IIR

Internet
Infrastructure
Review

Sep.2021
Vol.52

定期観測レポート

ブロードバンドトラフィックレポート ～2年目に入ったコロナ禍の影響～

フォーカス・リサーチ(1)

Verifiable CredentialとBBS+署名

フォーカス・リサーチ(2)

HaskellによるQUICの実装

IIJ
Internet Initiative Japan

Internet Infrastructure Review

September 2021 Vol.52

エグゼクティブサマリ

2021年9月、日本政府の新たな機関としてデジタル庁が設立されました。デジタル庁のウェブサイト[*1]には「デジタル時代の官民のインフラを今後5年で一気呵成に作り上げ、全ての国民にデジタル化の恩恵が行き渡る社会を実現する」と書かれています。組織体制図からも、国民向けサービスと省庁業務サービスのデジタル化に積極的に取り組もうとしていることが読み取れます。

行政サービスを受ける機会がないとあまり気にすることもありませんが、数年前に転居したときには各種手続きの多さや不便さに辟易することがありました。また、現在進行形の新型コロナウイルス禍においては、行政サービスでICTの有効活用が進んでいれば、各種給付金の支給やワクチン接種がよりスムーズに行えたのではないかという批判をマスコミ報道で目にします。

日本の行政におけるICTの利活用については、2020年7月に公表された国際連合経済社会局による「世界電子政府ランキング[*2]」において、国際連合加盟193ヵ国中第14位、同年9月に公表された早稲田大学電子政府・自治体研究所による「世界デジタル政府ランキング年次調査[*3]」では、世界のICT先進国64ヵ国中第7位となっています。評価方法にもよるとは思いますが、日本政府のデジタル化は海外各国と比較すると突出して先を行くわけではないものの、世間で批判されているほど遅れているわけではないということになるのでしょうか。

とはいえ、社会全体でICTの利活用を進め、デジタルトランスフォーメーションを推進することは、国民生活の向上にとって重要です。その際、インターネットは必要不可欠なインフラであり、IIJはインターネットを支えることを通じて、社会のデジタルトランスフォーメーションの実現に貢献していきたいと考えています。

「IIR」は、IIJで研究・開発している幅広い技術を紹介しており、日々のサービス運用から得られる各種データをまとめた「定期観測レポート」と、特定テーマを掘り下げた「フォーカス・リサーチ」から構成されます。

1章の定期観測レポートは、IIJの固定ブロードバンドとモバイルのトラフィックに関する定期的な分析です。インターネットのトラフィックがコロナ禍によって大きく変化して2年目になります。社会的な行動制限によるトラフィックへの影響、固定ブロードバンドのPPPoEからIPoEへのシフト、HTTPからHTTPSへのシフト、更には3章で紹介しているQUICの増加など、社会の動きや技術の変化がトラフィックの変化にも現れていることがよく分かる結果となっています。

2章のフォーカス・リサーチ(1)では、Self-Sovereign Identity(SSI)の中核となるVerifiable Credential(VC)とそれを実現するBBS+署名について解説しています。今後、デジタルトランスフォーメーションが進むにつれ、デジタルアイデンティティを利用者自身が主体的に管理できるSSIはますます重要になると考えられます。SSIについては、本レポートのVol.43(https://www.iij.ad.jp/dev/report/iir/043.html)でも紹介していますが、その後の2年間でSSIを実現するための技術開発が進んでいます。従来のデジタル証明書とVCの違い、VCに関する国内外の実装、標準化、今後の課題などにも触れています。

3章のフォーカス・リサーチ(2)は、先般RFC9000で標準化されたQUICをHaskellで実装した取り組みの紹介です。筆者は新しいプロトコルの議論に参加すると共に、実際に実装し、他の実装との相互接続性を確認することにより、使用時の完成度の向上に尽力しています。多くの実装がイベント駆動プログラミングを採用するなか、ここではHaskellの特徴を生かしたスレッドプログラミングを採用することにより、他の実装とは違う視点から仕様の検証ができたと考えています。本章で紹介する具体的な実装のポイントは、皆様のQUICに対するより深い理解の一助になるかと思います。

IIJは、このような活動を通してインターネットの安定性を維持しながら、日々、改善・発展させていく努力を行っています。今後も企業活動のインフラとして最大限に活用いただけるよう、様々なサービスやソリューションを提供し続けてまいります。

島上 純一(しまがみ じゅんいち)
IIJ 取締役 CTO。インターネットに魅かれて、1996年9月にIIJ入社。IIJが主導したアジア域内ネットワークA-BoneやIIJのバックボーンネットワークの設計、構築に従事した後、IIJのネットワークサービスを統括。2015年よりCTOとしてネットワーク、クラウド、セキュリティなど技術全般を統括。2017年4月にテレコムサービス協会MVNO委員会の委員長に就任。

＊1　デジタル庁、「デジタル庁について」(https://www.digital.go.jp/about-us)。
＊2　国際連合経済社会局、"UN E-Government Survey 2020"(https://publicadministration.un.org/en/Research/UN-e-Government-Surveys)。
＊3　早稲田大学電子政府・自治体研究所、「世界デジタル政府ランキング発表」(https://idg-waseda.jp/ranking_jp.htm)。

ブロードバンドトラフィックレポート ～2年目に入ったコロナ禍の影響～

1.1 概要

このレポートでは、毎年IIJが運用しているブロードバンド接続サービスのトラフィックを分析して、その結果を報告しています[*1*2*3*4]。今回も、利用者の1日のトラフィック量やポート別使用量などを基に、この1年間のトラフィック傾向の変化を報告します。

昨年に引き続き、コロナ禍で自宅でのインターネット利用が増え、ブロードバンドトラフィックも増加傾向が続いています。その一方で、外出が減った分モバイルの利用量はほぼ横ばいとなっています。

図-1は、IIJの固定ブロードバンドサービス及びモバイルサービス全体について、月ごとの平均トラフィック量の推移を示したグラフです。トラフィックのIN/OUTはISPから見た方向を表し、INは利用者からのアップロード、OUTは利用者へのダウンロードとなります。トラフィック量の数値は開示できないため、両サービスの2020年1月のOUTの値を1として正規化しています。

ブロードバンドサービスのトラフィックは、新型コロナウイルスの国内感染が本格的に広がり始めた昨年3月から5月にかけて急増し、緊急事態宣言解除後の6月には少し減りましたが、8月から再び増加に転じました。この1年のブロードバンドトラフィック量は、INは20%の増加、OUTは23%の増加となっています。1年前はそれぞれ43%と34%の増加でしたので、昨年に比べ増加率は減少したものの、それ以前の増加率に戻ったとも捉えることができます。逆に、モバイルサービスは、リモートワーク向けの利用が増えたものの、外出時の利用分が減ったため、全体としては横ばい傾向です。モバイルは、この1年で、INは39%の増加、OUTは1%の減少となっています。1年前はINが28%の増加、OUTが7%の減少でした。

ブロードバンドに関しては、IPv6 IPoEのトラフィック量も含めて示しています。IIJのブロードバンドにおけるIPv6は、IPoE方式とPPPoE方式があります[*5]。2021年6月時点で、IPoEのブロードバンドトラフィック量の全体に占める割合は、INで31%、OUTで30%と、昨年同月よりそれぞれ7ポイントと10ポイント増えていて、全体の3分の1に近いトラフィックがIPoE

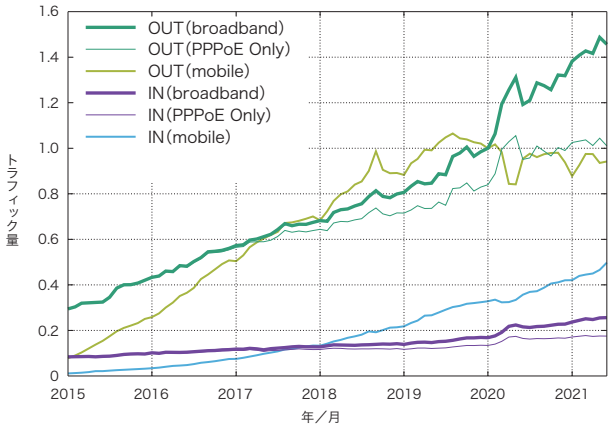

図-1 ブロードバンド及びモバイルの月間トラフィック量の推移

＊1　長健二朗. ブロードバンドトラフィックレポート：新型コロナウイルス感染拡大の影響. Internet Infrastructure Review. Vol.48. pp4-9. September 2020.
＊2　長健二朗. ブロードバンドトラフィックレポート：トラフィック量は緩やかな伸びが継続. Internet Infrastructure Review. Vol.44. pp4-9. September 2019.
＊3　長健二朗. ブロードバンドトラフィックレポート：ダウンロードの増加率は2 年連続で減少. Internet Infrastructure Review. Vol.40. pp4-9. September 2018.
＊4　長健二朗. ブロードバンドトラフィックレポート：トラフィック増加はややペースダウン. Internet Infrastructure Review. Vol.36. pp4-9. August 2017.
＊5　小川晃通, 久保田聡. 徹底解説v6 プラス. ラムダノート. January 2020.（https://www.jpne.co.jp/books/v6plus/）。

となっています。コロナ禍で顕著になってきたPPPoEの輻輳を避けて、IPoEへ移行する利用者が増えていて、IPoEの利用拡大が続いています。

次に、コロナ禍の平日と週末の時間別ブロードバンドトラフィック量の推移を見ていきます。ここでのトラフィック量はPPPoEとIPoEの合計値です。図-2及び図-3に以下の7つの週のトラフィックを示します。一斉休校が始まる前の2020年2月25日の週、最初の緊急事態宣言下の4月20日の週、緊急事態解除後の6月22日の週、感染第2波が落ち着いてきた8月31日の週、2回目の緊急事態宣言下の2021年1月18日の週、首都圏の緊急事態解除後の3月22日の週、そして、変異ウイルスによる感染第5波の始まりの7月5日の週です。ここでは、週の各時間の平均トラフィック量を、月曜から金曜の平日（休日は除く）と土日の週末に分けています。下側の波線はそれぞれの週のアップロード量ですが、今回はダウンロード量に注目します。

まず、平日のトラフィック量の推移を見ます。昨年の2月と4月を比較して最初の緊急事態宣言の影響を見ると、昼間のトラフィックが大きく増えていて、また夜のピーク時間帯でも増加しています。緊急事態宣言が解除された6月には昼間の増加分が半分以下に減少していますが、ピーク時間帯ではほとんど減っていません。その後、昼間のトラフィックは徐々に増えていきますが昨年4月のレベルに届くのは今年の3月です。この週は学校が春休みで昼間のトラフィック量は少し多めになっています。7月には昼間のトラフィックが少し減っていますが、これは、学校が授業期間であることに加え、リモートワークも少し減っているようです。また、20時から22時のピーク時間帯に着目するとほぼ一貫して増加していることが分かります。

一方、週末のトラフィック量は、平日に比べ変化が少なくなっています。週末の昼間の在宅率は学校やリモートワークよりも天気が大きく影響します。例えば、今年の1月23日・24日は悪天候で自宅でのインターネット利用が増え、3月27日・28日は関東以西で桜が満開となり人出が増えた結果トラフィックは少なめです。夜のピーク時間帯のトラフィック量については平日とほぼ同じです。

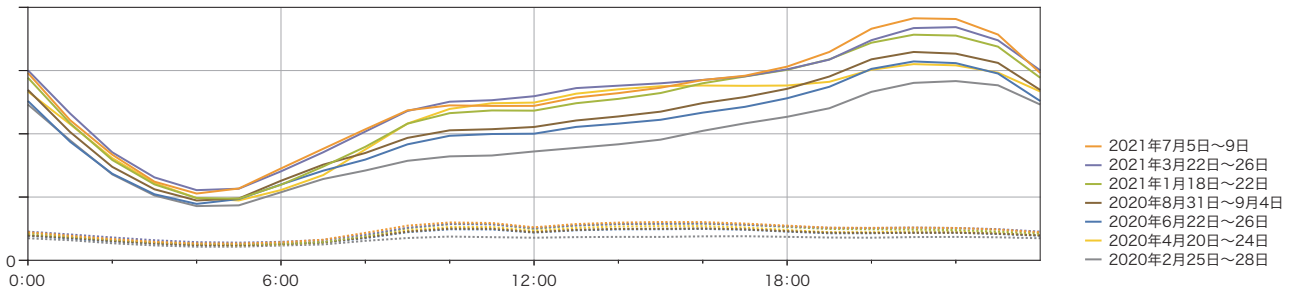

図-2 平日の時間別ブロードバンドトラフィック量の推移

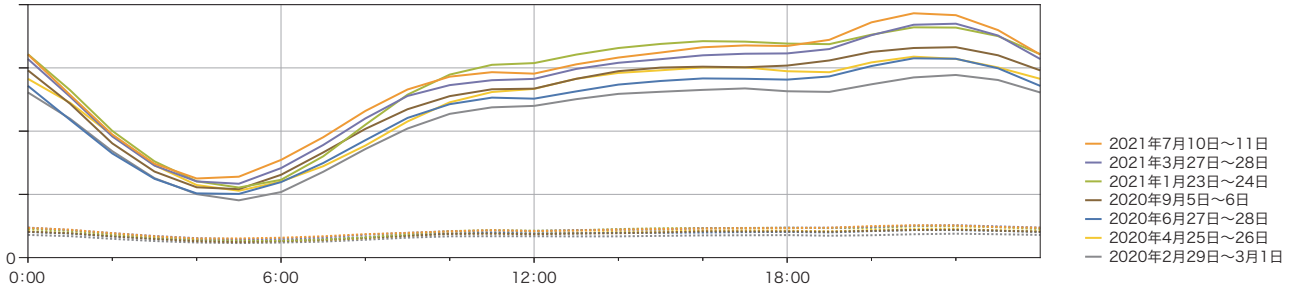

図-3 週末の時間別ブロードバンドトラフィック量の推移

なお、IPoEトラフィックはインターネットマルチフィード社のtransixサービスを利用していて詳細なデータが取得できていないため、以降の解析の対象にはなっていません。

1.2 データについて

今回も前回までと同様に、ブロードバンドに関しては、個人及び法人向けのブロードバンド接続サービスについて、ファイバーとDSLによるブロードバンド顧客を収容するルータで、Sampled NetFlowにより収集した調査データを利用しています。モバイルに関しては、個人及び法人向けのモバイルサービスについて、使用量にはアクセスゲートウェイの課金用情報を、使用ポートにはサービス収容ルータでのSampled NetFlowデータを利用しています。

トラフィックは平日と休日で傾向が異なるため、1週間分のトラフィックを解析しています。今回は、2021年5月31日から6月6日の1週間分のデータを使っていて、前回解析した2020年6月1日から6月7日の1週間分と比較します。

ブロードバンドの集計は契約ごとに行い、一方モバイルでは複数電話番号の契約があるので電話番号ごとの集計となっています。ブロードバンド各利用者の使用量は、利用者に割り当てられたIPアドレスと、観測されたIPアドレスを照合して求めています。また、NetFlowではパケットをサンプリングして統計情報を取得しています。サンプリングレートは、ルータの性能や負荷を考慮して、1/8192程度に設定されています。観測された使用量に、サンプリングレートの逆数を掛けることで全体の使用量を推定しています。

IIJの提供するブロードバンドサービスにはファイバー接続とDSL接続がありますが、今ではファイバー接続の利用がほとんどとなっています。2021年には観測されたユーザ数の99%はファイバー利用者でした。

1.3 利用者の1日の使用量

まずは、ブロードバンド及びモバイル利用者の1日の利用量をいくつかの切り口から見ていきます。ここでの1日の利用量は各利用者の1週間分のデータの1日平均です。

2019年のレポートから、利用者の1日の使用量は個人向けサービス利用者のデータのみを使っています。これは、利用形態が多様な法人向けサービスを含めると分布の歪みが大きくなってしまうため、全体の利用傾向を掴むには個人向けサービス分だけを対象にした方が、より一般性がありかつ分かりやすいと判断したからです。なお、次節のポート別使用量の解析では区別が難しいため法人向けも含めたデータを使っています。

図-4及び図-5は、ブロードバンドとモバイル利用者の1日の平均利用量の分布（確率密度関数）を示します。アップロード（IN）とダウンロード（OUT）に分け、利用者のトラフィック量

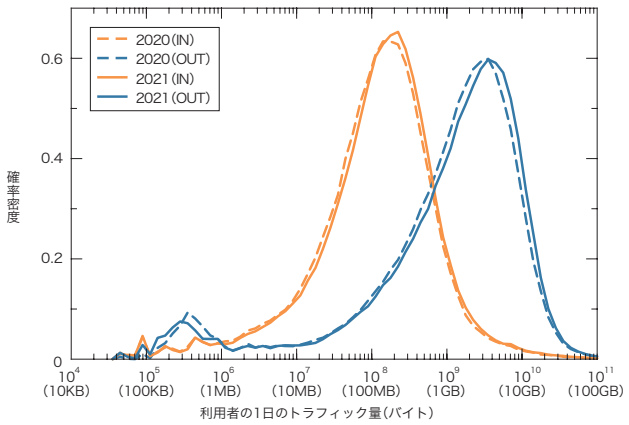

図-4 ブロードバンド利用者の1日のトラフィック量分布
2020年と2021年の比較

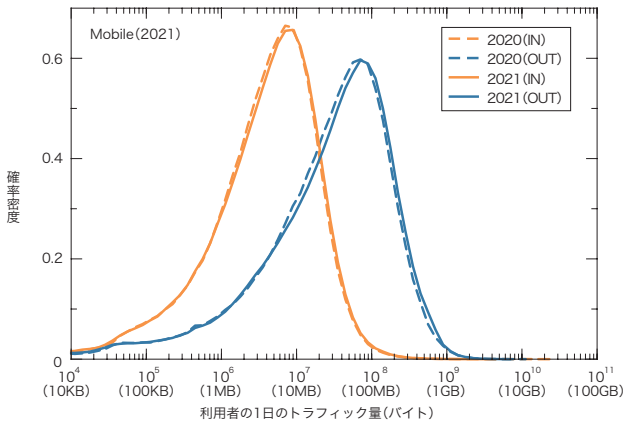

図-5 モバイル利用者の1日のトラフィック量分布
2020年と2021年の比較

をX軸に、その出現確率をY軸に示していて、2020年と2021年を比較しています。X軸はログスケールで、10KB（10^4）から100GB（10^{11}）の範囲を示しています。一部の利用者はグラフの範囲外にありますが、概ね100GB（10^{11}）までの範囲に分布しています。

ブロードバンドのINとOUTの各分布は、片対数グラフ上で正規分布となる、対数正規分布に近い形をしています。これはリニアなグラフで見ると、左端近くにピークがあり右へなだらかに減少する、いわゆるロングテールな分布です。OUTの分布はINの分布より右にずれていて、ダウンロード量がアップロード量より、ひと桁以上大きくなっています。2020年と2021年で比較すると、INとOUT共に分布の山が僅かながら右に移動しており、利用者全体のトラフィック量が増えていることが分かります。今回はこれまでに比べて分布にほとんど変化がありませんが、これはPPPoEの総量が伸びていないことからも窺えます。

右側のOUTの分布を見ると、分布のピークはここ数年間で着実に右に移動していますが、右端のヘビーユーザの使用量はあまり増えておらず、分布の対称性が崩れてきています。一方で、左側のINの分布は左右対称で、より対数正規分布に近い形です。

図-5のモバイルの場合も同様に、分布の山が僅かに右に移動していて、全体の利用量が僅かながら増えていることが分かります。モバイルの利用量は、ブロードバンドに比べて大幅に少なく、また、使用量に制限があるため、分布右側のヘビーユーザの割合が少なく、左右非対称な形になります。極端なヘビーユーザも存在しません。外出時のみの利用や、使用量の制限のため、各利用者の日ごとの利用量のばらつきはブロードバンドより大きくなります。そのため、1週間分のデータから1日平均を求めると、1日単位で見た場合より利用者間のばらつきは小さくなります。1日単位で同様の分布を描くと、分布の山が少し低くなり、その分両側の裾が持ち上がりますが、基本的な分布の形や最頻出値はほとんど変わりません。

表-1は、ブロードバンド利用者の1日のトラフィック量の平均値と中間値、分布の山の頂点にある最頻出値の推移を示します。分布の山に対して頂点が少しずれている場合は、最頻出値は分布の山の中央に来るように補正しています。今回はいずれの数値も伸びています。分布の最頻出値を2020年と2021年で比較すると、INでは158MBから200MBに、OUTでは3162MBから3981MBに増えており、伸び率で見ると、INで1.3倍、OUTでも1.3倍となっています。

	IN（MB/day）			OUT（MB/day）		
年	平均値	中間値	最頻出値	平均値	中間値	最頻出値
2007	436	5	5	718	59	56
2008	490	6	6	807	75	79
2009	561	6	6	973	91	100
2010	442	7	7	878	111	126
2011	398	9	9	931	144	200
2012	364	11	13	945	176	251
2013	320	13	16	928	208	355
2014	348	21	28	1124	311	501
2015	351	32	45	1399	443	708
2016	361	48	63	1808	726	1000
2017	391	63	79	2285	900	1259
2018	428	66	79	2664	1083	1585
2019	479	75	89	2986	1187	1995
2020	609	122	158	3810	1638	3162
2021	684	136	200	4225	1875	3981

表-1 ブロードバンド個人利用者の1日のトラフィック量の平均値と最頻出値の推移

一方、平均値はグラフ右側のヘビーユーザの使用量に左右されるため、2021年には、INの平均は684MB、OUTの平均は4225MBと、最頻出値よりかなり大きな値になります。2020年には、それぞれ609MBと3810MBでした。

モバイルでは、ヘビーユーザが少ないため、平均と最頻出値が近い値になります。表-2に示すように、今回はINの平均は僅かに減少しましたが、他の項目は増加しています。2021年の最頻出値は、INで8MB、OUTで71MBで、平均値は、INで10MB、OUTで86MBです。2020年の最頻出値は、INで7MB、OUTで63MB、平均値は、INで10MB、OUTで79MBでした。

図-6及び図-7では、利用者5,000人をランダムに抽出し、利用者ごとのIN/OUT使用量をプロットしています。X軸はOUT（ダウンロード量）、Y軸はIN（アップロード量）で、共にログスケールです。利用者のIN/OUTが同量であれば対角線上にプロットされます。

対角線の下側に対角線に沿って広がるクラスタは、ダウンロード量がひと桁多い一般的なユーザです。ブロードバンドでは、以前は右上の対角線上あたりを中心に薄く広がるヘビーユーザのクラスタがはっきり分かりましたが、今では識別ができなくなっています。また、各利用者の使用量やIN/OUT比率にも大きなばらつきがあり、多様な利用形態が存在することが窺えます。モバイルでも、OUTがひと桁多い傾向は同じですが、ブロードバンドに比べて利用量は少なく、IN/OUTのばらつきも小さくなっています。ブロードバンド、モバイル共に、2020年との違いはほとんど分かりません。

	IN（MB/day）			OUT（MB/day）		
年	平均値	中間値	最頻出値	平均値	中間値	最頻出値
2015	6.2	3.2	4.5	49.2	23.5	44.7
2016	7.6	4.1	7.1	66.5	32.7	63.1
2017	9.3	4.9	7.9	79.9	41.2	79.4
2018	10.5	5.4	8.9	83.8	44.3	79.4
2019	11.2	5.9	8.9	84.9	46.4	79.4
2020	10.4	4.5	7.1	79.4	35.1	63.1
2021	9.9	4.7	7.9	85.9	37.9	70.8

表-2 モバイル個人利用者の1日のトラフィック量の平均値と最頻出値

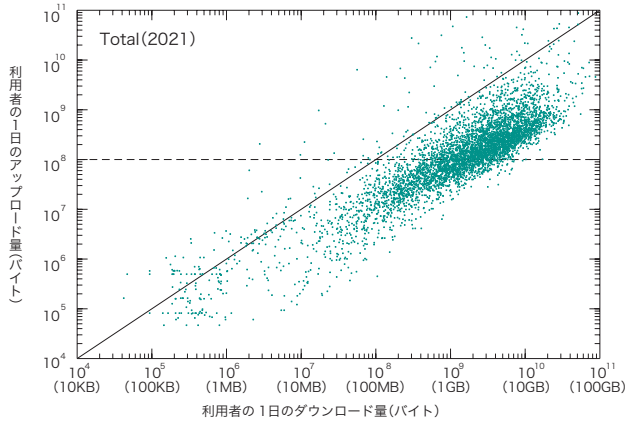

図-6 ブロードバンド利用者ごとのIN/OUT使用量

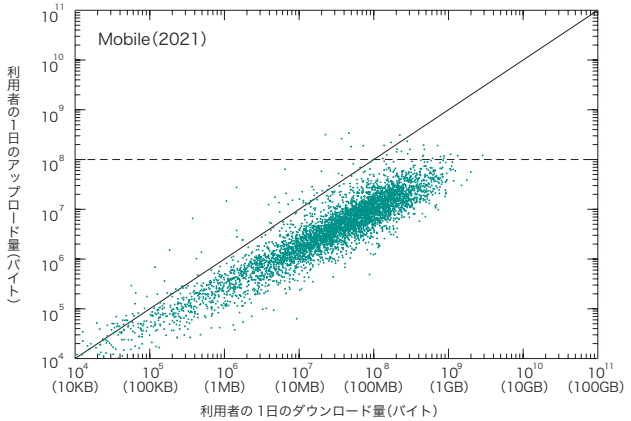

図-7 モバイル利用者ごとのIN/OUT使用量

図-8及び図-9は、利用者の1日のトラフィック量を相補累積度分布にしたものです。これは、使用量がX軸の値より多い利用者の、全体に対する割合をY軸に、ログ・ログスケールで示したもので、ヘビーユーザの分布を見るのに有効です。グラフの右側が直線的に下がっていて、べき分布に近いロングテールな分布であることが分かります。ヘビーユーザは統計的に分布しており、決して一部の特殊な利用者ではないと言えます。

ブロードバンドの分布は昨年とほとんど変わっていませんが、モバイルでは昨年観測されていた、大量にアップロードするユーザによるIN側分布の右下の出っ張りがなくなり、直線的な傾きとなっています。

利用者間のトラフィック使用量の偏りを見ると、使用量には大きな偏りがあり、結果として全体は一部利用者のトラフィックで占められています。例えば、ブロードバンド上位10%の利用者がOUTの48%、INの76%を占めています。更に、上位1%の利用者がOUTの15%、INの50%を占めています。昨年と比べても、偏りに大きな変化はありません。モバイルでは、上位10%の利用者がOUTの48%、INの49%を、上位1%の利用者がOUTの12%、INの16%を占めています。こちらも昨年のレポートから偏りにほとんど変化はありません。

1.4 ポート別使用量

次に、トラフィックの内訳をポート別の使用量から見ていきます。最近では、ポート番号からアプリケーションを特定することは困難です。P2P系アプリケーションには、双方が動的ポートを使うものが多く、また、多くのクライアント・サーバ型アプリケーションが、ファイアウォールを回避するため、HTTPが使う80番ポートを利用します。大まかに分けると、双方が1024番以上の動的ポートを使っていればP2P系のアプリケーションの可能性が高く、片方が1024番未満のいわゆる

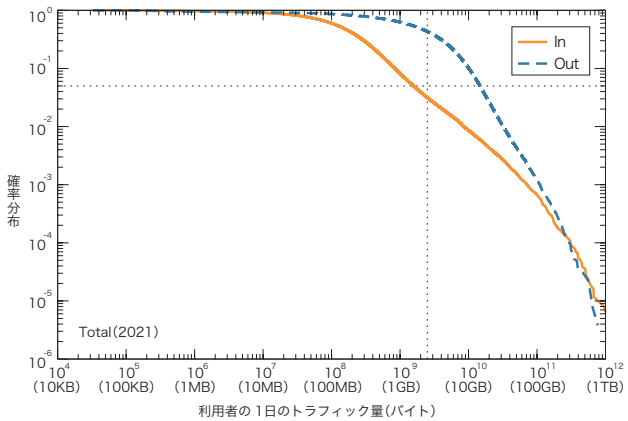

図-8 ブロードバンド利用者の1日のトラフィック量の相補累積度分布

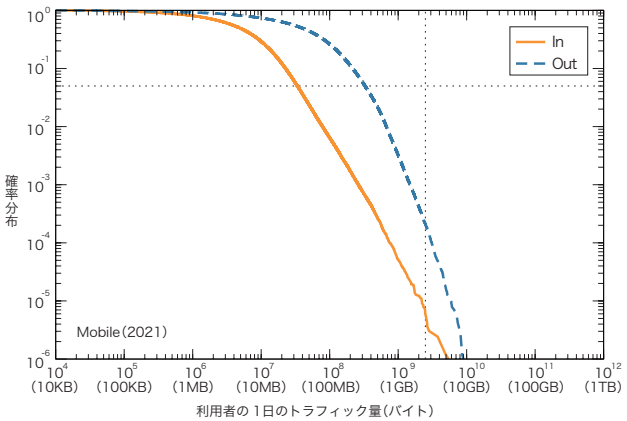

図-9 モバイル利用者の1日のトラフィック量の相補累積度分布

ウェルノウンポートを使っていれば、クライアント・サーバ型のアプリケーションの可能性が高いと言えます。そこで、TCPとUDPで、ソースとデスティネーションのポート番号の小さい方を取り、ポート番号別の使用量を見てみます。

表-3はブロードバンド利用者のポート使用割合の過去5年間の推移を示します。2021年の全体トラフィックの72%はTCPで、昨年から5ポイント減少しました。HTTPSのTCP443番ポートの割合は、54%で前回から2ポイント増加しました。HTTPのTCP80番ポートの割合は17%から12%に減っています。QUICプロトコルで使われるUDP443番ポートは、11%から16%に増えていて、HTTPの減った分とQUICの増えた分がほぼ同じです。

減少傾向のTCPの動的ポートは、2021年には6%にまで減りました。動的ポートでの個別のポート番号の割合は僅かで、最大の31000番でも0.6%となっています。また、Flash Playerが利用する1935番が0.2%ありますが、これら以外のトラフィックは、ほとんどがVPN関連です。

表-4はモバイル利用者のポート使用割合です。全体的にはブロードバンドの数字に近い値となっています。これは、スマートフォンでもPCと同様のアプリケーションを使うようになってきたことに加え、ブロードバンドにおけるスマートフォンの利用割合が増えているからだと思います。

図-10は、ブロードバンド全体トラフィックにおける主要ポート利用の週間推移を、2020年と2021年で比較したものです。TCPポートの80番、443番、1024番以上の動的ポート、UDPポート443番の4つに分けてそれぞれの推移を示しています。グラフでは、ピーク時の総トラフィック量を1として正規化して表しています。2020年と比較すると、UDP443番ポートが

year	2017	2018	2019	2020	2021
protocol port	(%)	(%)	(%)	(%)	(%)
TCP	**83.9**	**78.5**	**81.2**	**77.2**	**71.9**
(< 1024)	72.9	68.5	73.3	70.5	65.8
443(https)	43.3	40.7	51.9	52.4	53.5
80(http)	28.4	26.5	20.4	17.2	11.6
22(ssh)	0.1	0.1	0.2	0.2	0.2
993(imaps)	0.2	0.2	0.3	0.2	0.1
(>= 1024)	11.0	10.0	7.9	6.7	6.1
31000	0.1	0.1	0.2	0.4	0.6
8080	0.3	0.3	0.5	0.4	0.4
1935(rtmp)	1.1	0.7	0.3	0.4	0.2
UDP	**10.5**	**16.4**	**14.1**	**19.4**	**24.5**
443(https)	3.8	10.0	7.8	10.5	15.9
8801	0.0	0.0	0.0	1.1	0.9
4500(nat-t)	0.2	0.2	0.3	0.6	0.8
ESP	**5.1**	**4.8**	**4.4**	**3.2**	**3.3**
GRE	**0.1**	**0.1**	**0.1**	**0.1**	**0.2**
IP-ENCAP	**0.3**	**0.2**	**0.2**	**0.1**	**0.1**
ICMP	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**

表-3 ブロードバンド利用者のポート別使用量

year	2017	2018	2019	2020	2021
protocol port	(%)	(%)	(%)	(%)	(%)
TCP	**84.4**	**76.6**	**76.9**	**75.5**	**70.3**
443(https)	53.0	52.8	55.6	50.7	44.4
80(http)	27.0	16.7	10.3	7.4	5.0
993(imaps)	0.4	0.3	0.3	0.2	0.2
1935(rtmp)	0.2	0.1	0.1	0.1	0.1
UDP	**11.4**	**19.4**	**17.3**	**18.0**	**23.8**
443(https)	7.5	10.6	8.3	9.3	16.3
4500(nat-t)	0.2	4.5	3.0	1.8	3.7
8801	0.0	0.0	0.0	1.4	0.7
3480	0.0	0.0	0.0	0.4	0.3
12222	0.1	2.3	3.4	0.8	0.2
ESP	**0.4**	**3.9**	**5.8**	**6.4**	**5.8**
GRE	**0.1**	**0.1**	**0.0**	**0.1**	**0.1**
ICMP	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**

表-4 モバイル利用者のポート別使用量

TCP80番ポートより大きくなったことが分かります。昨年観測された平日昼間のトラフィック増加分が少し減っているのが分かります。全体のピークは19時から23時頃です。

図-11のモバイルでは、トラフィックの大半を占めるTCP80番ポートと443番ポート、UDP443番ポートについて推移を示します。2020年と比較すると、ここでもUDP443番ポートが更に増えていることに加え、昼休みのピークがよりはっきりと観測できるようになっています。ブロードバンドに比べると、朝から夜中までトラフィックの高い状態が続きます。平日には、朝の通勤時間、昼休み、夕方17時頃から22時頃にかけての3つのピークがあり、ブロードバンドとは利用時間の違いがあることが分かります。

1.5 まとめ

この1年半のコロナ禍のトラフィック状況を振り返ると、昨年の3月から5月は、人の移動が止まって在宅率が上昇した結果、平日昼間のブロードバンドトラフィックが大きく増えました。しかし、この期間を除くとトラフィック量は概ね成長曲線に乗って順調に伸びています。つまり、コロナ禍でトラフィック増加は年率40%程度に上昇したものの、当初危惧された劇的な増加にはならず、感染状況に伴う在宅率の変化による増減はあるものの、全体としては堅調な増加を続けています。

また、ボトルネックなどの制約のあるPPPoEに比較して、IPoEのトラフィックが大きく伸びていて、ブロードバンド全体の伸びを牽引しています。今後もIPoEの利用拡大が続くと思われます。

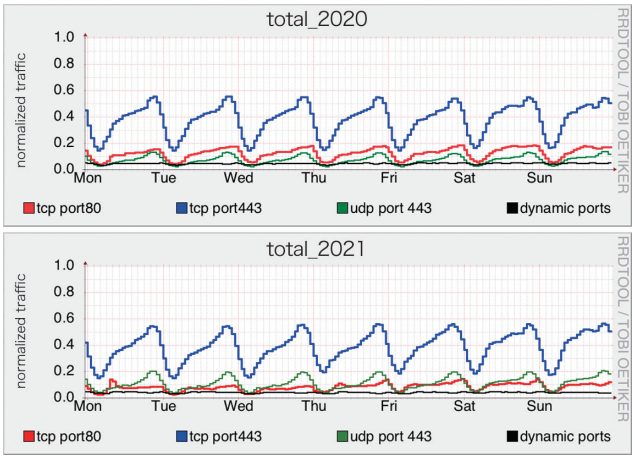

図-10 ブロードバンド利用者のポート利用の週間推移
2020年(上)と2021年(下)

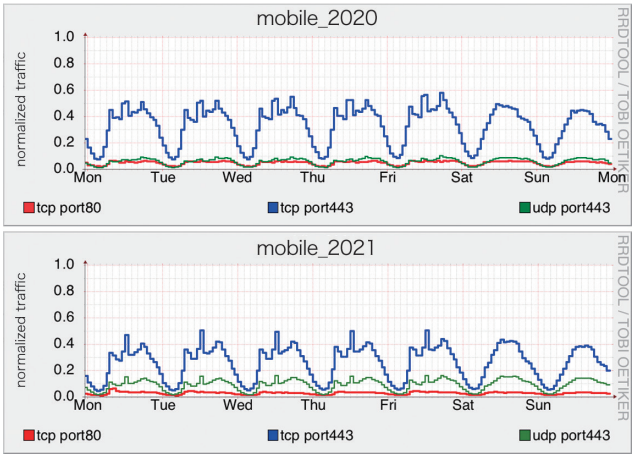

図-11 モバイル利用者のポート利用の週間推移
2020年(上)と2021年(下)

執筆者:
長 健二朗（ちょう けんじろう）
株式会社IIJ イノベーションインスティテュート 技術研究所所長。

Verifiable CredentialとBBS+署名

2.1 はじめに

新しいデジタルアイデンティティのあり方として、自己主権型アイデンティティ(Self-Sovereign Identity、SSI)が注目を集めています。デジタルアイデンティティは「自分が何者であるか」をデジタル空間の中で表現したもので、名前、生年月日、性別、メールアドレスのような属性の集まりでできています[*1]。従来、デジタルアイデンティティの管理はアプリケーションや業務システム、またはGAFAMに代表されるアイデンティティプロバイダによって行われてきました。これをアイデンティティの持ち主である自分自身が主体的に管理できるようにしようというのが自己主権型アイデンティティの考え方です。

本レポートのVol.43[*2]で自己主権型アイデンティティを取り上げてから2年が経過しました。この間に、Verifiable Credential(以下VCと略します)、非中央集権型識別子(Decentralized Identifier、DID)、デジタルエージェント、デジタルウォレット、ガバナンスフレームワークなど、自己主権型アイデンティティの実現に必要な技術や仕組みは発展を続けています。これらの網羅的な説明は他の文献[*3][*4]に譲り、本稿では自己主権型アイデンティティの中核と言えるVCにフォーカスを絞って概要を示します。また、昨年からコミュニティの関心を集めている、BBS+署名を使ったVCの実装についても簡単に紹介します。

2.2 クレデンシャルとVerifiable Credential

クレデンシャルという言葉は文脈によって様々な意味を持ちます[*5]が、本稿ではWorld Wide Web Consortium(W3C)の仕様[*6]を参考に「対象者に対する発行者のClaim(主張)の集まり」という意味で使います。例えば、運転免許証というクレデンシャルは、対象者(=免許証の持ち主)に対して発行者(=都道府県の公安委員会)が主張するClaimの集まり(=持ち主の氏名、住所、生年月日、顔写真、運転できる車の種類など)と言えます。

クレデンシャルを使うことで、私たちは「自分が何者であるか」をクレデンシャル発行者の言葉を借りて伝えることができます。例えば私が銀行口座を開設しようとして「私は○○市に住む××という者で生年月日が△△の男性である」と手ぶらで主張したとしても、銀行の担当者には信用してもらえないでしょう。こうした場合にはクレデンシャルである運転免許証を見せることで、私の主張が都道府県の公安委員会によって保証され、主張の信用度を高めることができます[*7]。

ただしクレデンシャルを使った主張を受け入れてもらうためには、相手にとって信用に足るクレデンシャルでなければなりません。それは誰によって発行されたものか。別の者に書き換えられたり偽造されたりしていないか。有効期限内であって失効されていないか。そうした検証作業を行って初めてクレデンシャルに書かれた情報が受け入れられます。

物理的なクレデンシャルの場合、相手は券面に書かれた情報を確認し、透かしのような特殊印刷があればそれをチェックすることで正しさを検証します。こうした検証作業はしばしば職人芸を要する難しいものとなります。

VCはクレデンシャルをデジタル化して機械的に検証できるようにしたものです。単に券面をスキャンして画像データにしたものとは違い、デジタル署名技術を使うことで「発行者が誰であるか」「内容に改ざんがないか」を機械的に検証することができます。Anonymous CredentialやAttribute-based

---

＊1　ISO/IEC 24760-1ではアイデンティティを「エンティティに関する属性の集合」と定義しています。
＊2　Internet Infrastructure Review Vol.43「2. ブロックチェーン技術をベースとしたアイデンティティ管理・流通の動向」(https://www.iij.ad.jp/dev/report/iir/043/02.html)。
＊3　Alex Preukschat and Drummond Reed, “Self-Sovereign Identity - Decentralized digital identity and verifiable credentials”, Manning Publications, 2021/5(https://www.manning.com/books/self-sovereign-identity)。
＊4　鈴木 研吾、合路 健人「アイデンティティはだれのもの? Hyperledger Indy & Aries で実現する分散アイデンティティ」インプレスR&D, 2021/5(https://nextpublishing.jp/isbn/9784844379447)。
＊5　Internet Infrastructure Review Vol.26「1.4.3 ID管理技術」(https://www.iij.ad.jp/dev/report/iir/026/01_04.html)。
＊6　Verifiable Credentials Data Model 1.0(https://www.w3.org/TR/vc-data-model)。
＊7　本来、運転免許証は運転資格を持つことを示すためのクレデンシャルです。氏名、住所、性別、生年月日、顔写真という代表的な属性が揃っているため、一般に本人確認のためのクレデンシャルとして利用されることが多くなっています。政府目標としてマイナンバーカードとの一体化も掲げられています。

Credentialと呼ばれる暗号理論の研究成果が土台となっており、ゼロ知識証明技術を使ったプライバシー保護の仕組みを備えることも可能です。

2.3 Verifiable Credentialの利用イメージ

VCの使われ方をより具体的にイメージするため、ここでは「住民票の写し」がVC化された世界を想像して、発行から利用までのシナリオを示します。

X市に住むAさんは、B社が提供する有料サービスを家族で利用しようと考えました。B社のサービスはX市の住民に割引が適用されます。割引を受けるためにはAさんは自身とその家族がX市に住んでいることを示す必要があります。

そこでAさんはX市の住民サービスを訪れ、VC版「住民票の写し」の発行を依頼することにしました。

X市の住民サービスは適切な方法によってAさんの本人確認を行います。例えば対面で顔写真付きの証明書の提示を求めたり、オンラインで別のVCの提示を求めたりします。

Aさんの本人確認ができたら、X市の住民サービスは住民情報データベースからAさんとその家族の属性を取得し、「住民票の写し」相当のVCを発行します。VCにはAさんとその家族の住所、氏名、生年月日、性別、住民となった年月日などが属性として含まれます。発行されたVCはAさんのスマートフォンへ格納されます。

続いてAさんはB社のサービスの利用申請を行います。AさんはX市に発行してもらったVCを提示することで、Aさんとその家族がX市在住であることを示そうとします。

AさんもB社もお互いに不要な個人情報のやり取りは避けたいと考えています。そこでAさんは、Aさんとその家族の住所だけを開示対象として、残りの氏名、性別、生年月日、住民となった年月日などは隠したまま、クレデンシャルをB社に提示します。図-1にそのイメージを示します。この例ではB社側で必要な属性(住所)をあらかじめ指定するケースを想定しましたが、Aさん側で提供する属性を選択することもあり得ます。

B社は提示されたクレデンシャルを検証し、Aさんとその家族がX市在住であることが、X市によって裏付けられていることを確認します。これによってAさんとその家族はB社のサービスを割引価格で利用できるようになりました。

なお、AさんがX市から受け取ったVCはB社のサービスに特化したものではありません。Aさんはこの後、例えば別のC社に対してX市在住であることを示したり、Aさんの家族が20歳以上であることを示したりすることも可能です。また、1つのVCを利用するだけでなく、複数のVCを組み合わせて属性を提供することも想定されています。

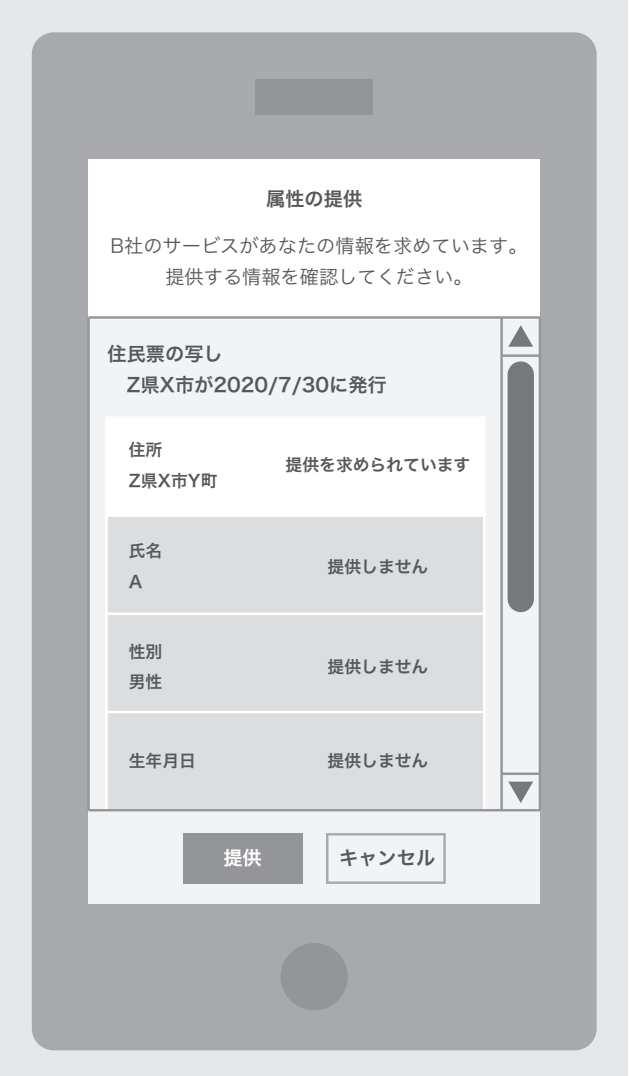

図-1 クレデンシャルの提示イメージ

2.4 Verifiable Credentialのエコシステム

上の例には、VCを発行するX市、それを受け取って提示するAさん、提示されたクレデンシャルを検証するB社が登場しました。VCに関わる登場人物とそれらの関係性はVCのエコシステムと呼ばれ、図-2のような形で整理されています。

発行者（Issuer）は、VCを発行する人や組織です。先程の例ではX市が該当します。

所有者（Holder）は、発行者が発行したVCを受け取り、自身のスマートフォンなどに保存します。そして必要に応じてそれを検証者に提示します。上の例ではAさんが「住民票の写し」VCの所有者でした。

対象者（Subject）は、VCによって主張されるClaimの対象です。多くの場合では所有者と同じですが、例えば出生証明書のように対象者が乳児、所有者はその保護者と異なることもあります。上の例でも、所有者であるAさんに加えてAさんの家族も対象者に含まれます。

検証者（Verifier）は、所有者が提示したVCを検証し、そこに書かれた情報を利用する人や組織です。上の例ではB社やC社がそれにあたります。

最後の検証可能データレジストリ（Verifiable Data Registry）は、発行者、所有者、検証者が利用するデータ置き場です。発行者の識別子や公開鍵、クレデンシャルの失効情報など、検証に必要な情報が記録されます。誰でも参照可能ですが書き換えはできません。その性質上、ブロックチェーンで実装されることが多いです。

VCや自己主権型アイデンティティはブロックチェーンと共に語られることが多く、しばしばVC自体がブロックチェーンに記録されると思われがちですが、これは誤解です。検証可能データレジストリは誰もが参照できて書き換えのできないレジストリなので、パーソナルデータを含むVCの置き場としては適切でないとされています[*8]。

先程の例で見たように、VCの2大イベントは、発行者から所有者へのクレデンシャル発行と、所有者から検証者へのクレデンシャル提示です。所有者は発行者にクレデンシャルの発行を依頼し、対象者の属性を含むVCを発行してもらいます。所有者はそれを自身のスマートフォンなどに保存した上で、必要な部分だけを検証者へ提示し、検証者がそれを検証します。結果として、発行者によって主張される属性を対象者が持っていることが、検証者によって確認されます。

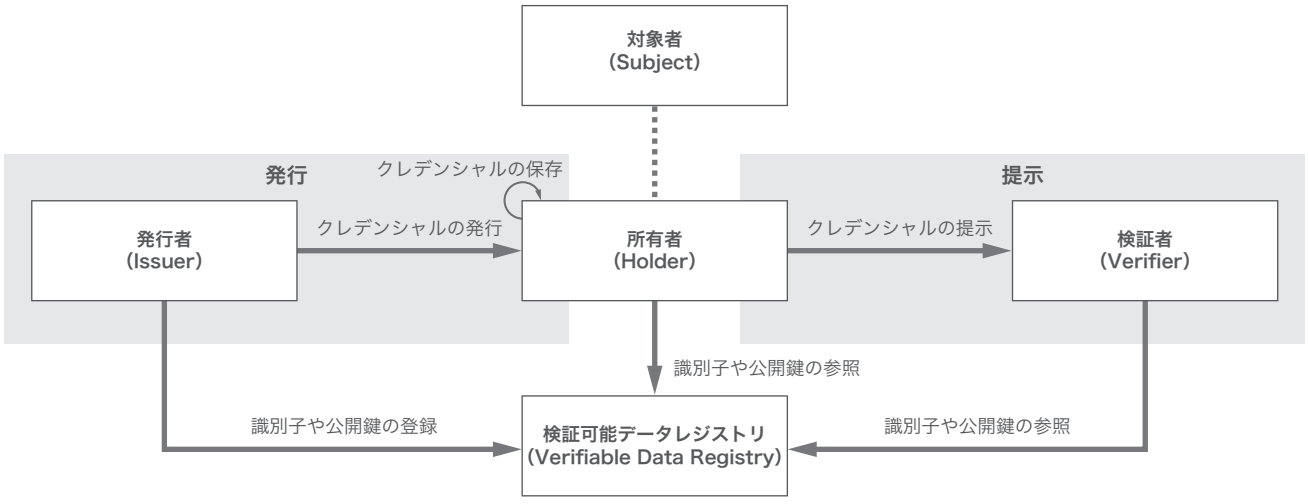

図-2 Verifiable Credential のエコシステム

*8 Stephen Curran, “Why Distributed Ledger Technology（DLT）for Identity?”, Hyperledger（https://www.hyperledger.org/blog/2021/04/21/why-distributed-ledger-technology-dlt-for-identity）。

2.5 Verifiable Credentialと従来のデジタル証明書の違い

実のところ、デジタル署名技術を使ったクレデンシャルは目新しいものではありません。私たちが日々利用しているHTTPS通信では、デジタル証明書(Digital Certificate)の検証が常に行われています。デジタルアイデンティティの連携によく使われるOpenID Connectでも、デジタル署名が付与されたIDトークンの中にアイデンティティ情報を格納し、検証可能にしています。その意味ではこれらのデジタル証明書やトークンもまた検証可能なクレデンシャル、すなわち"Verifiable Credential"であると言えそうです。

ではVCと従来のデジタル証明書の違いはどこにあるのでしょうか。筆者は主に以下の3つの点にあると考えています。すべてのVCがこれらを完全に満たすわけではありませんが、少なくとも1つ以上の特徴を備えるものがVCと呼ばれているように見受けられます。

1. 提供データを最小化する仕組みが備わっている
2. 発行者と検証者の間に必ず所有者が介在する
3. 非中央集権型識別子(DID)を使う

まず1点目について、多くのVCにはクレデンシャル所有者が開示するデータを最小化する仕組みが用意されています。中でも特徴的なのはゼロ知識証明と呼ばれる暗号学的な技術を利用するものです。ゼロ知識証明を使うことで、所有者はクレデンシャルに書かれた属性の一部を隠しながら必要な属性だけを検証者に提示できます。また、隠した属性がある条件を満たすことだけを示すことも可能です。例えば、運転免許証の氏名、住所、生年月日は隠しつつ、普通自動車の運転資格を持つことや、年齢が20歳以上であることだけを示せます。こうした仕組みは所有者や対象者のプライバシー保護にとって重要な特徴です。

2点目もまた所有者のプライバシー保護に関する特徴です。発行者をIdentity Provider(IdP)、検証者をRelying Party(RP)とみなすと、VCの仕組みはOpenID ConnectやSAMLのような現在広く使われているアイデンティティ連携の仕組みに似ています。VCがこれらと大きく異なるのは、発行者と検証者の間で直接のやり取りをさせずに、必ず両者の間に所有者が介在する点です。自己主権型アイデンティティでVCが中心的な役割を果たす理由の1つがここにあります。所有者がいつ、どの検証者に対して、どのような情報を提示したか。そうした所有者の一挙一動を、発行者や他の検証者に知られたくない場合にこうした特性が役に立ちます。

3点目は、VCと並んで自己主権型アイデンティティの要とされる非中央集権型識別子(DID)に関するものです。DIDは人、組織、モノに付けられる識別子で、デジタル署名の検証に必要な公開鍵と結びつけられるものです。DIDと公開鍵の結びつきは、認証局のような信頼できる第三者には頼らずに、ブロックチェーンなどを用いて非中央集権的に保証されます。DIDを使わずともVCのメリットを得ることは可能ですが、互いの長所を活かし合うことのできるペアとして併用されることが多いです。

2.6 Verifiable Credentialに関する国内外の動向

こうしたVCの特徴を活かし、ワクチン証明書の実装をはじめとする実験的な試みが国内外で活発に行われています。

2020年4月、VCを使って相互運用性とプライバシー保護に配慮したCOVID-19向けデジタルクレデンシャルを実現するためのCOVID-19 Credentials Initiative（CCI）という取り組みが立ち上がりました[*9]。現在はLinux Foundation Public Health（LFPH）に合流して活動を継続しています[*10]。2021年6月には国をまたがるワクチン証明書の交換基盤であるGlobal COVID Certificate Network（GCCN）もLFPHにより開始されています[*11]。並行して、今年の1月にはMicrosoft、Oracle、Salesforce他によるVaccination Credential Initiative（VCI）も発足し、VCに基づくワクチン証明書のデジタル化が進められています[*12]。

同じく2021年6月、欧州委員会が発表したEuropean Digital Identityフレームワークでは、EU（欧州連合）加盟国のすべての市民と居住者が利用可能なDigital Identity Walletというコンセプトが示されました。VCや自己主権型アイデンティティの利用は明記されていませんが、発行者・所有者・検証者で構成されるモデルやユースケース、属性の選択的開示機能など、VCの影響を強く受けていることが見て取れます[*13]。

他にもNPO団体KivaによるマイクロファイナンスのためのeKYC（オンライン本人確認）[*14]や国際航空運送協会（IATA）のIATA Travel Pass[*15]など、VCの活用分野は広がりを見せています。

国内では慶應義塾大学が国内の企業5社と共同で、Microsoftとも連携しながらVCやDIDを使った学生向けアイデンティティ基盤の実証実験を2020年10月に始めています[*16]。またTrusted Web推進協議会が2021年3月に公開したホワイトペーパーでは、Web上に検証可能な領域を増やしていくという目標に向けて、VCの応用可能性についても触れられています[*17]。

こうしたユースケースを支えるプロダクトも次々と開発されています。Linux FoundationのHyperledgerプロジェクトでは、DIDの基盤となる分散型台帳Hyperledger Indy[*18]や、VCを扱うエージェントHyperledger Aries[*19]、それらが活用する暗号ライブラリHyperledger Ursa[*20]を中心に活発な開発が行われています。また、MicrosoftのIdentity as a Service（IDaaS）であるAzure ADにはVC機能が組み込まれ、今年4月からパブリックプレビューが始まっています[*21]。

＊9　CCI（COVID-19 Credentials Initiative）（https://www.covidcreds.org/）。
＊10 LFPH（Linux Foundation Public Health）（https://www.lfph.io/）。
＊11 Introducing the Global COVID Certificate Network（GCCN）（https://www.lfph.io/2021/06/08/gccn/）。
＊12 Vaccination Credential Initiative（VCI）（https://vaccinationcredential.org/）。
＊13 European Digital Identity - European Commission（https://ec.europa.eu/info/strategy/priorities-2019-2024/europe-fit-digital-age/european-digital-identity_en）。
＊14 Hyperledger Indy、Aries、Ursaを使ったKiva Protocol。シエラレオネ共和国はこのプロトコルを採用し、国民が小口融資を受けるための本人確認を約11秒で実施可能な基盤を構築したそうです（https://www.hyperledger.org/blog/2021/01/20/kiva-protocol-built-on-hyperledger-indy-ursa-and-aries-powers-africas-first-decentralized-national-id-system）。
＊15 IATA - Travel Pass Initiative（https://www.iata.org/en/programs/passenger/travel-pass/）。
＊16 慶應義塾大学、次世代デジタルアイデンティティ基盤の実証実験を開始　在学証明書や卒業見込証明書をスマートフォンアプリへ発行（https://www.keio.ac.jp/ja/press-releases/2020/10/26/28-75892/）。
＊17 Trusted Web ホワイトペーパー ver1.0（https://www.kantei.go.jp/jp/singi/digitalmarket/trusted_web/pdf/documents_210331-2.pdf）。
＊18 Hyperledger Indy（https://www.hyperledger.org/use/hyperledger-indy）。
＊19 Hyperledger Aries（https://www.hyperledger.org/use/aries）。
＊20 Hyperledger Ursa（https://www.hyperledger.org/use/ursa）。
＊21 本人確認ソリューション- Microsoft Security（https://www.microsoft.com/ja-jp/security/business/identity-access-management/verifiable-credentials）。

2.7 Verifiable Credentialの実装

VCの標準化はW3Cで行われていますが、標準化の対象はデータモデルが中心で、細かい部分は実装によって大きく異なるのが現状です。CCIとLFPHによる解説文書[*22]は、こうした実装のバリエーションを複数の「フレーバー」として整理しています。

本稿では、Internet Identity Workshop(IIW)[*23]とその周辺で注目を集めているJSON-LD ZKP with BBS+というフレーバーを紹介します。JSON-LD ZKP with BBS+は、ニュージーランドのMATTR社が2020年4月のIIWで発表した比較的新しい方式です。コミュニティによって好意的に受け止められ[*24]、現在はMATTR社以外の技術者も加わってW3C Credentials Community Group(CCG)[*25]やDecentralized Identity Foundation(DIF)のCrypto WG[*26]において仕様策定や議論が進められています。開発はGitHub上でオープンに行われており[*27]、筆者もバグ修正などの貢献をしています。

JSON-LD ZKP with BBS+の特徴は、クレデンシャルの記述にJSON-LDというフォーマットを使う点と、デジタル署名方式としてゼロ知識証明と相性の良いBBS+署名を用いる点にあります。

JSON-LDはデジタルアイデンティティの分野ではJWT(JSON Web Token)に比べて知名度が低いものの、セマンティックWebやSearch Engine Optimization(SEO)の領域で広く利用されている仕様です。JSONデータにLinked Dataの要素を取り込み、JSONの簡潔さを保ちながら、データの記述に使う用語をURIを使って一意に特定できる点が長所です。昨今は多くのWebサイトにJSON-LDで表現されたメタデータが埋め込まれています。図-3にJSON-LDで書かれたクレデンシャルの例を示します。

BBS+署名はBBSグループ署名[*28]を拡張したマルチメッセージ型のデジタル署名です[*29*30*31]。楕円曲線暗号の一種で、

```
{
    "@context": [                              // JSON-LD コンテキスト
        "https://www.w3.org/2018/credentials/v1",
        "https://schema.org",
        ...
    ],
    "id": "http://example.edu/creds/1234",     // クレデンシャルの識別子
    "type": "VerifiableCredential",            // クレデンシャルの種類
    "issuer": "https://example.edu/issuers/1", // クレデンシャルの発行者
    "issuanceDate": "2021-06-22T00:00:00Z",    // クレデンシャルの発行日時
    "expirationDate": "2022-06-22T00:00:00Z",  // クレデンシャルの有効期限
    "credentialSubject": {
        "id": "did:example:ebfeb1f712ebc6f1c276e12ec21", // 対象者の識別子
        "type": "Person",                              // 対象者の種類
        "birthDate": "1970-01-01",                     // 対象者の生年月日
        "name": "John Smith",                          // 対象者の名前
        ...                                            // その他の属性
    },
    "proof": { ... }                           // 検証に必要な署名値など
}
```

図-3 JSON-LD クレデンシャルの例

*22 A Path Towards Interoperability: CCI Released a Paper on Different Flavors of Verifiable Credentials(https://www.lfph.io/2021/02/11/cci-verifiable-credentials-flavors-and-interoperability-paper/)。
*23 Internet Identity Workshop(https://internetidentityworkshop.com/)。
*24 Why the Verifiable Credentials Community Should Converge on BBS+(https://www.evernym.com/blog/bbs-verifiable-credentials/)。
*25 BBS+ Signatures 2020, W3C Community Group Draft Report(https://w3c-ccg.github.io/ldp-bbs2020/)。
*26 DIF - Applied Crypto Working Group(https://identity.foundation/working-groups/crypto.html)。
*27 mattrglobal/jsonld-signatures-bbs: A linked data proof suite for BBS+ signatures(https://github.com/mattrglobal/jsonld-signatures-bbs/)。
*28 Dan Boneh, Xavier Boyen, and Hovav Shacham, "Short Group Signatures", CRYPTO 2004(http://dx.doi.org/10.1007/978-3-540-28628-8_3)。
*29 Jan Camenisch and Anna Lysyanskaya, "Signature Schemes and Anonymous Credentials from Bilinear Maps", CRYPTO 2004(http://dx.doi.org/10.1007/978-3-540-28628-8_4)。
*30 Man Ho Au, Willy Susilo, and Yi Mu, "Constant-Size Dynamic k-TAA", SCN 2006(http://dx.doi.org/10.1007/11832072_8)。
*31 Jan Camenisch, Manu Drijvers, and Anja Lehmann, "Anonymous Attestation Using the Strong Diffie Hellman Assumption Revisited", Trust 2016(http://dx.doi.org/10.1007/978-3-319-45572-3_1)。

ペアリングという演算を活用しています。一般的に使われているRSA署名やECDSA署名とは異なり、(1つのデータではなく)複数のデータを並べたリストに署名を付けることが可能です。また、ゼロ知識証明と組み合わせやすい構造を備えており、署名したデータのリストから一部の要素を隠したまま署名の有効性を検証できたり、一部の要素を隠したままそれがある条件を満たすことだけを証明することもできます。

JSON-LD ZKP with BBS+では、JSON-LDで書かれたクレデンシャルをLD Canonicalizationという方法でstatementと呼ばれるデータに分解します。そしてstatementのリストに対してBBS+署名の生成や検証を行います。例えば図-3のJSON-LDクレデンシャルは図-4のようなstatementのリストに変換されてから署名されます。BBS+署名を使ってstatementのリストに署名を付けることで、statement単位で見せる/見せないの制御が可能になります。ただし、statementの中に含まれる値(氏名や生年月日など)を隠したまま、それらがある条件(例えば生年月日がある範囲に収まることなど)を満たすような高度な証明はまだ実現できていません。

2.8 Verifiable Credentialの今後

VCやJSON-LD ZKP with BBS+の実用化に向けては解決すべき課題も残っています。ここでは代表的な3つの課題を示し、各課題の解決に向けたアプローチや動向を紹介します。

■ 課題1:既存のアイデンティティ連携技術との相互運用性

VCという新しい概念と、既存のデジタルアイデンティティ関連仕様や製品との相互運用性の確保が第一の課題です。これに対してはOpenID Connectがもともと備えているSelf-Issued OpenID Provider(SIOP)という枠組みを使って、VCをOpenID Connectの上で扱う方法がOpenID Foundation(OIDF)で検討されています。検討にはJSON-LD ZKP with BBS+の提案者であるMATTR社の技術者も参加しています。

■ 課題2:各種仕様の標準化

上で紹介したJSON-LD ZKP with BBS+やLD Canonicalizationの仕様はいまだ議論の途中であり、標準仕様としては確立されていません。JSON-LD ZKP with BBS+に関しては上で述べたとおり、W3C CCGでの仕様策定とDIFのCrypto WGにおける議論が並行して行われています。検討の固まった内容から順にW3C仕様として標準化され、将来的な内容について

```
<did:example:ebfeb1f712ebc6f1c276e12ec21> <http://schema.org/birthDate> "1970-01-01"^^<http://schema.org/Date> .
<did:example:ebfeb1f712ebc6f1c276e12ec21> <http://schema.org/name> "John Smith" .
<did:example:ebfeb1f712ebc6f1c276e12ec21> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://schema.org/Person> .
<http://example.edu/creds/1234> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <https://www.w3.org/2018/credentials#VerifiableCredential> .
<http://example.edu/creds/1234> <https://www.w3.org/2018/credentials#credentialSubject> <did:example:ebfeb1f712ebc6f1c276e12ec21> .
...
```

図-4 署名されるstatementのリスト(一部抜粋)。一行一行がstatementと呼ばれる

は後者のWGで検討や議論が重ねられていくようです。例えば、生年月日を明かさずに20歳以上であることだけを示すような高度な証明の実現方式については、後者のCrypto WGの議論対象に含まれています。LD Canonicalizationに関しては、W3Cで現在立ち上げ準備が進んでいるLinked Data Signatures WGの中で、RDF Dataset Canonicalization(RDC)やLinked Data Integrity(LDI)として検討が進む見込みです。本稿執筆時(2021年8月)のProposed Charter[*32]には、2021年9月から作業を開始し、2023年9月までの2年間でW3C Recommendation化を目指す旨が記載されています。

■ 課題3:量子コンピュータへの耐性

3点目は長期的な課題として、本レポートのVol.49[*33]でも取り上げた「耐量子計算機暗号」に関する話題を紹介します。BBS+署名は楕円曲線上の離散対数問題の難しさに安全性の根拠を置いています。これは量子コンピュータによって効率良く解かれてしまうことが分かっている問題です。したがってBBS+署名やそれを用いるJSON-LD ZKP with BBS+方式は、残念ながら量子計算機への耐性を持ちません。それはHyperledger Indyが採用しているCamenisch-Lysyanskaya(CL)署名や、JWTでよく利用されるRSA、ECDSA、EdDSA署名も同様です。耐量子計算機暗号として知られる格子ベースの署名方式やZero-Knowledge Scalable Transparent Arguments of Knowledge(ZK-STARK)の応用も提案されていますが、実用に向けてはまだ性能向上を含めた改善の余地が多く残されている状況です。

2.9 おわりに

国内外で注目を集めるVCと、その実装形態の1つであるJSON-LD ZKP with BBS+フレーバーについて、現在の状況と今後の課題を紹介しました。個人的には、VCはこれまでのデジタル証明書やIDトークンをすべて置き換えてしまうものではなく、適材適所での使い分けが進むものと考えています。人、組織、モノのプライバシーが保護されるべき場面、特に提供するデータの最小化が望まれる場面でVCの真価は発揮されます。また、組織や業界にまたがる広い範囲でのデジタルアイデンティティ連携において、JSON-LDを使ったVCはその表現力と相互運用性の高いクレデンシャル記述を活かせます。実用化に向けた課題は山積していますが、関係団体による標準化や普及の活動を注視しつつ、コミュニティや社会の発展に向けて些少なりとも貢献していきたいと考えます。

執筆者:
山本 暖(やまもと だん)
IIJ セキュリティ本部 セキュリティ情報統括室 シニアエンジニア。
2021年より現職。デジタルアイデンティティと情報セキュリティに関わる調査・研究活動に従事。

*32 Linked Data Signatures Working Group Charter(https://w3c.github.io/lds-wg-charter/index.html)。
*33 Internet Infrastructure Review Vol.49「2. 耐量子計算機暗号の動向2020」(https://www.iij.ad.jp/dev/report/iir/049/02.html)。

HaskellによるQUICの実装

IIJの目標の1つはインターネットの発展に貢献することであり、技術研究所は貢献方法の1つとして新しいプロトコルの標準化に参加しています。新しいプロトコルの仕様を議論し、その仕様を実装し、他の実装と相互接続性を検証することで、仕様の完成度を高めることに長年取り組んでいます。

2013年以降はHTTP/2とTLS 1.3の標準化に参加しました。最近の2年半は、この2つのプロトコルに関連が深いQUICやHTTP/3の標準化に関わりました。この記事では、QUICやHTTP/3をどのように実装したかについて説明します。

3.1 QUICとHTTP/3

QUICは、UDPを利用する新しいトランスポートプロトコルです。以下の機能を取り込む大きな仕様として定義されています。

- TCPが提供する信頼性、フロー制御、及び輻輳制御
- HTTP/2由来の非同期ストリームによる多重化
  (ストリームの分割と再構成)
- TLS 1.3が提供するセキュリティ機能
  (鍵交換や通信相手の認証、通信の暗号化)

QUICの基本単位は「パケット」と呼ばれます。パケットには、「フレーム」と呼ばれる単位のデータを複数格納できます。フレームにはいくつかの種類があり、例えばアプリケーションのデータは、STREAMフレームに収められます。ACK(確認応答)の情報は、ACKフレームに格納されます。QUIC上に定義されたHTTPが、HTTP/3と呼ばれています。

3.2 Haskellで実装する理由

QUIC及びHTTP/3は、これまでに実装したHTTP/2やTLS 1.3と同様に、プログラミング言語Haskellで実装しています。我々がHaskellを選んだ理由は以下のとおりです。

- 豊かなデータ型で問題を簡潔に表現でき、強い型検査でコードの誤りの多くを検出可能
- 軽量スレッドが標準で提供されているおかげで、状態管理が煩雑になりがちなイベント駆動プログラミングよりも見通しが良く、なおかつスレッドを切り替えたり生成したりする際のオーバーヘッドが少ないスレッドプログラミングが可能(以下、単にスレッドと言えば、軽量スレッドを意味する)
- 多くのデータ型が不変であり、安全にスレッド間で共有できる
- STM (Software Transactional Memory)が標準で提供されており、デッドロックのないスレッドプログラミングが可能

我々以外のチームが手がけるQUICの多くの実装では、イベント駆動プログラミングを採用しています。これに対して我々はスレッドプログラミングを採用しています。スレッドプログラミングを採用することでプログラムの見通しが良くなるだけでなく、他の実装者とは違う観点で仕様を検証できると考えています。

以下、具体的な実装のポイントについて説明します。

3.3 QUICのストリームとコネクション

QUICは、1つのコネクションの中で通信を多重化するために通信を「ストリーム」という単位で分割します。HTTP/2においても同様の目的で「ストリーム」を用いますが、HTTP/2のストリームがHTTPの要求と応答しか運べない一方、QUICのストリームはどのようなアプリケーションのデータでも運搬できます。

QUIC用のAPIを長期に渡って考案した結果、以下のような抽象化を発見しました。

- QUICのコネクションの役割は、OSが担っているネットワーク入出力管理に相当する
- QUICのストリームは、TCPコネクションに相当する

ここでのTCPコネクションは、HTTP/1.0のように、コンテンツを1つだけやり取りする最も単純な利用形態のTCPコネクションを意味しています。このように考えると、ストリームはソケットAPIを模倣したAPIで操作できることに気付きました。現時点でのAPIの一部を以下に示します。

```
-- ストリームを表す抽象データ型
data Stream

-- ストリームを作成する関数
stream :: Connection -> IO Stream

-- ストリームを閉じる関数
closeStream :: Stream -> IO ()

-- 通信相手が作成したストリームを受け取る関数
acceptStream :: Connection -> IO Stream

-- ストリームからデータを受信する関数
recvStream :: Stream -> Int -> IO ByteString

-- ストリームにデータを送信する関数
sendStream :: Stream -> ByteString -> IO ()
```

このようにHaskellの型注釈は、型を右矢印で区切った形で表現されます。一番右に返り値の型を書きます。それ以外は引数の型です。型の左横にIOが付くと、入出力など副作用を伴うという意味です。付いていなければ副作用がない不変データ型です。()は返り値がない型、ByteStringはバイト列の型です。よって、IO ()は返り値には意味がなく、副作用だけに興味がある型です。

HaskellでHTTP/1.0サーバを実装する際は、クライアントからのTCPコネクション1つにつき、1つのスレッドを起動し、要求を読んで応答を書き込み、スレッドを終了させる同期的な手法が常套手段です。HTTP/2では、多重化を実現するために、複数のスレッドを管理する必要がありました。HTTP/3を実現する際は、この多重化はQUICライブラリが担います。そのため、前述のAPIを用いると、1つのストリームにつき、1つのスレッドを起動する同期的な常套手段の利用が可能となりました。

3.4 サーバでのコネクションの受け付け

サーバを起動する関数の型注釈は、以下のとおりです。

```
run :: ServerConfig -> (Connection -> IO ()) -> IO ()
```

すなわちrunは、サーバの設定とサーバアプリケーション関数(コネクションを受け取って入出力を含む何らかの処理をする関数)を引数に取ります。この関数で起動されるDispatcherスレッドは、ネットワークインタフェースごとに待ち受け(ワイルドカード)ソケットを開きます。新しいコネクションを受け付けると、コネクションを構成するスレッド群を起動します(3.5節参照)。

QUICパケットは6種類あります。Initialパケット、0-RTTパケット、Handshakeパケット、及び1-RTTパケットは本文が暗号化され、ヘッダも保護されます。Version Negotiationパケットと Retryパケットは、本文の暗号化やヘッダの保護が施されません。これらのパケットを統一的に解析するために、解析を2段階に分ける方法を考案しました。

(1) ヘッダ保護で守られていないヘッダ領域の構文解析
　(パケットの種類の判別など)
(2) 暗号化されている本文の復号とヘッダ保護の除去

(1) はDispatcherスレッドで実行されます。Dispatcherスレッドは、(1)の解析結果を見て、Initialパケットであれば新しいコネクションを作成し、適切な1-RTTパケットであればマイグレーション処理を実施します(3.9節参照)。また、Version NegotiationパケットやRetryパケットをサーバが受け取ることは仕様上不正なので、その場合は単に破棄します。

(1) は後述のReaderスレッドでも実行され、(2) は後述のReceiverスレッドで実施されます。この2段階の解析というアイディアにより、ヘッダ情報のデータ構造が共通化され、それ以前の実装に比べてコードが簡潔になりました。

3.5 コネクションを構成するスレッド群

Dispatcherスレッドが新しいコネクションを作成する際は、そのコネクションのメインとなるスレッドを起動し、作成を依頼します。メインのスレッドは、図-1に示すようなコネクションを構成するスレッド群を起動し、終了を待ちます。

コネクションの生成時には、接続済みソケットが生成されます。そのため、このコネクションに関係するパケットは、Dispatcherスレッドではなく、Readerスレッドが読み込むようになります。Readerは前述のパケット解析(1)を実行し、解析できたヘッダ情報、保護されたヘッダ、及び暗号化された本文を、キュー(RecvQ)を通じてReceiverスレッドに渡します。

Receiverスレッドは、前述の(2)を実行し、パケット中のフレーム群を取り出し、それぞれを処理します。STREAMフレームに関しては、再構成をして、キュー(InputQ)を通じてServerスレッドに渡します。また、ACKフレームを受け取ると、3.10節で述べる再送情報コンテナ(SentPackets)から対応する情報を削除します。

Serverスレッドは、サーバアプリケーション関数を起動するスレッドです。出力は、キュー(OutputQ)を通じてSender

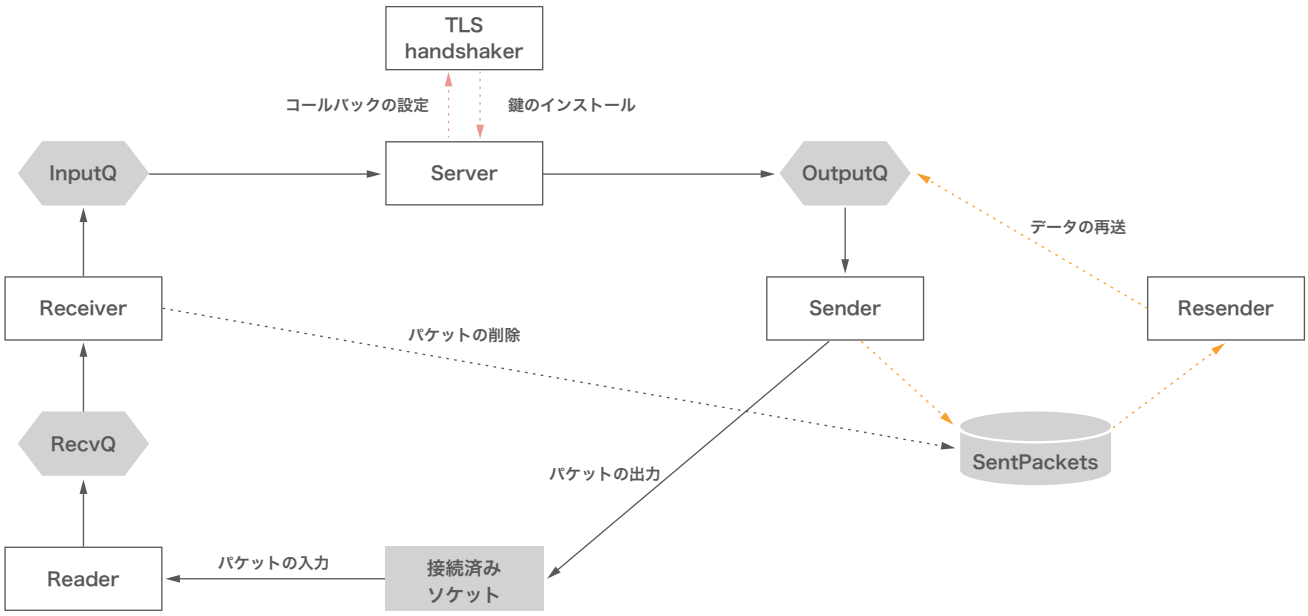

図-1 コネクションを構成するスレッド群

スレッドに送られます。Serverスレッドは、アプリケーションを起動する前に、TLS handshakerスレッドを起動し鍵交換を実行させると共に、鍵が利用可能なタイミングの同期を取る役割も果たします。

Senderスレッドは、Serverスレッドからの出力を分割してSTREAMフレームに格納し、QUICパケットを構成します。そして、パケットの本文を暗号化し、ヘッダを保護して、最終的なUDPデータグラムを生成し、接続済みのソケットを通じて出力します。また、QUICパケットに含まれた情報を再送情報コンテナに格納します。

Resenderスレッドは、パケットのロスを検知すると、再送情報コンテナから関連する情報を取り出し、OutputQに入れることで再送します。

キューやその他のデータの共有にはSTMを活用しており、これらのスレッドはデッドロックを起こすことはありません。また、どれか1つのスレッドが致命的なエラーを起こすと、スレッド群全体が消滅します。この際、リソースは正しく解放され、リークは起こりません。

3.6 接続済みソケット

TCPではaccept()システムコールを利用すれば、ワイルドカードソケットから、接続済みソケットを生成できます。しかしながら、accept()システムコールはUDPに対しては使用できません。

例えば、サーバにワイルドカードソケット{UDP, 192.0.2.1, 443, *, *}が存在し、203.0.113.1:3456のクライアントから接続要求があったとしましょう。生成したい接続済みソケットは{UDP, 192.0.2.1, 443, 203.0.113.1, 3456}です。素朴な方法としては、以下が考えられます。

(1) 新しいUDPソケットを開き、SO_REUSEADDRオプションを設定する
(2) 192.0.2.1:443を指定してbind()システムコールを呼ぶ
(3) 203.0.113.1:3456を指定してconnect()システムコールを呼ぶ

残念ながらBSD系OSでは、(2)でエラーを起こします。Linuxは(2)を受け付けますが、レースコンディションが発生すると思われます。これらの問題を解決する方法は、以下のとおりです。

(1) 新しいUDPソケットを開き、SO_REUSEADDRオプションを設定する
(2) *:443を指定してbind()システムコールを呼ぶ
(3) 203.0.113.1:3456を指定してconnect()システムコールを呼ぶ。この際、ローカルアドレスが192.0.2.1に設定される

この方法は、多くのOSで問題なく動作し、レースコンディションも発生しません。ただし、特権の扱いには注意する必要があります。TCPで、root特権を保有しているプロセスが特権ポートに対するワイルドカードソケットを生成したとしましょう。このプロセスがセキュリティ上の理由からroot特権を手放したとしても、accept()システムコールは問題なく実行できます。しかし、上記の方法でUDP接続済みソケットを生成する際には、Linuxでは最低限CAP_NET_BIND_SERVICEケーパビリティが必要です。

3.7 コネクションの消去

ソケットAPIでTCPを利用している場合、アプリケーションがclose()システムコールを呼んだ際は、制御は直ちにアプリケーションに戻され、以降のTCPの終了処理はOSが担います。QUICの実装でも、このような制御を可能にすべきでしょう。

我々の実装では、サーバ(あるいはクライアント)アプリケーション関数が終了する際は、メインのスレッドを除いた他のスレッド群は終了し、不要な情報は破棄されます。また、CONNECTION_CLOSEフレームを再送するための最小限の情報と共に、終了処理用の別スレッドがメインのスレッドから起動されます。

QUICでは、CONNECTION_CLOSEフレームを含むパケットには、ACKが返ってきません。通信相手がCONNECTION_CLOSEフレームを受け取ったなら、直ちにあらゆるパケットの送信を停止します。そこで、CONNECTION_CLOSEフレームを送った後は、一定期間待ち、相手からのパケットが届かなくなることを確かめます。届く場合は、CONNECTION_CLOSEフレームが紛失したと考えられるので、CONNECTION_CLOSEフレームを含むパケットを再送します。

3.8 TLSハンドシェイク

QUICでは、TLS 1.3を利用してハンドシェイクを実行し、通信相手を認証したり、鍵を交換したりします。TLS 1.3のメッセージは、TLSのレコード層から切り離され、単なるデータ書式として、CRYPTOフレームに格納されます。

QUICでのフルハンドシェイクの様子を図-2に示します。

クライアントは、乱数的に生成したコネクションIDを元にInitial鍵を生成します。TLS 1.3のClientHelloをCRYPTOフレームに収め、更にInitialパケットに収め、Initial鍵で暗号化して送信します。Initial鍵は中継装置でも生成できるためプライバシは保護されていないことに注意してください。

サーバはこれを受信したあと、Initial鍵を生成してInitialパケットを復号します。次に、取り出したClientHelloを元に、Handshake鍵と1-RTT鍵を生成します。そして、生成したServerHelloはInitialパケットに収めてInitail鍵で暗号化して送信し、その他のTLSメッセージはHandshakeパケットに収めHandshake鍵で暗号化してから送ります。

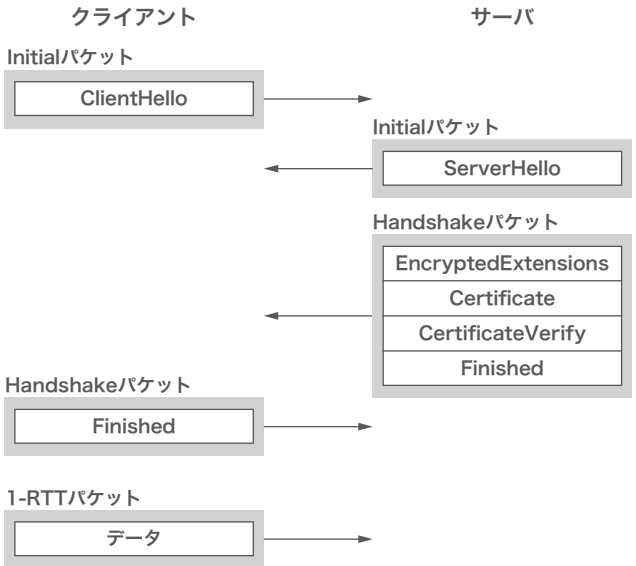

図-2 QUICのフルハンドシェイク

これらを受け取ったクライアントは、Handshake鍵と1-RTT鍵を生成します。また、生成したFinishedをHandshakeパケットに格納し、Handshake鍵で暗号化して送信します。この時点で、アプリケーションデータを格納できる1-RTTパケットが送信可能になります。

2回目のコネクションの場合、クライアントは保存している情報から0-RTT鍵を生成し、Initialパケットの後に、アプリケーションデータを格納できる0-RTTパケットを0-RTT鍵で暗号化して送信できます。

TLS 1.3の機能をQUICで利用可能とするために、TLSライブラリの拡張を様々な方法で試みました。レコード層を分離するためには大改造が必要でしたが、それ以上の改造をせずにTLSライブラリを再利用するには、TLS専用のスレッドを起動する方法が適切だと分かりました。

Client/ServerスレッドでTLS 1.3の状態を管理しなくて済むよう、TLSライブラリ内に状態を閉じ込めるには、コールバック方式が有効でした。鍵を生成すると、指定されたコールバックを用いて、共有データに鍵をインストールします。また、STMを利用することで、鍵がインストールされたタイミングを他のスレッドが測れるようにしました。

サーバ側のTLS handshakerスレッドは、NewSessionTicketメッセージを1-RTTパケットで送信したあとに終了します。一方、クライアント側のTLS handshakerスレッドは、HANDSHAKE_DONEフレームを受け取った後、一定時間後に終了します。

3.9 マイグレーション

クライアント側のIPアドレスやポート番号が変更されることがあります。例えばネットワークインタフェースを携帯電話からWi-Fiに切り替えたり、途中にあるNATがポートの対応表を変更したりした際に起こります。このような場合でも、コネクションを継続させる機能がコネクションマイグレーションです。

クライアント側のIPアドレスやポート番号が変更されると、我々の実装のサーバ側では待ち受けソケットに1-RTTパケットが届くようになります。コネクションIDを調べると、不正なパケットではなく、マイグレーションが起こったと判断できます。

この際、DispatcherスレッドはMigratorスレッドを起動します(図-3)。Migratorスレッドは、新しい接続済みソケットを生成し、そのソケットを利用するReaderスレッドを起動し、パス検証を実施します。パス検証の詳細についてはRFC9000[*1]を参照してください。

Dispatcherスレッドは、新しい接続済みソケットを生成するまでに届いたパケットをMigratorスレッドに渡し、Migratorスレッドはそれらをreceiverスレッドに渡します。また、一定期間の後、古い接続済みソケットを閉じることで、古いReaderスレッドを終了させます。

ここで、クライアント側でのマイグレーションの方法を考察します。まず、接続済みソケットを使う場合です。

(1) 何らかの方法で、優先させる新しいネットワークインタフェースが利用可能になったことを検知する
(2) マイグレーションのAPIを呼ぶ。新しいソケットが生成され、connect()システムコールが呼ばれると、OSはリモートのアドレスとポートを引数から設定する。そして、経路表をリモートのアドレスで検索し、経路が向いているネットワークインタフェースを得る。そのネットワークインタフェースのIPアドレスが、ソケットのローカルアドレスに選ばれる。ローカルポートは、乱数的に選ばれる
(3) 作成した接続済みソケットとsend()を使ってパケットを送信する

この方法の利点は(2)でパス検証を実施できること、欠点は(1)を実施するためにはOS固有の手法を使わなければならないことです。一方、ワイルドカードソケットを利用する方法も考えられます。

・ sendto()が呼ばれると、OSは引数からリモートのアドレスとポートを設定する。また、経路表をリモートのアドレスで検索し、経路が向いているネットワークインタフェースを得る。そのネットワークインタフェースのIPアドレスが、ソケットのローカルアドレスに選ばれる。ローカルポートは、最初にsendto()が呼ばれた際に、乱数的に選択される

この方法の利点は、優先されるネットワークインタフェースを見張る必要もなければ、特別なマイグレーションのAPIを用意する必要もなく、マイグレーションが自動的に実施されることです。欠点は、パケットの送信にコストがかかって性能が悪くなること、パス検証をするタイミングがないことです。

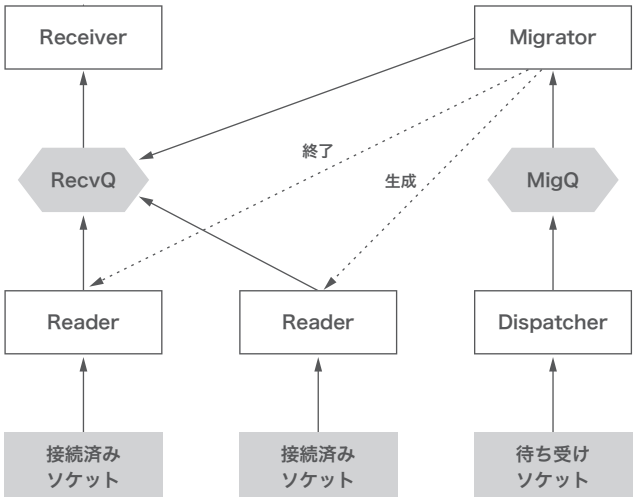

図-3 コネクションマイグレーションの流れ

*1 参考文献 J. Iyengar, M. Thomson, "QUIC:A UDP-Based Multiplexed and Secure Transport", RFC9000, 2021

どちらの方法にも一長一短があります。そこで、両方の方法を提供し、クライアントを起動する際の設定で選択できるようにする予定です。

3.10 ACK処理のアルゴリズム

QUICでは再送の際に新しいパケット番号が使われます。再送の際にシーケンス番号を再利用するTCPと違って、QUICにはACKの曖昧さの問題が存在しません。再送の際には、パケット番号や暗号文も変わります。そのためRFC9000では、単純なパケットの再送ではなく「情報の再送である」と表現されています。

標準的なTCPでは、ACKは次に届くべきシーケンス番号を指定するので、その他のTCPセグメントが通信相手に届いているか判定できません。一方で、QUICのACKフレームには、受信したパケット番号を列挙できます。

送信したパケットの情報を再送できるようにするには、再送情報コンテナを用意して、送信時に情報を蓄えます。再送情報コンテナの操作は、以下の3つが考えられます。

- 送信時に、パケット番号をキーとして、情報を挿入し保存する
- ACKの受信時に、パケット番号をキーとして、情報を削除する
- 一定期間ACKが返らない場合、情報を削除して取り出し、それぞれを再送する

このような操作が提供されているデータ構造で、Haskellで一般的なのは、PSQ(Priority Search Queue)です。キーとしてパケット番号、プライオリティとして送信時間、そして値として情報を指定します。

再送情報コンテナをPSQで実装したところ、性能が著しく悪くなる場合があることに気付きました。通常の実装では、ACKを例えば以下のように返します。

```
[0,1,2,3]
[4,5,6,7]
[8,9,10,11]
```

すなわち、ACKに対するACKを処理して、ACKを返すべきパケット番号を動的に管理します。一方で、一時のFirefox Nightlyは以下のようなACKを返していました。

```
[0,1,2,3]
[0,1,2,3,4,5,6,7]
[0,1,2,3,4,5,6,7,8,9,10,11]
```

ACKに対するACKを処理しないので、不要になったパケット番号が削除されません。この形態は、仕様では許されています。PSQの大きさをn、ACKに指定されたパケット番号の数をmとすると、全体の削除操作の計算量はO(m log n)です。Firefox Nightlyのようにmが巨大になると、削除操作は極めて高価になります。

この問題を解決するために、述語(predicate)を活用する方法を思いつきました。例えば、`[4,5,7,8,9]`というパケット番号群は、ACKフレームの中では`[(4,5),(7,9)]`のように範囲で指定されます。この範囲を以下のような述語に変換します。

```
predicate :: PlainPacket -> Bool
predicate pkt = (4 <= n && n <= 5) || (7 <= n && n <= 9)
  where
    n = packetNumber pkt
```

Haskellには、双方向リストのように両端の操作がやりやすいデータ構造であるFinger Treeが標準で提供されています。このFinger Treeには、述語に合致する要素のみを含むFinger Treeと、合致しない要素のみを格納するFinger Treeとに分割する操作があり、その計算量はO(n)です。そこで、PSQの代わりにFinger Treeと述語による分割を組み合わせることで、ACKを受信した際の操作の計算量を低減することに成功しました。

3.11 ストリームの再構成

QUICパケットは、複数のIPパケットに跨りません。すなわち、IPレベルにおいて分割されたり、再構成されたりしません。一方で、ストリームのデータは複数のQUICパケットに跨ります。そのため、送信側ではストリームのデータを適切な大きさに分割し、受信側では再構成する必要があります。

STREAMフレームが到着した際は、その断片を再構成用のコンテナに挿入します。そして、期待しているオフセットから始まる連続した断片群が存在するなら、それらを取り出してrecvStreamのキューに入れます。このように、再構成用のコンテナの操作には、挿入と取り出し削除があります。

我々の古い実装では、再構成用のコンテナに一方向リストを利用していました。挿入と取り出し削除の計算量は、それぞれO(n)とO(n)です。実践環境での通信のプロファイルを取ると、ストリームの再構成がボトルネックとなっていました。

そこで、再構成用のコンテナのデータ構造として、前述のFinger Treeを要素としたねじれヒープを採用しました。Finger Treeは、先頭にも末尾にも要素をO(1)で追加でき、連続した断片群を表現します。挿入と取り出し削除の計算量は、それぞれO(log n)とO(n)となり、計算量を低減できました。

3.12 フロー制御

フロー制御とは、受信者が受信できる範囲で、送信者がパケットを送信する仕組みです。QUICでは、受信できるデータ量(クレジット)を送信者に伝える方式を採用しています。3.13節で述べる輻輳制御と同一視されがちですが、別々の仕組みです。

我々の実装では、ストリームAPIのレベルでフロー制御を実現しています。

- sendStreamは、通信相手が許す範囲でデータを送り、超える場合は、通信相手からのクレジットを待ちます
- recvStreamは、アプリケーションがこのデータを消費すると考え、受信したデータ分のクレジットを通信相手へ送ります

3.13 ロス検知と輻輳制御

QUICのロス検知と輻輳制御は、RFC9002[*2]で定義されています。ロス検知は、ACKに基づく方法とprobeタイムアウトに基づく方法の両方を活用します。また、輻輳制御にはNewRenoに基づくアルゴリズムを採用しています。我々は、掲載されている疑似コードを忠実にHaskellで実装しました。その過程で、仕様の不整合をいくつか発見し報告しました。この貢献が認められて、この記事の筆者である山本の名前が、RFC9002の貢献者リストに掲載されました。

ロス検知と輻輳制御に関しては、qlog形式[*3]で書き出したログをビジュアライザーであるqvis[*4]で可視化し、動作を確認したり、間違いを発見したりしました。

＊2　参考文献 J. Iyengar, I. Swett, "QUIC Loss Detection and Congestion Control", RFC9002, 2021
＊3　参考文献 R. Marx, "QUIC and HTTP/3 event definitions for qlog", Internet-Draft, 2020
＊4　qvis、"Welcome to qvis v0.1, the QUIC and HTTP/3 visualization toolsuite!"(https://qvis.quictools.info/)。

3.14 テスト

我々のQUICライブラリやHTTP/3ライブラリでは、様々な単体テストを実装しています。この節では、特筆すべき単体テストと外部テストの利用に関して説明します。

■ ロス検知

ロス検知が正しく動くかテストするために、中継スレッドを通じてUDPデータグラムを中継する仮想ネットワークを実装しました。中継スレッドは与えられたシナリオを元に、UDPデータグラムを落とします。乱数的にUDPデータグラムを落とすテストはもちろん実装しています。また、クライアントの第1パケットを落とすテスト、第2パケットを落とすテストのように、問題を起こしやすいハンドシェイクのパケットロスに対して、すべてのパターンを網羅しています。

■ h3spec

テスト項目で漏れがちなのが、エラーケースです。HTTP/2の場合、エラーケースをサーバに対してテストする優れたツールh2spec[*5]が存在します。我々は、HaskellのQUICライブラリにフックを用意すれば、簡単にエラーケースに対するテストを実現できることに気付きました。以下にフックの一部を示します。

```
onTransportParametersCreated :: Parameters -> Parameters
```

このフックはトランスポートパラメータが生成された際に、その値から別の値へ変換します。エラーとなる値へ変換することで、エラーケースを作成できます。このアイディアに基づいて、QUICあるいはHTTP/3サーバに対してエラーケースをテストするh3spec[*6]というツールをリリースしました。現時点では、QUICのエラーテストを32項目、HTTP/3のエラーテストを16項目提供しています。h3specは、HaskellのQUICライブラリに対してはもちろん、他の実装のテストにも利用され、実装を安定化させることに役立ちました。

■ QUIC tracker

QUIC trackerは、公開サーバに対して様々なテストを1日1回実行し、その結果を公開するサービスです。我々の公開サーバも登録していただき、たくさんのバグを発見できました。最終的には、対応していない2項目を除いたすべてのテストケースでパスするようになりました。

3.15 謝辞

接続済みソケットの生成方法のアイディアは、奥一穂さんからいただきました。また、マイグレーションのAPIに関しては、Tatsuhiro Tsujikawaさんに議論していただきました。Robin Marxさんにはqlogやqvisに関して様々なことを教えていただきました。ここに名前を記して感謝します。

執筆者:
山本 和彦（やまもと かずひこ）
株式会社IIJイノベーションインスティテュート 技術研究所 技術開発室 室長
プログラミング言語Haskellの並行技術をネットワークプログラミングへ応用することに興味を持つ。
IIJエンジニアブログに「QUICをゆっくり解説」を連載中。

＊5　h2spec、"A conformance testing tool for HTTP/2 implementation"（https://github.com/summerwind/h2spec）。
＊6　h3spec、"Test tool for error cases of QUIC and HTTP/3"（https://github.com/kazu-yamamoto/h3spec）。

Information

IIJ Engineers Blog 最新トピック

今号(IIR vol.52)の執者は、開発・運用の現場エンジニアが執筆するIIJ公式ブログ「IIJ Engineers Blog」をはじめとするIIJのWebメディアでも今号の話題に関連する様々な記事を書いています。
是非こちらも併せてお読みください。

■1章 定期観測レポート　ブロードバンドトラフィックレポート〜2年目に入ったコロナ禍の影響〜
長 健二朗(ちょう けんじろう)の関連記事

- 新型コロナウイルスのフレッツトラフィックへの影響
- その後の新型コロナウイルスのフレッツトラフィックへの影響
- さらにその後の新型コロナウイルスのフレッツトラフィックへの影響

筆者の記事はこちらからご覧いただけます(https://eng-blog.iij.ad.jp/archives/author/kjc)。

■ 2章 フォーカス・リサーチ(1)Verifiable CredentialとBBS+署名
山本 暖(やまもと だん)の関連記事

- IIW (Internet Identity Workshop) の Verifiable Credential チケット
  (https://sect.iij.ad.jp/blog/2021/04/iiw-verifiable-credential/)

■3章 フォーカス・リサーチ(2)HaskellによるQUICの実装
山本 和彦(やまもと かずひこ)の関連記事

【絶賛連載中】

QUICをゆっくり解説 – 新しいインターネット通信規格

2021年5月にRFC9000 として仕様が公開された QUIC は、RFCを読みこなせばその全容が掴めますが、膨大かつ実際に実装してみて初めて腑に落ちることもあります。本連載では、実際にQUICを実装した経験を持つ筆者が、経験者目線でQUICを解説しています。

本連載の過去記事:

- QUICをゆっくり解説(1) :QUICが標準化されました
- QUICをゆっくり解説(2) :ネゴせよ
- QUICをゆっくり解説(3) :QUICパケットの構造
- QUICをゆっくり解説(4) :ハンドシェイク
- QUICをゆっくり解説(5) :2回目以降のハンドシェイクと0-RTT
- QUICをゆっくり解説(6) :増幅攻撃との戦い

連載記事はこちらからご覧いただけます(https://eng-blog.iij.ad.jp/quic)。

■ IIJ 公式Twitterアカウント@IIJ_ITS

なお、各媒体の記事公開やイベント開催情報は、IIJ公式Twitterアカウント@IIJ_ITSでお知らせしています。ご興味のある方はぜひフォローしてください。

@IIJ_ITS https://twitter.com/IIJ_ITS

September 2021 Vol.52

IIJ
Internet Initiative Japan

株式会社インターネットイニシアティブ(IIJ)について

IIJは、1992年、インターネットの研究開発活動に関わっていた技術者が中心となり、日本でインターネットを本格的に普及させようという構想を持って設立されました。
現在は、国内最大級のインターネットバックボーンを運用し、インターネットの基盤を担うと共に、官公庁や金融機関をはじめとしたハイエンドのビジネスユーザに、インターネット接続やシステムインテグレーション、アウトソーシングサービスなど、高品質なシステム環境をトータルに提供しています。
また、サービス開発やインターネットバックボーンの運用を通して蓄積した知見を積極的に発信し、社会基盤としてのインターネットの発展に尽力しています。

本冊子の情報は2021年9月時点のものです。

IIJ-MKTG019-0052

株式会社インターネットイニシアティブ
〒102−0071 東京都千代田区富士見2-10-2　飯田橋グラン・ブルーム
E-mail：info@iij.ad.jp　URL：https://www.iij.ad.jp