TTC技術レポート
Technical Report

TR-1071

IoT向けトランスポート技術の概説

Overview of Transport for IoT

第１版

2019年2月22日制定

一般社団法人
情報通信技術委員会
THE TELECOMMUNICATION TECHNOLOGY COMMITTEE

Telecommunication Technology Committee

目　次

＜参考＞

１．　国際勧告等との関連

本技術レポートに関する国際勧告は本文中に記載している。

２．　改版の履歴

版数	制定日	改版内容
第１版	2019年2月22日	制定

３．参照文章

主に、本文内に記載されたドキュメントを参照した。

４．技術レポート作成部門

第１版　：　IoTエリアネットワーク専門委員会　（SWG3604：通信インタフェースSWG）

1　はじめに

本概説（TR-1071）は、センサデータやコマンドといったショートメッセージを主に扱う IoT 向け通信技術のうち、レイヤ４（OSI 参照モデルの４層）より上の層で実現されるトランスポート（伝送）技術について記載したものである。

物理的な機器を直接接続する IoT エリアネットワークにおいては、単一のデータリンク（レイヤ２）技術でメッセージのやりとりを行う規格が開発されてきたが、IoT システムの構成によっては、広域ネットワークを通じてこれらの情報が流通するような状況も生じ、インターネット、VPN 等の TCP/IP を用いた広域網において、こうしたメッセージを円滑に交換できるしくみが必要とされている。

こうした要請に対し、元来はテキストチャットや Web 向けに開発された技術を IoT 向けにも転用し、IoT 向け通信の広域インフラとして活用しようとする流れが生じている。本 TR-1071 では以下の技術をとりあげ、概要を述べる。

・HTTP、MQTT、CoAP、QUIC
・Kafka、REST、AMQP、XMPP
・WebSocket、NATS/NATS Streaming、MapR Streams

2　プロトコルスタックとOSI7層モデル

プロトコルスタックで、TCP/IPでは第3層のネットワーク層であるIP（Internet Protocol）の上で、第4層のトランスポート層としてTCP（Transmission Control Protocol）とUDP（User Datagram Protocol）の二種類が使われている。また、TCPのバージョンにはいくつか異なったものがあるが、トランスポート層としてのTCPのみを入れ替えるだけで下位のネットワーク層も上位のアプリケーション層も変更なく利用することができる。

この層を何段にするのが適切かといった議論は1970年代から1980年代にかけて盛んに行われたが、ISOのもとで標準化されたOSI（Open Systems Interconnection）では7層に定められ、これが現在でも参照モデルとして利用されている。OSI 7層モデルを図2-1に示す。

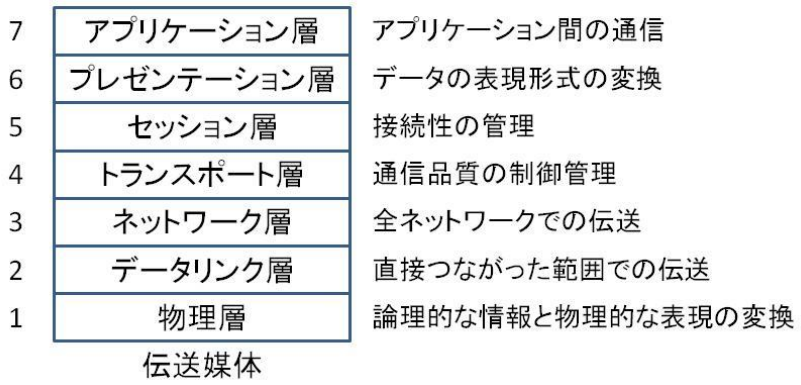

図 2-1 OSI 7 層モデル

現在のネットワークシステムの作り方においてはセッション層やプレゼンテーション層は独立した層として存在しているというよりもアプリケーションの一部として扱われていることが多く、実際には層構造として7層ではなく5層として扱うことも少なくない。このような場合でもアプリケーション層は7層と呼ばれる。また、最近のネットワークシステムではWebSocketなどのように、アプリケーションプロトコルまでは一種類ながら、そこから上にまた層を展開しているようなシステムが主流になりつつあり、7層の上に層を

展開しないと適切にモデル化できない部分も増えている。このように、現実の技術との乖離は目立つようになってはいるものの、こうした事情を取り込んだ標準的なモデルはまだ存在していないため、OSI 7層モデルが現在でも様々な場面で利用されている。

3. 各プロトコル概要

各プロトコルの特徴等の概要を、以下３つのグループに分けて表3-1～表3-3に記載する。

表 3-1 各種プロトコルの概要(HTTP、MQTT、CoAP、QUIC)

名称		HTTP	MQTT	CoAP	QUIC
特徴		ヘッダサイズの増大や通信セッションの多さが､M2M/IoT 向けでは課題視されることがある	・一方向、1 対 1 の通信のみでなく、双方向、1 対多の通信が可能でありながら、プロトコルヘッダが小さい ・バッテリーの消費を抑えたいモバイル向けの通信に適す	･センサネットワーク等の狭帯域の環境における通信に適す ・ヘッダが小さい ・ヘッダをバイナリ化し圧縮も可能､シーケンスも簡略化	･コネクションの最初に使用するロングヘッダと､その後の送受信で使用するショートヘッダがある ･TLS 同様のセキュリティ機能、接続確立の高速化 ･パケットロス検出のシグナリングを改善、 パケットロスを防ぐためのペーシング ･コネクション ID の導入により､特にモバイルクライアントの再接続ロスを低減
ソース		Hypertext Transfer Protocol HTTP/1.1：RFC 7230 HTTP/2：RFC7540	IBM 社と Eurotech 社のメンバーにより考案されたプロトコル	RFC7252 The Constrained Application Protocol (CoAP)	Google QUIC プロトコルをベース、2018 年現在 IETF で検討中 (draft-ietf-quic-http-08)
ヘッダサイズ		50 バイト以上	2 バイト	4 バイト	ロング：17バイト ショート：2～13 バイト
概要		HTML などのハイパーテキストを Web サーバーと Web クライアントの間で送受信を行うために用いられるプロトコル	Publish-Subscribe 型の軽量メッセージプロトコル｡米国 IBM が仕様策定し OASIS で標準化	・HTTP の簡易版プロトコル ・フィンランドの Sensinnode(英国 ARM 社が買収) が提案し、IETF で仕様化された HTTP に変わるプロトコル	･UDP 上にトランスポート層を作成した高速プロトコル
プロトコルスタック	トランスポート層	TCP	TCP	UDP	UDP
	ネットワーク層	IPv4、IPv6、6LoWPAN			
	データリンク層(物理)	イーサネット、無線 LAN、IEEE802.15.4 (ZigBee など)、802.15.1 (Bluetooth)、PLC など			
低消費電力化	無線通信制御	―	・1 度にたくさんの人にほぼリアルタイムでメッセージを配信 ・逆にたくさんの人から来るのは不向き ・コネクションを確立し続けるので､通信がほぼリアルタイム	･非同期通信のサポート(パケットごと Transaction ID を持つ) ･HTTP と同様の応答マッピング ・再送の処理がないため、必要に応じて上位(アプリ等) で処理要(比較的信頼性の低い通信回線では不向き)	―
	異常検知	―	･異常な切断が発生した場合､「Last Will and Testament (遺言)」機能によって､異常切断の発生をプロトコル利用者に通知	―	―

表 3-2 各種プロトコルの概要(Kafka、REST、AMQP、XMPP)

名称		Kafka	REST	AMQP	XMPP
特徴		大容量のメッセージ処理用Pull 型高スループットな分散メッセージングシステム	RESTシステムでは、多くの場合、HTML文書またはXML文書を使う。 RESTは「リソース」を扱うための考え方であり、Webページ全体のコンテンツといった、一固まりの情報を指す。	HTTPなどの手法と同じように異なるベンダ間で正しく相互運用できるような振る舞いを要求する。ただ単にAPIを定義したJMSと異なり、AMQPはワイヤレベルプロトコルである。	XMPPは電子メールのような機構の技術である。電子メールが、異なる様々な電子メールソフトで相互に交信できるように、他の数え切れないほど多数ある様々なXMPPクライアントと相互に交信できる。
ソース		Apache Kafka 2011年、LinkedInがオープンソース公開し、その後、Apache Software Foundationが開発を行った。	HTTPプロトコル規格の主要著者の一人であるRoy Fieldingが、ウェブについて論文で初めて現れ、ネットワーキングコミュニティの中ですぐに広く使われることになった。	元々は金融機関向けに開発されたようで、バンク・オブ・アメリカ、JPMorganなど金融サービス業界で利用されているようである。	Jeremie Millerにより作られたJabberが公開され、そのプロトコルは、XMPPとしてRFCとなり、公開されており、RFC 6120以下多数のRFCがある。
ヘッダサイズ		ー	ー	ー	ー
概要		分散ストリーミングプラットフォーム。「Pull型」「高スループット」などの特徴があり、ストリーミングデータパイプライン構築に利用できる。	RESTはWebサービスの設計モデルで、「リソース」を扱うための考え方。リソースはそれぞれ固有のURIを持ち、そのURIにアクセスすることで、それぞれのリソースを操作する。	ビジネスメッセージをアプリケーションや組織間で伝達するための公開規格。MQTTと同一の通信プロトコルの一種で、HTTP等と同じAP層のプロトコル。	XML ベースのプロトコルである。他のメジャーなインスタントメッセンジャーはその仕様もプロトコルも非公開となっているのが普通だが、XMPP はサーバーもクライアントもオープンソースであり、その仕様は全て公開されている
プロトコルスタック	トランスポート層	TCP	ー	ー	ー
	ネットワーク層	IPv4、IPv6、6LoWPAN			
	データリンク層(物理)	イーサネット、無線LAN、IEEE802.15.4（ZigBeeなど）、802.15.1（Bluetooth）、PLCなど			

表 3-3 各種プロトコルの概要(WebSocket、NATS/NATS Streaming、MapR Streams)

名称		WebSocket	NATS/NATS Streaming	MapR Streams
特徴		サーバーとクライアントがコネクションを行った後は、必要な通信を全てそのコネクション上で新たなコネクションを張ることがなくなり、HTTPコネクションとは異なる軽量プロトコルを使うなど通信ロスが減る。	高性能なメッセージシステムでNATSをベースにAt-least-once-deliveryなメッセージングを実現しており、Kafkaより構成がシンプルで遅延が少ない。	MapR Streamsは全体としてはApache Kafkaプログラミングモデルを使用しており、ストリームと呼ばれる新種のオブジェクトが組み込まれている。ストリームはそれぞれ多数のトピックを処理でき、ユーザーは1つのクラスタで多数のストリームを持つことができる。
ソース		W3CがAPIを、IETFがWebSocketプロトコル（RFC 6455）を担当して、ウェブサーバーとウェブブラウザとの間の通信のための規定を策定。	Derek Collisonにより開発されたもので2010年にRubyで書かれたが、その後2012年にGoに書き換えられたオープンソースでMITによりライセンスされている。	MapRは2011年にEMCコーポレーションと技術ライセンス契約を締結し、EMC独自のApache Hadoopディストリビューションのサポートを開始した。また、MapRはAmazon Web Servicesに採用されている。
ヘッダサイズ		ー	ー	ー
概要		Webにおいて双方向通信を低コストで行うための仕組みで、HTMLで書かれた文書を転送するためのプロトコルの一種。	NATSストリーミングは、NATSに基づいたデータストリーミングシステムであり、Googleが開発したGoプログラミング言語で書かれている。 NATSストリーミングサーバーは、MITライセンスの下でオープンソースソフトとして提供されている。	MapR Streamsは、リアルタイムに流れるデータ分析などに利用されるストリーミングの機能。オープンスタンダードなKAFKA APIを持っており、グローバルにレプリケーションが可能。IoTのような大規模大量データにも対応可能。
プロトコルスタック	トランスポート層	TCP	ー	ー
	ネットワーク層	IPv4、IPv6、6LoWPAN		
	データリンク層(物理)	イーサネット、無線LAN、IEEE802.15.4（ZigBeeなど）、802.15.1（Bluetooth）、PLCなど		

4 各プロトコル詳細

4.1 MQTT

4.1.1 概要

MQTTとはMessage Queuing Telemetry Transportの略語で、発信/購読型のメッセージ処理を実現するための軽量なプロトコルとして設計されており、PubSubに基づく軽量なメッセージプロトコルになっている。PubSubとはPublisher（発行者＝送信側）とSubscriber（購読者＝受信者）に分かれるメッセージングの仕組みになっているが、MQTTの場合、SubscriberがPublisherに対してメッセージの購読を希望し、Publisherが発行したメッセージをSubscriberが受け取って、処理を行う。この時、Subscriberは直接Publisherにつなぐのではなく、Broker（ブローカー＝仲介者）に接続して購読する。

すなわちPublisherはBrokerにだけメッセージを発信すればよく、各Subscriberはメッセージができたタイミングで受け取れるようになるので、よりリアルタイムになる。

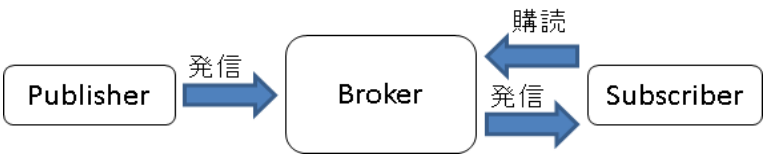

図 4-1 ブローカー以下の要素と役割

4.1.2　特徴

送信側が送るデータをデータ領域に一旦保持しながら、受信側の処理が完了するのを待たずに次の処理へ移る方式で、遠隔にあるセンサーやデバイスなどが収集したデータを、受信側の処理状況を気にせず効率よく送信することができるプロトコルである。

MQTTは、双方向、1対多の通信が可能でありながら、ヘッダサイズは「HTTP：50バイト～、MQTT：2バイト～」となっており、バッテリーの消費を抑えたいモバイル向けの通信に適している。その他の特徴を以下に示す。

・QoS：通信環境が不安定な場合など、レベルに応じた通信を可能にする
・非同期のパブリッシュ/サブスクライブ型通信
・クライアント（デバイス）障害時などに、“遺言”を送信することが可能

4.1.3　用途

「MQTT」が、M2M（Machine-to-Machine）や、IoT（Internet of Things）を実現するのに適したプロトコルと言われるゆえんは、シンプルで軽量なところにあり、多数のデバイスの間で、短いメッセージを頻繁に送受信することを想定して作られており、具体例を下図に示す。

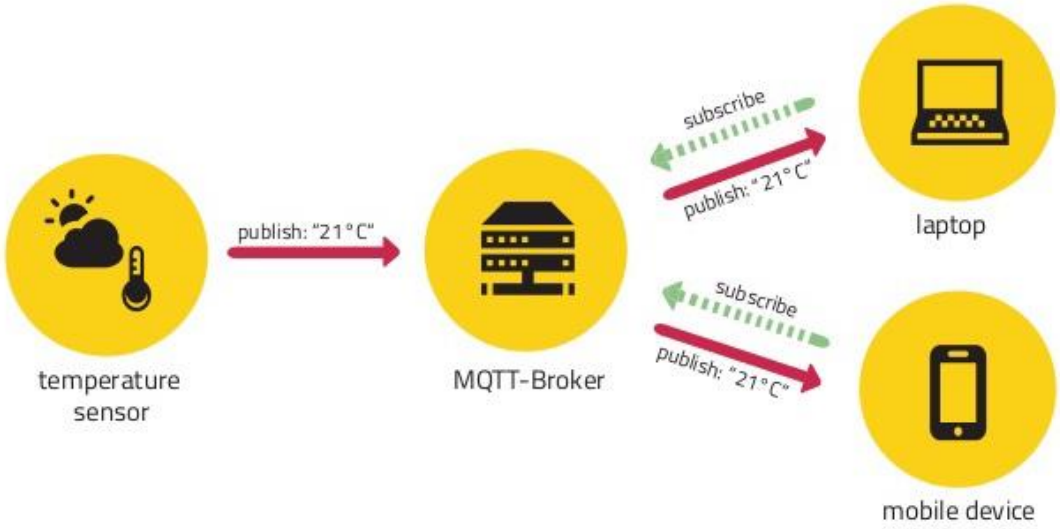

図 4-2　用途具体例

出典：ブログ「IoT時代のプログラミング（主にMQTTについて）」より引用
（https://qiita.com/darai0512/items/37158f56e9a6b4ce83ed）

4.2　CoAP

4.2.1　概要

CoAP（Constrained Application Protocol）は、Webサービスのためにオープンソースで開発された簡易的なHTTPで、IETF7252で規定されており、非同期通信のサポートやパケットヘッダの簡略化などの特徴を持ち、M2M用のプロトコルとして期待されている。

4.2.2　特徴

CoAPの特徴を下記に示す。

・UDPを使っている（TCPと比較して処理が少ない）
・ヘッダが小さい

しかし、UDPだからこそ課題もある。すなわちCoAPはUDPを使用していることにより、信頼性の部分でTCPに劣り、TCPのように再送処理などが出来ないので、これは上位（アプリ等）で対応する必要がある。

4.2.3　用途

CoAPは制約の有るノードや制約のあるネットワーク（例：低消費電力、パケット損失が多い）で使用するためのWeb転送プロトコルである。RFC7252によると、ノードは低容量のRAM and ROMで8bitのマイコンであることが多く、6LoWPANsでは高いパケットエラー率でありスループットは数10kbit/s程度である。このプロトコルはスマートエネルギーやビルオートメーション用としてM2Mアプリケーション向けに設計されている。

4.3　QUIC

4.3.1　概要

QUICは、Googleが開発しているトランスポートプロトコルで、Googleの「SPDY」プロトコルをベースにスタートし、Google QUICプロトコルを基に、2018年現在、IETFで「draft-ietf-quic-http-08」として検討中である。TCPではなくUDPをベースとして開発された、Web向けのプロトコルで、TCP接続を開始するための遅延を少なくすることを目的としている。

4.3.2　特徴

UDP上で動作するプロトコルであり、TCPのような信頼性を持ちTLSのような暗号化された通信路で効率的なHTTPのメッセージをやりとりする。すでにChromeやGoogleのサービスで実装されている。Webの配信高速化のために、HTTP/1.1からHTTP/2へ改定され、より高速化が図られている。

QUICの概要を以下の図に示す。

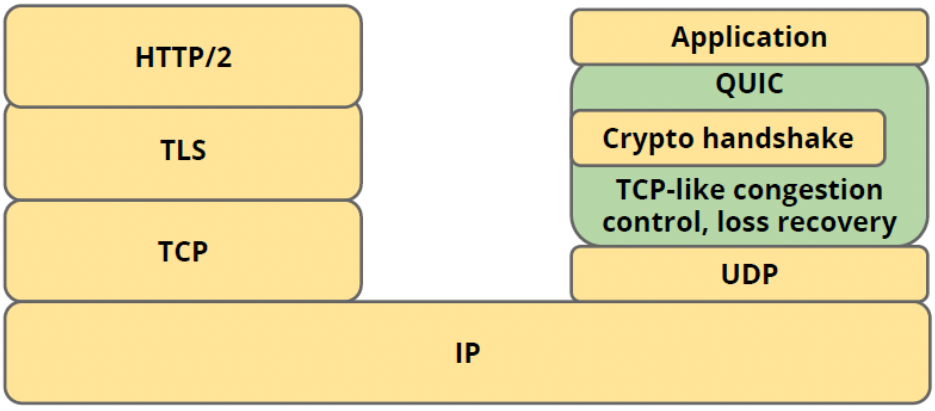

図 4-3　QUIC 概要（プロトコルスタック）

出典：IETF資料「QUIC A New Internet Transport」より引用

（https://www.ietf.org/proceedings/96/slides/slides-96-quic-5.pdf）

左側のスタックが一般的に使用されているHTTP/2で、右側がQUICのスタックである。

QUICの構成としては

- ◆　QUIC で提供されるトランスポート上で HTTP のメッセージをやりとりする。
- ◆　コネクション確立時に TLS1.3 のメッセージをやり取りし、以後のメッセージを暗号化する鍵を取得する。

◆　UDP 上で、TCP のような信頼性(パケロス回復・パケットの順番の補正)や輻輳制御フロー制御を提供。

QUICのメリットとしては以下の項目がある。

◆　UDP 上で通信するので TCP のように通信環境（IP アドレス）が変わってもセッションが切れない（UDP 上でセッション ID が管理される）

◆　TCP の場合パケットロスが発生した場合、そのパケットを回復してからアプリケーションとして処理可能になるが、UDP なので届いたパケットから処理することが出来る

◆　ACK といった制御情報を含むほとんどの領域を暗号化し、データの保護

◆　TCP だと改善しようとするとカーネルを更新する必要があり時間が掛かるが、ユーザランドで動作するため更新が容易

4.3.3　用途

WEBアプリケーションの最適化・高速化、低遅延化など

4.4　Kafka

4.4.1　概要

Kafka（カフカ）とは、分散ストリーミングプラットフォームで、「Pull型」「高スループット」などの特徴があり、ストリーミングデータパイプライン構築に利用できる。また、分散環境において「高スループット」かつ「低遅延」で、大規模データ（ログデータ/イベントデータなど）を高速に取り込み、配信できるメッセージングシステムである。オンライン/オフライン両方のメッセージ取得に対応する。

「リアルタイムストリーミングデータパイプライン構築」や「データストリームを変換するリアルタイムストリーミングアプリケーション」などを構築できる。

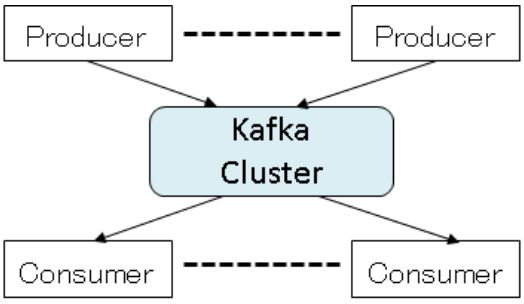

図 4-4 Apache Kafka

4.4.2　特徴

特徴を以下に示す。

◆Pull型（Publish-Subscribeモデル）

Pull型（Publish-Subscribeモデル）を採用しており、データ転送量などを意識する必要がなく、自らスループット調整でき、バッチ処理にも対応できるメリットがある。

◆高速大量処理

大量のメッセージを高速処理でき、カーネルメモリキャッシュを最大限使用するなどの仕組みにより高速処理を実現している。

◆高可用性

分散環境運用コーディネーションエンジンであるApache ZooKeeperを動かして、複数のKafkaサーバーを連携させてクラスタ化し、耐障害性/高可用性を実現する。

◆スケーラブル

Kafkaクラスタをダウンタイムなしで、柔軟かつ透過的に拡張できる。

◆ストリームデータ格納構成

1つ以上のサーバー上でクラスタとして実行され、Kafkaクラスタは「トピック」と呼ばれるカテゴリにストリームレコードを格納する。

4.4.3 用途

Kafkaのユースケースを以下に示す。

◆メッセージブローカー

大規模なメッセージ処理アプリケーションに適しており、メッセージブローカーとして機能し、同様機能の「ActiveMQ」「RabbitMQ」などのメッセージングシステムに匹敵する。

◆Webサイトのアクティビティトラッキング

1つのユーザーアクティビティが1つのトピックとなり、トピックに対してサイトアクティビティが公開され、さまざまなユースケースのサブスクリプションに利用できる。

◆メトリック

運用監視として、分散アプリケーションからの統計を集約して、運用データを作成処理に利用できる。

◆ログ集計

ファイルとして、ログやイベントデータをメッセージストリームとして抽象化する機能を持つ。

◆ストリーム処理

リアルタイムでイベントに反応するストリーム処理アプリケーションを作成でき、パイプライン処理を実現できる。

4.5 REST

4.5.1 概要

REST（Representational State Transfer）はHTTP仕様に準拠したWebサービスで、Webサービスの設計モデル、「リソース」を扱うための考え方である。リソースはそれぞれ固有のURIを持ち、そのURIにアクセスすることで、それぞれのリソースを操作する。その際の操作はHTTPのメソッドを「正しく」使うことで行う。つまりHTTPの4つのメソッド、「GET」「POST」「PUT」「DELETE」で何を行うかを伝えるものである。

WikipediaのREST説明によると、初めはアーキテクチャの原則と制約の集まりを指していたが、次第に、XMLやHTTPを使った簡易なウェブベースのインタフェースのうち、WebサービスのSOAPプロトコルのようなMEP（Message Exchange Pattern;）ベースの特別な抽象化をしないもののことを、大まかに意味する用語として使われるようになった、とされている。

4.5.2 特徴

RESTは設計に際し、以下を設計原則とするよう提言されている。

- アドレス指定可能な URI で公開されている
- HTTP メソッドの利用の統一がされている（URI で表されたリソースに対して各 HTTP メソッドで操作を行う）
- ステートレスである（サーバーはクライアントのセッション情報を保持しない）
- 処理結果が HTTP ステータスコード（Web サーバーからのレスポンスを表すコード）で通知される

これらの原則に則ったWebサービスをRESTfulなサービスと言い、例えばステートレスでは下記のようなメリットがある。

- クライアントのリクエストはリソース操作に必要十分な情報になる
- セッション管理がシンプルになる
- スケーラブルなシステムにできる

4.6 AMQP

4.6.1 概要

Advanced Message Queuing Protocol（AMQP） は ビジネスメッセージをアプリケーションや組織間で伝達するための公開規格のアプリケーション層プロトコルである。

AMQPの機能は、メッセージ指向、キューイング、ルーティング、信頼性、セキュリティで定義づけられており、元々は金融機関向けに開発されたもので、バンク・オブ・アメリカ、JPMorgan等金融サービス業界で利用されていた。

4.6.2 特徴

AMQPはメッセージングプロバイダ（サーバー）とクライアントに、HTTPなどの手法と同じように異なるベンダ間で正しく相互運用できるような振る舞いを要求し、異なる組織・異なるプラットフォーム・異なる時間（非同期処理）・異なる場所（離れている場所やネットワーク環境）でも、繋げることができる点に強みがある。

ただ単にAPIを定義した JMS（Java Message Service）と異なり、AMQPはネットワークを通して、オクテットストリームとして送信されるデータフォーマットの記述するワイヤレベルプロトコルで、実装言語に非依存でプロトコルに従ったツール間で相互運用されているフォーマットのメッセージであれば、どんなツールでも生成・翻訳が可能で、以下の特徴がある。

- **効率性：** AMQP は、プロトコル命令とそれを介して転送されるビジネスメッセージにバイナリエンコードを使用する接続指向のプロトコルである。
- **信頼性：** AMQP プロトコルを使用すると、ファイア アンド フォーゲット配信から、厳密に 1 回限り承認される信頼性の高い配信まで、信頼性を幅広く保証しながらメッセージを交換できる。
- **柔軟性：** AMQP は、さまざまなトポロジのサポートに使用できる柔軟なプロトコルである。
- **ブローカー モデルに依存しない：** AMQP の仕様には、ブローカーによって使用されるメッセージング モデルの要件がない。

下記に、標準の JMS API を使用して記述された Linux で実行される Java クライアントと、Windows で実行される .NET クライアントが、AMQP 1.0 を使用して Service Bus 経由でメッセージを交換するサンプルを示す。

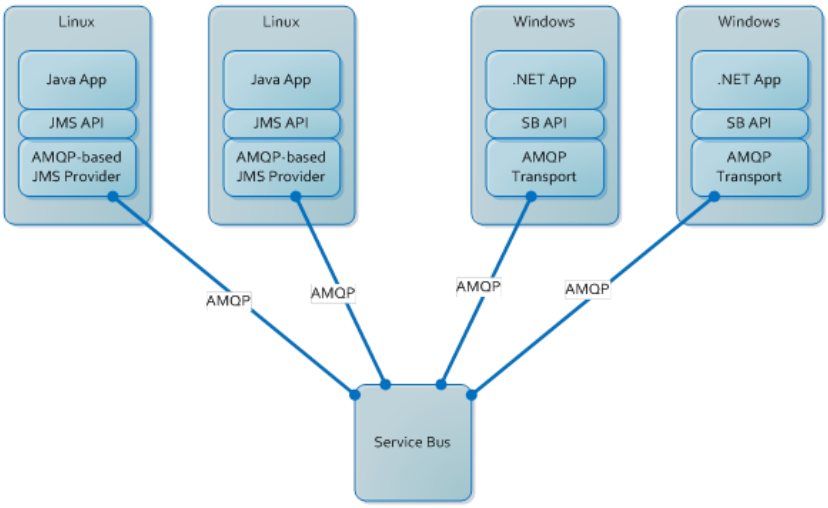

図 4-5　Service Bus と AMQP 1.0 を使用したクロスプラットフォーム メッセージングを示すサンプルのデプロイメント シナリオ

出典：Microsoft Azure 「Service Bus での AMQP 1.0 サポート」より引用
（https://docs.microsoft.com/ja-jp/azure/service-bus-messaging/service-bus-amqp-overview）

4.7　XMPP

4.7.1　概要

XMPP（Extensible Messaging and Presence Protocol）は、XML（Extensible Markup Language）ベースの通信プロトコルである。

XMPPは従来のインスタントメッセンジャー（IM）とは異なり、オープンソースであり、仕様は公開されており、誰でも自分専用のXMPPサーバーを立ち上げることができる。

XMPPは電子メールのような機構の技術で、電子メールが、異なる様々な電子メールソフトで相互に交信できるように、多数ある様々なXMPPクライアントと相互に交信できる。

4.7.2　特徴

Jabber社が開発したインスタントメッセージソフト「Jabber」のプロトコルを、セキュリティ機能などを追加して改良したものであり、XMPPは柔軟性や拡張性が高いのが特徴でインスタントメッセージソフトで必要不可欠となるメッセージの送受信や、プレゼンス状態の通知などを受け持っている。

メッセージ転送の仕組みの例を下記に示す。

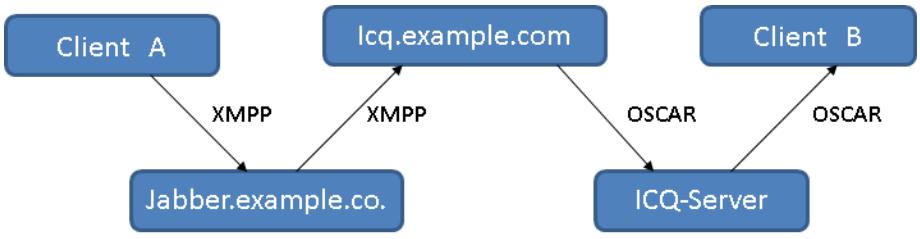

図 4-6　メッセージ転送の仕組みの例

・Aは XMPP ネットワークを通して ICQ トランスポートにメッセージを送る。

・次にメッセージは ICQ ネットワークを通じて Bへ送られる。

と言う感じである。

XMPP の他の便利な特徴はトランスポートで、他のプロトコルを使うネットワークにアクセスすることが可能で、インスタントメッセージングのプロトコルだけでなく、SMS や電子メールなどのプロトコルでも可能である。

4.8 WebSocket

4.8.1 概要

WebSocketは、コンピュータネットワークの双方向通信用の技術規格で、W3CがAPIを、また、IETFがWebSocketプロトコルRFC 6455を担当して、ウェブサーバーとウェブブラウザとの間の通信のための規定を策定している。なお、WebSocketはXMLHttpRequest（XHR）の欠点を解決するための技術として開発されており、現在のComet（コメット：サーバー上の新規データをクライアントからの要請無しに取得する技術）に代わる技術を目的としている。

4.8.2 特徴

WebSocketの特徴を下記に示す。

◆双方向通信

WebサーバーとWebブラウザの双方間にTCPコネクションを構築することにより、Webで双方向通信を可能にしている。

◆HTTPに比べて小さなデータサイズで必要な情報を送信

HTTPに比べて小さなデータサイズで必要な情報を送信できるため、小さいメッセージを頻繁にやり取りする場合は有利になる。

4.9 NATS/NATS Streaming

4.9.1 概要

NATSは軽量でハイパフォーマンスなメッセージングシステムを提供するミドルウェアで、いわゆるNATSと呼ばれるものとNATS Streamingと呼ばれる大きく2つの仕組みがある。

4.9.2 特徴

◆NATSの特徴

- シンプルなテキストベースのプロトコル
- 遅いクライアントを自動で切断

◆NATS Streamingの特徴

NATSストリーミングは、NATSに基づいたデータストリーミングシステムであり、Goプログラミング言語で書かれている。NATSストリーミングサーバーの実行可能ファイル名は、nats-streaming-serverである。 NATSストリーミングは、コアNATSプラットフォームとシームレスに組み込み、拡張、相互運用でき、MITライセンスの下でオープンソースソフトウェアとして提供されている。

NATSをベースにAt-least-once-deliveryなメッセージングを実現しており、単純なPub/Subの場合Subクライアントは接続した時点からメッセージを取得できるが、NATS Streamingの場合、インメモリでメッセージを保持し、クライアントに必ず最低1回はメッセージを配信できる。

・ At-least-once はメッセージ配信機構

・ Publisher/Subscriber Rate Limit NATS ストリーミングを実現するサーバーが NATS ストリーミングサーバーで、NATS サーバーをベースに実装されている。内部では Protocol Buffers も使用。

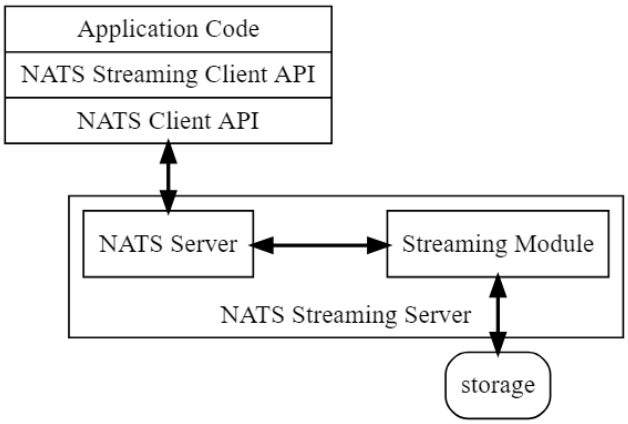

図 4-7　NATS Streaming Server の仕組み

NATS WEB Site「NATS Streaming Concepts」より引用

（https://nats.io/documentation/streaming/nats-streaming-intro/）

4.10　MapR Streams

4.10.1　概要

MapR Streamsは、大容量イベントデータのストリーミング向けメッセージングシステムで、MapRのコンバージドプラットフォームに組み込んで使用する。MapR Streamsは全体としてはApache Kafkaプログラミングモデルを使用しているものの、いくつか重要な相違点がある。例えば、MapRファイルシステムには、「ストリーム」と呼ばれる新種のオブジェクトが組み込まれている。ストリームはそれぞれ多数のトピックを処理でき、ユーザーは1つのクラスタで多数のストリームを持つことができる。TTL（有効期間）、ACE（アクセス制御式）などのポリシーは、ストリームレベルで設定でき、多数のトピックの同時管理に便利である。

4.10.2　特徴

MapR Streamsの特徴を下記に示す。

◆ スケーラブルで高スループットの継続稼働するストリームを、数千もの場所をまたいで容易に構築できるようになる。MapR Streamsは、何百万ものトピックスや何十億ものメッセージを処理可能。

◆ 分析、トランザクション、ストリーム処理を1つに統合し、レイテンシやデータの重複を削減して、クラスタの散在を解消する。同時に、Spark Streamingや Apache Storm、Apache Flink、Apache Apexといった既存のオープンソースプロジェクトの利用も可能。

◆ 自動フェイルオーバーおよび順序一貫性の確保により、安定したメッセージ配信が可能。

◆ サイトを跨ぐレプリケーションによりグローバル規模のリアルタイムアプリケーションの構築可能。

◆ ストリーム内のすべてのメッセージを容量無制限に保存可能。