合）と LAN 環境（aws を用いる場合），ブローカと Reader 間は LAN 環境を用いた．表 1 に，予備実験環境をまとめた．

評価に利用した SINET モバイル環境のベースライン通信性能を ping および iperf で測定した．表 2 に ping で 10 回測定した RTT（Round-Trip Time）の最小値，平均値，最大値，標準偏差，表 3 に iperf3 で 10 秒間測定した際の平均通信スループット値を示す．RTT は，40ms 前後の値を示しており，通信スループットはアップストリームでは 12.1Mbps，ダウンストリームはやや小さく 9.90Mbps となっていた．ただし，これらは通信キャリア，測定場所におけるモバイルネットワークの電波状況等により変化するので注意が必要である．

図 9 に LAN 環境での測定結果，図 10 にモバイル環境での測定結果を MiB/s で示す．いずれも Kafka と MQTT ブローカを用いたときの 10 秒間の測定での平均スループット値であり，左は SINETStream の有無の比較，中央は TLS 通信暗号化の結果，右はデータ暗号化の結果を表している．各グラフは，横軸は送信するメッセージサイズ 100B，1KiB，64KiB，縦軸はスループットを MiB/s で示している．

図 9 では，Kafka，MQTT いずれのブローカを用いた場合も SINETStream のオーバヘッドは僅かであること，TLS およびデータ暗号化を行うとスループットが大幅に劣化することが分かる．TLS とデータ暗号化の比較では，データ暗号化の方が TLS より性能劣化がやや大きいが，送信メッセージサイズが大きくなると MQTT では通信スループットが TLS より高くなる傾向が示された．

図 10 のモバイル環境での結果では，SINETStream および TLS，データ暗号化による通信スループットの劣化は僅かであり，モバイルネットワークの性能に律速されていた．

4.2　SINETStream を用いた IoT アプリケーションの構築

SINETStream を用いた IoT アプリケーション構築事例として，2.1 節のオンライン動画像解析システムと温度・湿

図 11 気温，湿度モニタリング結果のスナップショット．

度モニタリングライブデモについて紹介する．

2.1 節のオンライン動画像解析システムでは，2) センサ用プログラム，3) オンラインアプリケーションプログラム，5) ストリーミング画像出力プログラムを SINETStream を用いて開発した．2) では Writer API，3) では Reader および Writer API，5) では Reader API を実装し，メッセージング基盤ソフトウェアに対する読み書きを SINETStream を介して行うように修正した．修正後のプログラムを用いて，同様にオンライン動画像解析が行えることを確認した．

図 11 に，温度・湿度モニタリングライブデモの概要を示す．本ライブデモは，SINETStream の GitHub Pages サイトで公開されている [16]．本実験では，温度・湿度センサ付き Raspberry Pi 端末から SINETStream の Writer API を介して測定した温度・湿度データを Kafka ブローカに常時書き込む．Kafka ブローカおよび Reader 用 Python プログラムはそれぞれクラウドの VM 上で実行しており，Reader プログラムは SINETStream API を介して Kafka ブローカから温度・湿度センサの実測値を取得する．Reader プログラムでは，読み込んだ実測値の 10 分ごとの平均値を計算し，実測値と平均値をそれぞれ AWS のオブジェクトストレージ S3 に書き込む．GitHub Pages では，Javascript を用いて定期的に 5 分間，及び 24 時間の温度・湿度の実測値および平均値をそれぞれグラフ化して表示している．図 11 右下にある 2 つのグラフのうち，上が 5 分間，下が 24 時間の測定結果を表しており，24 時間のグラフでは温度・湿度変化が明確に現れている．

オンライン動画像解析システム及び温度・湿度モニタリングライブデモのいずれにおいても，SINETStream を用いることでメッセージング基盤システムの変更やセキュリティ機能の追加も設定ファイルの修正のみで容易に行えるようになっている．

5. 関連研究

Apache Kafka や MQTT ベースのブローカ等のメッセージング基盤ソフトウェアは複数開発されている．Kafka はメッセージングスループットが比較的高いものの，プロトコルオーバヘッドの点では MQTT の方が優位であり，IoT アプリケーションの要求に応じて使い分ける必要がある．SINETStream ではそのようなメッセージング基盤ソフトウェアの変更を容易に行えるようにしている．

AWS SDK[17] では，Java や Python 等の IoT デバイス SDK を提供しており，MQTT ベースのブローカに対して TLS 通信を利用可能とするなど，IoT アプリケーション開発を支援する．AWS の IoT プラットフォームの利用を前提としている点，MQTT のみを対象としている点は SINETStream と異なる．

IoT アプリケーションのための分散プラットフォーム開発も盛んに行われている．CSPOT は，IoT をターゲットとした Functions-as-a-Service（FaaS）プログラミングモデルを提供する分散ランタイムシステムを提供する [18]．WooF と呼ばれる append-only のオブジェクトストレージをセンサ，エッジデバイス，クラウドと多階層に配備した分散環境を提供し，その上で Function ベースで IoT アプリケーション開発および配備を支援する．個々の利用者の Function は，Docker コンテナとネームスペースで隔離される．SINETStream はメッセージング基盤ソフトウェアを抽象化する API および SPI を提供するため，CSPOT の WooF をメッセージング基盤のバックエンドとして利用することも可能であると考えられる．

6. おわりに

本研究では，高度な IoT アプリケーションの開発を容易にするためのソフトウェアライブラリ SINETStream を開発した．SINETStream では，IoT ストリームデータ処理のためのメッセージング API，セキュリティ機能，および多様なメッセージング基盤ソフトウェアに対応するための SPI を提供し，IoT アプリケーションの開発を支援する．

評価では SINETStream のオーバヘッドやセキュリティ機能を用いた場合の性能劣化を LAN 環境とモバイル環境で調査し，SINETStream 自体のオーバヘッドは小さいこと，暗号化のオーバヘッドは大きいものの，現状のモバイル環境では通信性能が低いため性能への影響は僅かであることがわかった．また，SINETStream を用いて IoT アプリケーションが容易に構築できることを示した．

今後は，Android 端末をターゲットとした非同期 API の開発，ブローカおよびクラウド側処理環境の構築支援，Apache Flink のようなストリーム処理系との連携を行っていく．また，第五世代移動通信システム（5G）の実用化により，これまで収集が難しかった大容量のストリームデータを扱うアプリケーションの構築も期待される．よって，他のメッセージング基盤ソフトウェアのプラグイン開発等を行い，そのようなアプリケーションの構築において

も SINETStream が有効であることを示す．

謝辞 本研究を進めるにあたり，貴重なご意見をいただいたジョージア工科大学の Calton Pu 教授，SINETStream の開発，評価にご協力いただいた数理技研の鯉江英隆様，小泉敦延様に感謝いたします．

参考文献

[1] SINET 広域データ収集基盤，https://www.sinet.ad.jp/wadci.
[2] 竹房あつ子，孫 静涛，長久 勝，吉田 浩，政谷好伸，合田憲人：SINET 広域データ収集基盤を用いたリアルタイムビデオ処理機構の検討，マルチメディア，分散，協調とモバイル (DICOMO2019) シンポジウム論文集，pp. 1581–1588 (2019).
[3] MQTT (Message Queue Telemetry Transport), http://mqtt.org/.
[4] Amazon Kinesis, https://aws.amazon.com/kinesis/.
[5] Google Cloud Pub/Sub, https://cloud.google.com/pubsub/.
[6] YOLO, https://pjreddie.com/darknet/yolo/.
[7] A PyTorch implementation of the YOLO v3 object detection algorithm, https://github.com/ayooshkathuria/pytorch-yolo-v3.
[8] Wei, S.-E., Ramakrishna, V., Kanade, T. and Sheikh, Y.: Convolutional pose machines, *CVPR* (2016).
[9] Simon, T., Joo, H., Matthews, I. and Sheikh, Y.: Hand Keypoint Detection in Single Images using Multiview Bootstrapping, *CVPR* (2017).
[10] Cao, Z., Simon, T., Wei, S.-E. and Sheikh, Y.: Realtime Multi-Person 2D Pose Estimation using Part Affinity Fields, *CVPR* (2017).
[11] Cao, Z., Hidalgo, G., Simon, T., Wei, S.-E. and Sheikh, Y.: OpenPose: realtime multi-person 2D pose estimation using Part Affinity Fields, *arXiv preprint arXiv:1812.08008* (2018).
[12] OpenPose, https://github.com/CMU-Perceptual-Computing-Lab/openpose.
[13] SINETStream, https://nii-gakunin-cloud.github.io/sinetstream/.
[14] Kafka Python client, https://github.com/dpkp/kafka-python.
[15] Eclipse Paho, https://www.eclipse.org/paho/.
[16] SINETStream Live Monitoring Demo, https://nii-gakunin-cloud.github.io/sinetstream/docs/livedemo/livedemo.html.
[17] AWS SDK, https://aws.amazon.com/jp/getting-started/tools-sdks/.
[18] Wolski, R., Krintz, C., Bakir, F., George, G. and Lin, W.-T.: CSPOT: portable, multi-scale functions-as-a-service for IoT, *Proc. the Fourth ACM/IEEE Symposium on Edge Computing (SEC '19)*, pp. 236–249 (online), DOI: 10.1145/3318216.3363314 (2019).