TTC技術レポート
Technical Report

TR-1111

QUIC に関する技術報告書

Technical Report on QUIC: A UDP-Based Multiplexed and Secure Transport

第 1 版

2026 年 3 月 16 日制定

一般社団法人
情報通信技術委員会

THE TELECOMMUNICATION TECHNOLOGY COMMITTEE

目　次

＜参考＞

１．国際勧告等との関連

本技術レポートは、RFC 9000 及び RFC 8999 を調査したものである。

２．上記国際勧告等に対する追加項目等

なし

３．改定の履歴

版　数	発　行　日	改　版　内　容
第 1 版	2026 年 3 月 16 日	制定

４．　参考文献

[QUIC-TRANSPORT] Iyengar, J., Ed. and M. Thomson, Ed., "QUIC: A UDP-Based Multiplexed and Secure Transport", RFC 9000, DOI 10.17487/RFC9000, May 2021, <https://www.rfc-editor.org/info/rfc9000>.

[QUIC-INVARIANTS] Thomson, M., "Version-Independent Properties of QUIC", RFC 8999, DOI 10.17487/RFC8999, May 2021, <https://www.rfc-editor.org/info/rfc8999>.

５．工業所有権

本技術レポートに関わる「工業所有権等の実施許諾に係る声明書」の提出状況は、TTC ホームページで閲覧可能である。

６．技術レポート作成部門

企業ネットワーク専門委員会

I.　本技術レポートの概要

QUIC は、従来インターネット通信にてトランスポート層プロトコルとして広く使用されている TCP に替わる次世代の通信プロトコルである。

低遅延と高速化、キャッシュ制御とプロトコル効率化、セキュリティの強化、多重化の改善といった特徴を持ち、現代のインターネットにおける「高速・安全・効率的な通信」を実現するための本質的な進化として位置付けられ、特に HTTP/3 との組み合わせで、ウェブ閲覧体験のさらなる向上に貢献している。

QUIC は Google が主導して開発し、IETF により 2021 年に RFC として策定された。

本報告書では IETF によって標準化された QUIC に関する以下の RFC について日本語に翻訳する。

- RFC 9000：QUIC:UDP ベースの多重化されたセキュアなトランスポート
- RFC 8999：QUIC のバージョンに依存しないプロパティ

II. RFC 9000 原文の著作権について

III. RFC 9000 の和訳

RFC 9000　QUIC:UDP ベースの多重化されたセキュアなトランスポート

要旨

この文書では、QUIC トランスポートプロトコルのコアを定義する。QUIC は、構造化通信、低遅延コネクションの確立、およびネットワークパスの移行のためのフロー制御ストリームをアプリケーションに提供する。QUIC には、展開環境の範囲で機密性、整合性、および可用性を確保するセキュリティ対策が含まれる。付属の文書では、鍵交換、ロス検知、および例示的な輻輳制御アルゴリズムのための TLS の統合について説明する。

1. 概要

QUIC は、セキュアな汎用トランスポートプロトコルである。この文書では、[QUIC-INVARIANTS] で定義されている QUIC のバージョンに依存しないプロパティに準拠する QUIC のバージョン 1 を定義する。

QUIC は、クライアントとサーバ間のステートフルな対話を作成するコネクション指向プロトコルである。

QUIC ハンドシェイクは、暗号化パラメータとトランスポートパラメータのネゴシエーションを組み合わせたものである。QUIC は TLS ハンドシェイク [TLS13] を統合するが、パケットを保護するためにカスタマイズされたフレームを使用する。TLS と QUIC の統合の詳細については、[QUIC-TLS] を参照。ハンドシェイクは、アプリケーションデータをできるだけ早く交換できるように構成されている。これには、クライアントがデータをすぐに送信するオプション (0-RTT) が含まれている。これを有効にするには、何らかの形式の事前通信または構成が必要である。

エンドポイントは、QUIC パケットを交換することによって QUIC で通信する。ほとんどのパケットには、エンドポイント間で制御情報とアプリケーションデータを伝送するフレームが含まれている。QUIC は各パケット全体を認証し、実用的な範囲で各パケットを暗号化する。QUIC パケットは、既存のシステムおよびネットワークへの展開を容易にするために、UDP データグラム [UDP] で伝送される。

アプリケーションプロトコルは、バイトの順序付きシーケンスであるストリームを介して QUIC コネクションで情報を交換する。両方のエンドポイントがデータを送信できる双方向ストリームと、単一のエンドポイントがデータを送信できる単一方向ストリームの 2 種類のストリームを作成できる。クレジットベースのスキームを使用して、ストリームの作成を制限し、送信できるデータの量を制限する。

QUIC は、信頼性の高い配信と輻輳制御を実装するために必要なフィードバックを提供する。データ損失を検出して回復するためのアルゴリズムについては、[QUIC-RECOVERY] のセクション 6 で説明されている。QUIC は、ネットワークの輻輳を回避するために輻輳制御に依存している。輻輳制御アルゴリズムの例については、[QUIC-RECOVERY] のセクション 7 で説明されている。

QUIC コネクションは、単一のネットワークパスに厳密にバインドされていない。コネクションの移行では、コネクション識別子を使用して、コネクションを新しいネットワークパスに転送できるようにする。このバージョンの QUIC では、クライアントのみが移行できる。この設計では、NAT の再バインドによって発生する可能性のあるネットワークトポロジまたはアドレスマッピングの変更後も、コネクションを継続できる。

コネクションが確立されると、コネクションの終了に複数のオプションが提供される。アプリケーションはグレースフルシャットダウンを管理でき、エンドポイントはタイムアウト期間をネゴシエートでき、エラーは即時コネクション切断を引き起こし、ステートレスメカニズムは 1 つのエンドポイントが状態を失った後のコネクションの終了を提供する。

1.1 文書の構造

この文書は、コア QUIC プロトコルについて説明し、次のように構成されている。

- ストリームは、QUIC が提供する基本的なサービス抽象化である。
  - 第 2 章では、ストリームに関連するコア概念について説明する。
  - 第 3 章では、ストリーム状態のリファレンスモデルについて説明する。
  - 第 4 章では、フロー制御の操作について概説する。
- コネクションは、QUIC エンドポイントが通信するコンテキストである。
  - 第 5 章では、コネクションに関連するコア概念について説明する。
  - 第 6 章では、バージョンネゴシエーションについて説明する。
  - 第 7 章では、コネクションを確立するためのプロセスについて説明する。
  - 第 8 章では、アドレス検証と重大なサービス拒否の軽減策について説明する。
  - 第 9 章では、エンドポイントが新しいネットワークパスにコネクションを移行する方法について説明する。
  - 第 10 章では、開いているコネクションを終了するためのオプションを示す。
  - 第 11 章では、ストリームおよびコネクションのエラー処理に関するガイダンスを示す。
- パケットとフレームは、QUIC が通信に使用する基本単位である。
  - 第 12 章では、パケットとフレームに関連する概念について説明する。
  - 第 13 章では、データの送信、再送信、および確認応答のモデルを定義する。
  - 第 14 章では、QUIC パケットを伝送するデータグラムのサイズを管理するためのルールを指定する。
- 最後に、QUIC プロトコル要素のエンコードの詳細について以下で説明する。
  - 第 15 章 (バージョン)
  - 第 16 章 (整数エンコーディング)
  - 第 17 章 (パケットヘッダ)
  - 第 18 章 (トランスポートパラメータ)
  - 第 19 章 (フレーム)
  - 第 20 章 (エラー)

付属の文書では、QUIC のロス検知と輻輳制御 [QUIC-RECOVERY]、および TLS とその他の暗号化メカニズム [QUIC-TLS] の使用について説明する。

この文書では、[QUIC-INVARIANTS] のプロトコルインバリアントに準拠する QUIC バージョン 1 を定義する。

QUIC バージョン 1 を参照するには、この文書を引用する。QUIC のバージョンに依存しないプロパティの限定されたセットへの参照は、[QUIC-INVARIANTS] を引用できる。

1.2 用語と定義

この文書内のキーワード[MUST]、[MUST NOT]、[REQUIRED]、[SHALL]、[SHALL NOT]、[SHOULD]、[SHOULD NOT]、[RECOMMENDED]、[NOT RECOMMENDED]、[MAY]、および[OPTIONAL]は、ここに示すようにすべて大文字で使用されている場合に限り、BCP 14 [RFC2119] [RFC8174] で説明されているとおりに解釈される。

この文書でよく使用される用語を次に示す。

QUIC: この文書で説明されているトランスポートプロトコル。QUIC は名前であり、略語ではない。

エンドポイント(Endpoint):QUIC パケットを生成、受信、および処理することによって QUIC コネクションに参加できるエンティティ。QUIC には、クライアントとサーバの 2 種類のエンドポイントのみがある。

クライアント(Client): QUIC コネクションを開始するエンドポイント。

サーバ (Server):　QUIC コネクションを受け入れるエンドポイント。

QUIC パケット (QUIC packet):　UDP データグラムにカプセル化できる QUIC の完全な処理可能単位。1 つ以上の QUIC パケットを 1 つの UDP データグラムにカプセル化できる。

確認要求パケット (Ack-eliciting packet): ACK、PADDING、および CONNECTION_CLOSE 以外のフレームを含む QUIC パケット。これらにより、受信側は確認応答を送信する。第 13.2.1 項を参照。

フレーム (Frame):　構造化プロトコル情報の単位。複数のフレームタイプがあり、それぞれが異なる情報を伝送する。フレームは QUIC パケットに含まれる。

アドレス (Address): 修飾なしで使用する場合、ネットワークパスの一方の端を表す IP バージョン、IP アドレス、および UDP ポート番号のタプル。

コネクション ID (Connection ID):エンドポイントで QUIC コネクションを識別するために使用される識別子。各エンドポイントは、ピアエンドポイントに送信されるパケットに含める 1 つ以上のコネクション ID を選択する。この値はピアエンドポイントに対して非透過的である。

ストリーム (Stream):　QUIC コネクション内の順序付けられたバイトの単方向または双方向チャネル。QUIC コネクションでは、複数のストリームを同時に伝送できる。

アプリケーション (Application):　QUIC を使用してデータを送受信するエンティティ。

この文書では、「QUIC パケット」、「UDP データグラム」、および「IP パケット」という用語を使用して、それぞれのプロトコルの単位を示す。すなわち、1 つ以上の QUIC パケットを UDP データグラムにカプセル化し、それを IP パケットにカプセル化できる。

1.3.　表記規則

この文書のパケットおよびフレーム図では、カスタム形式を使用する。この形式の目的は、プロトコル要素を定義するのではなく、要約することである。構文は、構造体の完全なセマンティクスと詳細を定義する。

複合フィールドには名前が付けられ、その後に対応する中かっこで囲まれたフィールドのリストが続く。このリストの各フィールドはコンマで区切られる。

個々のフィールドには、長さの情報に加えて、固定値、オプション、または繰り返しに関する指示が含まれる。個々のフィールドは以下の表記規則を使用する。長さはすべてビット単位である。

x (A):　x が A ビット長であることを示す。

x (i):　第 16 章で説明されている可変長符号化を使用して、x が整数値を保持することを示す。

x (A..B):　x が A から B までの任意の長さであることを示す。A を省略すると最小 0 ビットを示し、B を省略すると上限を設定しないことを示す。この形式の値は常にバイト境界で終了する。

x (L) =C:　x の固定値が C であることを示す。x の長さは L で表され、上記の長さ形式のいずれかを使用できる。

x (L) =C..D:　x が C から D までの範囲の値を持ち、上記のように L で表される長さを持つことを示す。

[x (L)]:　x がオプションであり、長さが L であることを示す。

x (L) ...:　x が 0 回以上繰り返され、各インスタンスの長さが L であることを示す。

この文書では、ネットワークバイトオーダー(すなわち、ビッグエンディアン)値を使用する。フィールドは、各バイトの上位ビットから順に配置される。

慣例により、個々のフィールドは複合フィールドの名前を使用して複合フィールドを参照する。

図 1 に例を示す。

```
Example Structure {
  One-bit Field (1),
  7-bit Field with Fixed Value (7) = 61,
  Field with Variable-Length Integer (i),
  Arbitrary-Length Field (..),
  Variable-Length Field (8..24),
  Field With Minimum Length (16..),
  Field With Maximum Length (..128),
  [Optional Field (64)],
  Repeated Field (8) ...,
}
```

図 1　形式の例

単一ビットフィールドを文中で参照する場合、フィールドの値セットを持つフィールドを保持するバイトの値を使用して、そのフィールドの位置を明確にすることができる。たとえば、値 0x80 を使用して、バイトの最上位ビットの単一ビットフィールドを参照できる(図 1 の 1 ビットフィールドなど)。

2.　ストリーム

QUIC のストリームは、アプリケーションに軽量で順序付けられたバイトストリームの抽象化を提供する。ストリームは、単方向または双方向にすることができる。

ストリームは、データを送信することによって作成できる。ストリーム管理に関連するその他のプロセス(終了、キャンセル、フロー制御の管理) はすべて、最小限のオーバーヘッドを課すように設計されている。

たとえば、1 つの STREAM フレーム(第 19.8 節)で、ストリームのオープン、データの伝送、およびストリームのクローズを行うことができる。

また、ストリームの存続期間を長くすることもでき、コネクション期間全体を継続することができる。

ストリームは、どちらのエンドポイントでも作成でき、他のストリームと交互配置されたデータを同時に送信でき、キャンセルすることができる。QUIC は、異なるストリーム上のバイト間の順序を保証する手段を提供しない。

QUIC を使用すると、任意の数のストリームが同時に動作し、任意のストリームで任意の量のデータを送信できる。ただし、フロー制御の制約とストリームの制限に従う（第 4 章を参照）。

2.1　ストリームの種類と識別子

ストリームには、単方向または双方向がある。単方向ストリームは、ストリームの発信側から着信側への一方向でデータを伝送する。双方向ストリームでは、双方向にデータを送信できる。

ストリームは、コネクション内でストリーム ID と呼ばれる数値によって識別される。ストリーム ID は、コネクション上のすべてのストリームで一意の 62 ビット整数 (0 から 2^{62}-1) である。ストリーム ID は可変長整数として符号化される（第 16 章を参照）。QUIC エンドポイントは、コネクション内でストリーム ID を再利用してはならない[MUST NOT]。

ストリーム ID の最下位ビット (0x01) は、ストリームの発信側を識別する。クライアントから開始されたストリームには偶数のストリーム ID (ビットは 0 を設定) があり、サーバから開始されたストリームには奇数のストリーム ID (ビットは 1 を設定) がある。

ストリーム ID の 2 番目の最下位ビット (0x02) は、双方向ストリーム (ビットは 0 を設定) と単一方向ストリーム (ビットは 1 を設定) を区別する。

したがって、ストリーム ID の 2 つの最下位ビットは、表 1 にまとめられているように、ストリームを 4 つの種類のいずれかとして識別する。

表 1　ストリーム ID タイプ

ビット	ストリームの種類
0x00	クライアント開始、双方向
0x01	サーバ開始、双方向
0x02	クライアント開始、単方向
0x03	サーバ開始、単方向

各タイプのストリームスペースは最小値(それぞれ 0x00 から 0x03)から始まる。各タイプの連続するストリームは、数値が増加するストリーム ID で作成される。ストリーム ID が順不同で使用されると、同じタイプで、それよりストリーム ID が小さいすべてのストリームが開かれる。

2.2　データの送受信

ストリームフレーム(第 19.8 節)は、アプリケーションによって送信されたデータをカプセル化する。エンドポイントは、ストリームフレームのストリーム ID フィールドとオフセットフィールドを使用して、データを順序どおりに配置する。

エンドポイントは、ストリームデータを順序付けられたバイトストリームとしてアプリケーションに配信できる必要がある[MUST]。順序付けられたバイトストリームを配信するには、エンドポイントが通知されたフロー制御制限まで、順序どおりに受信されたデータをバッファする必要がある。

QUIC では、順序どおりにストリームデータを配信するための特定の許容量はない。ただし、実装では、受信側アプリケーションに順不同でデータを配信する能力を提供することも選択できる[MAY]。

エンドポイントは、同じストリームオフセットでストリームのデータを複数回受信できる。受信済みのデータは破棄できる。指定されたオフセットのデータは、複数回送信された場合に変更してはならない[MUST NOT]。エンドポイントは、ストリーム内の同じオフセットで異なるデータを受信した場合に、PROTOCOL_VIOLATION タイプのコネクションエラーとして処理しても構わない[MAY]。

ストリームは、QUIC から参照できる他の構造を持たない、順序付けられたバイトストリームの抽象化である。データが送信されるとき、あるいはパケット損失後に再送信されるとき、または受信側でアプリケーションに配信されるときに、ストリームフレームの境界が保持されることは想定されていない。

エンドポイントは、ピアによって設定されたフロー制御制限内にあることを確認せずに、ストリームでデータを送信してはならない[MUST NOT]。フロー制御については、第 4 章で詳しく説明する。

2.3　ストリームの優先順位付け

ストリームに割り当てられたリソースが正しく優先順位付けされている場合、ストリーム多重化はアプリケーションのパフォーマンスに大きな影響を与える可能性がある。

QUIC は、優先順位付け情報を交換するためのメカニズムを提供しない。代わりに、アプリケーションからの優先順位情報の受信に依存する。

QUIC の実装は、アプリケーションがストリームの相対的な優先順位を示す方法を提供する必要がある[SHOULD]。実装では、アプリケーションによって提供される情報を使用して、アクティブなストリームにリソースを割り当てる方法を決定する。

2.4　ストリームに対する操作

この文書では、QUIC の API を定義しない。代わりに、アプリケーションプロトコルが依存できるストリームに対する一連の関数を定義する。アプリケーションプロトコルは、QUIC 実装がこの節で説明されている操作を含むインタフェースを提供すると想定できる。特定のアプリケーションプロトコルで使用するように設計された実装は、そのプロトコルで使用される操作のみを提供する場合がある。

ストリームの送信部では、アプリケーションプロトコルは次のことを実行できる。

- データの書き込み、書き込まれたデータを送信するためにストリームフロー制御クレジット(第 4.1 節)が正常に予約されたことを理解する。
- ストリームを終了 (クリーンな終了) し、FIN ビットが設定された STREAM フレーム(第 19.8 節)を生成する。
- ストリームをリセット (突然の終了) し、ストリームがまだ終了状態になっていない場合は RESET_STREAM フレーム(第 19.4 節)を生成する。

ストリームの受信側では、アプリケーションプロトコルは次のことを実行できる。

- データの読み取り。
- ストリームの読み取りを中止して終了を要求する。これにより、STOP_SENDING フレームが発生する可能性がある(第 19.5 節)。

アプリケーションプロトコルは、ピアがストリームを開いたときやリセットしたとき、ピアがストリームの読み取りを中止したとき、新しいデータが使用可能になったとき、フロー制御のためにデータをストリームに書き込むことができるかどうかなど、ストリームの状態の変化を通知するように要求することもできる。

３．　ストリームの状態

この章では、送信または受信コンポーネントの観点からストリームについて説明する。エンドポイントがデータを送信するストリーム(第 3.1 節)と、エンドポイントがデータを受信するストリーム(第 3.2 節)の 2 つのステートマシンについて説明する。

単方向ストリームでは、ストリームの種類とエンドポイントの役割に応じて、送信側または受信側のステートマシンが使用される。双方向ストリームでは、両方のエンドポイントで両方のステートマシンが使用される。ほとんどの場合、これらのステートマシンの使用方法は、ストリームが単方向でも双方向でも同じである。双方向ストリームでは、送信側または受信側のいずれかが開くとストリームが双方向に開かれるため、ストリームを開くための条件が少し複雑になる。

この章で示すステートマシンは大いに有益である。この文書では、ストリームの状態を使用して、さまざまな種類のフレームをいつどのように送信できるか、およびさまざまな種類のフレームを受信したときに予想される反応についての規則を説明する。これらのステートマシンは QUIC の実装に役立つことを目的としているが、これらの状態は実装を制約するものではない。実装は、その動作がこれらの状態の実装と一致している限り、異なるステートマシンを定義できる。

注意:場合によっては、1 つのイベントまたはアクションによって複数の状態が遷移することがある。たとえば、FIN ビットが設定された STREAM を送信すると、"Ready"状態から"Send"状態、および"Send"状態から"Data Sent"状態と、送信ストリームに対して 2 つの状態遷移が発生する可能性がある。

3.1　送信ストリームの状態

図 2 は、ピアにデータを送信するストリームの一部の状態を示している。

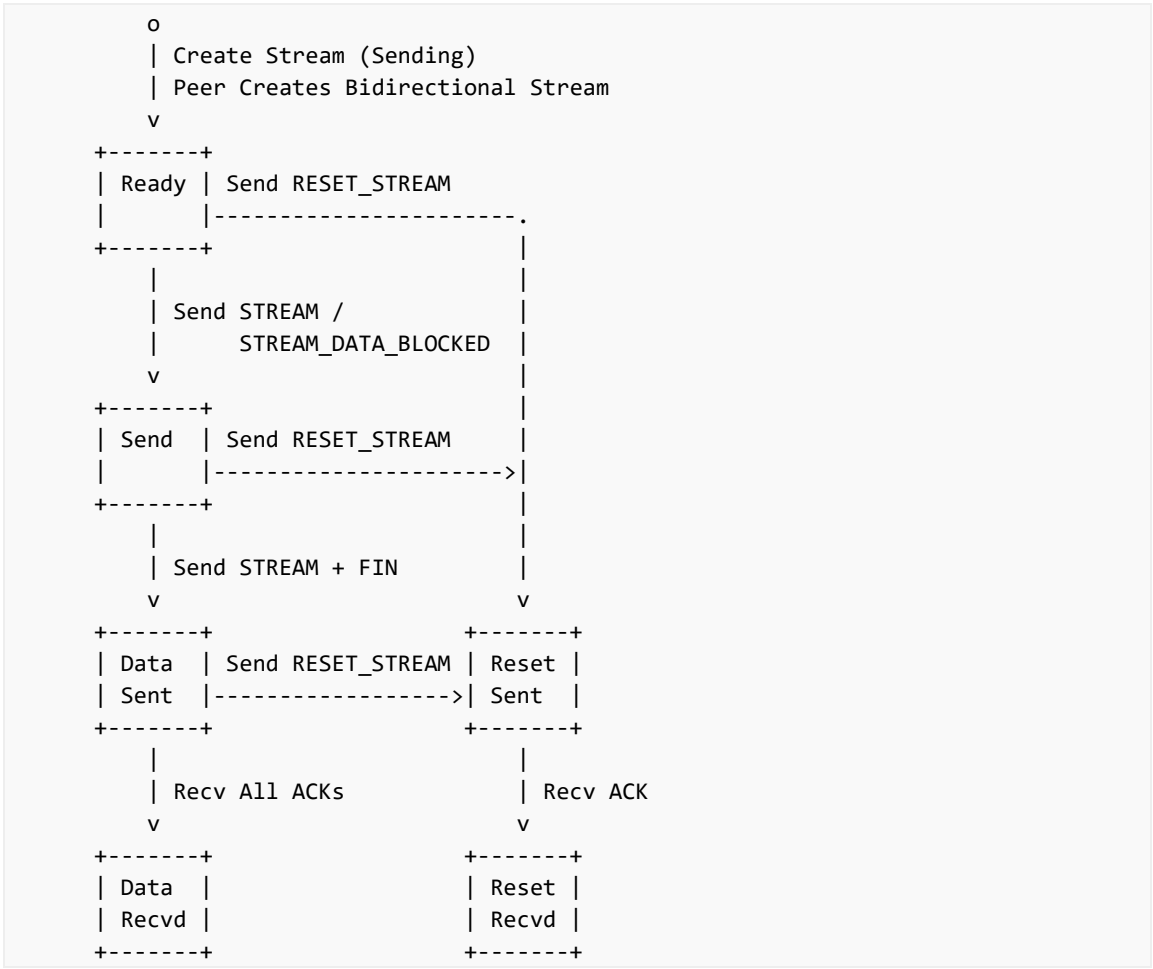

図 2　ストリームの送信部の状態

エンドポイントが開始するストリームの送信部(クライアントのタイプ 0 と 2、サーバのタイプ 1 と 3)は、アプリケーションによって開かれる。"Ready"状態は、アプリケーションからデータを受け取ることができる新しく作成されたストリームを表す。ストリームデータは、送信の準備としてこの状態でバッファに格納される場合がある。

最初の STREAM フレームまたは STREAM_DATA_BLOCKED フレームを送信すると、ストリームの送信部が"Send"状態になる。実装では、最初の STREAM フレームを送信してこの状態になるまで、ストリームへのストリーム ID の割り当てを延期することを選択できる。これにより、ストリームの優先順位付けを向上させることができる。

ピア(サーバの場合は 0、クライアントの場合は 1)によって開始された双方向ストリームの送信部は、受信部が作成されると"Ready"状態で開始される。

"Send"状態では、エンドポイントは STREAM フレーム内のストリームデータを送信し、必要に応じて再送信する。エンドポイントは、ピアによって設定されたフロー制御制限を順守し、MAX_STREAM_DATA フレームの受け入れと処理を続行する。"Send"状態のエンドポイントは、ストリームフロー制御制限(第 4.1 節)によって送信がブロックされている場合に STREAM_DATA_BLOCKED フレームを生成する。

すべてのストリームデータが送信され、FIN ビットを含む STREAM フレームが送信されたことをアプリケーションが示すと、ストリームの送信部は"Data Sent"状態になる。この状態から、エンドポイントは必要に応じてストリームデータのみを再送信する。エンドポイントは、この状態のストリームに対して、フロー制御制限を確認したり、STREAM_DATA_BLOCKED フレームを送信したりする必要はない。MAX_STREAM_DATA フレームは、ピアが最終的なストリームオフセットを受信するまで受信される可能性がある。エンドポイントは、この状態のストリームに対してピアから受信した MAX_STREAM_DATA フ

レームを安全に無視できる。

すべてのストリームデータが正常に受信確認されると、ストリームの送信部は、終端状態である"Data Recvd"状態になる。

アプリケーションは、"Ready"状態、"Send"状態、または"Data Sent"状態のいずれかの状態から、ストリームデータの転送を中止することを通知できる。または、エンドポイントがピアから STOP_SENDING フレームを受信することもある。どちらの場合も、エンドポイントは RESET_STREAM フレームを送信する。これにより、ストリームは"Reset Sent"状態になる。

エンドポイントは、ストリームに言及する最初のフレームとして RESET_STREAM を送信することがある[MAY]。これにより、そのストリームの送信部が開き、すぐに"Reset Sent"状態に遷移する。

RESET_STREAM を含むパケットが受信確認されると、ストリームの送信部は、終端状態である"Reset Recvd"状態になる。

3.2 受信ストリームの状態

図 3 は、ピアからデータを受信するストリームの部分の状態を示している。ストリームの受信部の状態は、ピアでのストリームの送信部の状態の一部のみを反映する。ストリームの受信部は、"Ready"状態など、監視できない送信部の状態は追跡しない。代わりに、ストリームの受信部は、アプリケーションへのデータの配信を追跡する。その一部は、送信側が監視することはできない。

図 3　ストリームの受信部の状態

ピア(クライアントの場合はタイプ 1 とタイプ 3、サーバの場合はタイプ 0 とタイプ 2)によって開始されたストリームの受信部は、そのストリームの最初の STREAM、STREAM_DATA_BLOCKED、または

RESET_STREAM フレームを受信したときに作成される。ピアによって開始された双方向ストリームの場合、ストリームの送信部の MAX_STREAM_DATA または STOP_SENDING フレームを受信すると、受信部も作成される。ストリームの受信部の初期状態は"Recv"状態である。

双方向ストリームの場合、エンドポイント(クライアントの場合は 0、サーバの場合は 1)によって開始された送信部が"Ready"状態になると、受信部は"Recv"状態になる。

エンドポイントは、そのストリームのピアから MAX_STREAM_DATA または STOP_SENDING フレームを受信すると、双方向ストリームを開く。開いていないストリームに対して MAX_STREAM_DATA フレームを受信すると、リモートのピアがストリームを開いていて、フロー制御クレジットを提供していることを示す。開いていないストリームに対して STOP_SENDING フレームを受信すると、リモートのピアがこのストリームでデータを受信する必要がなくなったことを示す。パケットが失われたり、並べ替えられたりした場合、いずれかのフレームが STREAM または STREAM_DATA_BLOCKED フレームより前に到着する可能性がある。

ストリームを作成する前に、同じ種類でストリーム ID の番号が小さいすべてのストリームを作成する必要がある[MUST]。これにより、両方のエンドポイントでストリームの作成順序が一貫していることが保証される。

"Recv"状態では、エンドポイントは STREAM および STREAM_DATA_BLOCKED フレームを受信する。受信データはバッファに格納され、アプリケーションに配信するために正しい順序に再構成できる。アプリケーションによってデータが消費され、バッファ領域が使用可能になると、エンドポイントは MAX_STREAM_DATA フレームを送信して、ピアがより多くのデータを送信できるようにする。

FIN ビットを含む STREAM フレームを受信すると、ストリームの最終サイズが判明する（第 4.5 節を参照）。その後、ストリームの受信部は"Size Known"状態になる。この状態では、エンドポイントは MAX_STREAM_DATA フレームを送信する必要がなくなり、ストリームデータの再送信のみを受信する。

ストリームのすべてのデータが受信されると、受信部は"Data Recvd"状態になる。これは、"Size Known"状態に遷移する原因となる同じ STREAM フレームを受信した結果として発生する可能性がある。すべてのデータを受信した後、ストリームの STREAM または STREAM_DATA_BLOCKED フレームは破棄できる。

“Data Recvd"状態は、ストリームデータがアプリケーションに配信されるまで保持される。ストリームデータが配信されると、ストリームは"Data Read"状態になる。

"Recv"状態または"Size Known"状態の RESET_STREAM フレームを受信すると、ストリームは"Reset Recvd"状態になる。これにより、アプリケーションへのストリームデータの配信が中断される可能性がある。

RESET_STREAM を受信したときに、すべてのストリームデータが既に受信されている可能性がある(すなわち、”Data Recvd”状態)。同様に、RESET_STREAM フレームを受信した後に残りのストリームデータが到着する可能性がある ("Reset Recvd"状態) 。実装では、この状況を自由に管理できる。

RESET_STREAM の送信は、エンドポイントがストリームデータの配信を保証できないことを意味する。ただし、RESET_STREAM を受信した場合にストリームデータが配信されないという要件はない。実装は、ストリームデータの配信を中断し、使用されなかったデータを破棄し、RESET_STREAM の受信を通知する場合がある[MAY]。ストリームデータが完全に受信され、アプリケーションによって読み取られるようにバッファされている場合、RESET_STREAM シグナルは抑制または保留されることがある。RESET_STREAM が抑制された場合、ストリームの受信部は"Data Recvd"状態のままになる。

ストリームがリセットされたことを示すシグナルをアプリケーションが受信すると、ストリームの受信部は、終端状態である"Reset Read"状態に移行する。

3.3 許可されるフレームの種類

ストリームの送信側は、送信側または受信側のストリームの状態に影響を与えるフレームの種類として、

STREAM (第 19.8 節)、STREAM_DATA_BLOCKED (第 19.13 節)、および RESET_STREAM (第 19.4 節)の 3 種類のみを送信する。

送信側は、これらのフレームを終端状態 ("Data Recvd"状態または"Reset Recvd"状態) から送信しては ならない[MUST NOT]。送信側は、"Reset Sent"状態または任意の終端状態のストリームに対して、すなわち、RESET_STREAM フレームを送信した後、STREAM または STREAM_DATA_BLOCKED フレームを送信してはならない[MUST NOT]。これら 3 種類のフレームを伝送するパケットの配信は遅延する可能性があるため、受信側はこれらのフレームを任意の状態で受信できる。

ストリームの受信側は、MAX_STREAM_DATA フレーム (第 19.10 節) と STOP_SENDING フレーム(第 19.5 節)を送信する。

受信側は、"Recv"状態の MAX_STREAM_DATA フレームのみを送信する。受信側は、RESET_STREAM フレームを受信していない任意の状態、つまり"Reset Recvd"状態や"Reset Read"状態以外の状態でも STOP_SENDING フレームを送信することができる[MAY]。ただし、すべてのストリームデータが受信されているため、STOP_SENDING フレームを"Data Recvd"状態で送信することにはほとんど意味がない。送信側は、パケットの遅延配信の結果として、これら 2 種類のフレームのいずれかを任意の状態で受信できる。

3.4 双方向ストリームの状態

双方向ストリームは、送信部と受信部で構成される。実装では、双方向ストリームの状態を送信ストリームの状態と受信ストリームの状態の複合として表すことができる。最も単純なモデルでは、送信部と受信部のいずれかが非終端状態の場合はストリームが"open"状態として表示され、送信ストリームと受信ストリームの両方が終端状態の場合は"closed"状態として表示される。

表 2 は、HTTP/2 [HTTP2] で定義されているストリームの状態に大まかに対応する双方向ストリームの状態のより複雑なマッピングを示している。これは、ストリームの送信部または受信部の複数の状態が同じ複合状態にマッピングされていることを示している。これはマッピングの 1 つの可能性にすぎないことに注意すること。このマッピングでは、”closed”状態または”half-closed”状態に移行する前にデータが確認される必要がある。

表 2　ストリームの状態から HTTP/2 へ可能なマッピング

送信部	受信部	複合状態
No Stream / Ready	No Stream / Recv (*1)	idle
Ready / Send / Data Sent	Recv / Size Known	open
Ready / Send / Data Sent	Data Recvd / Data Read	half-closed (remote)
Ready / Send / Data Sent	Reset Recvd / Reset Read	half-closed (remote)
Data Recvd	Recv / Size Known	half-closed (local)
Reset Sent / Reset Recvd	Recv / Size Known	half-closed (local)
Reset Sent / Reset Recvd	Data Recvd / Data Read	closed
Reset Sent / Reset Recvd	Reset Recvd / Reset Read	closed
Data Recvd	Data Recvd / Data Read	closed
Data Recvd	Reset Recvd / Reset Read	closed

注意 (*1) :ストリームがまだ作成されていない場合、またはストリームの受信部がまだフレームを受信せずに"Recv"状態にある場合、ストリームは"idle"状態と見なされる。

3.5　要請された状態遷移

アプリケーションがストリームで受信しているデータに関心がなくなった場合は、ストリームの読み取りを中止し、アプリケーションエラーコードを指定できる。

ストリームが"Recv"状態または"Size Known"状態の場合、トランスポートは STOP_SENDING フレームを送信して反対方向のストリームの終了を促すことでこれを通知する必要がある[SHOULD]。これは通常、受信側アプリケーションがストリームから受信したデータを読み取らなくなったことを示すが、受信データが無視される保証はない。

STOP_SENDING フレームを送信した後に受信した STREAM フレームは、受信時に破棄できる場合でも、コネクションおよびストリームフロー制御に対してカウントされる。

STOP_SENDING フレームは、受信側エンドポイントが RESET_STREAM フレームを送信することを要求する。ストリームが"Ready"状態または"Send"状態の場合、STOP_SENDING フレームを受信するエンドポイントは RESET_STREAM フレームを送信する必要がある[MUST]。ストリームが"Data Sent"状態の場合、エンドポイントは、未処理のデータを含むパケットが確認されるか、失われたと宣言されるまで、RESET_STREAM フレームの送信を延期することができる[MAY]。未処理のデータが失われたと宣言された場合、エンドポイントはデータを再送信するのではなく、RESET_STREAM フレームを送信する必要がある[SHOULD]。

エンドポイントは、STOP_SENDING フレームから送信する RESET_STREAM フレームにエラーコードをコピーする必要があり、任意のアプリケーションエラーコードを使用できる[SHOULD]。

STOP_SENDING フレームを送信するエンドポイントは、その後にそのストリームに対して受信した RESET_STREAM フレームのエラーコードを無視する可能性がある[MAY]。

STOP_SENDING フレームは、ピアによってリセットされていないストリームに対してのみ送信する必要がある。STOP_SENDING フレームは、"Recv"状態または"Size Known"状態のストリームに最も役立つ。

以前の STOP_SENDING フレームを含むパケットが失われた場合、エンドポイントは別の STOP_SENDING フレームを送信する必要がある。ただし、ストリームのすべてのストリームデータまたは RESET_STREAM フレームが受信されると (すなわち、ストリームが"Recv"状態または"Size Known"状態以外の状態にある場合)、STOP_SENDING フレームを送信する必要はない。

双方向ストリームの両方向を終了する必要があるエンドポイントは、RESET_STREAM フレームを送信することによって一方の方向を終了できる。また、STOP_SENDING フレームを送信することによって、反対方向の終了を促すことができる。

4.　フロー制御

受信側は、送信側が受信側よりはるかに高速なため過負荷になる、あるいは悪意のある送信者(送信側)が大量のメモリを消費するのを防ぐために、バッファとして要求するデータの量を制限する必要がある。受信側がコネクションのメモリコミットメントを制限できるようにするために、ストリームは個別に、またはコネクション全体にわたってフロー制御される。第 4.1 節および第 4.2 節で説明されているように、QUIC 受信側は、送信側がストリーム上およびすべてのストリームで送信できるデータの最大量を常に制御する。

同様に、コネクション内の同時実行を制限するために、第 4.6 節で説明されているように、QUIC エンドポイントはピアが開始できるストリームの最大累積数を制御する。

CRYPTO フレームで送信されるデータは、ストリームデータと同じ方法ではフロー制御されない。QUIC では、データの過剰なバッファリングの回避は暗号プロトコル実装に依存している([QUIC-TLS] を参照。)。

複数のレイヤでの過剰なバッファリングを回避するために、QUIC は、暗号プロトコルがバッファリング制限した通信を行うためのインタフェースを提供する必要がある[SHOULD]。

4.1 データフロー制御

QUIC は、受信側が特定のストリームまたはコネクション全体で受信する準備ができている合計バイト数の制限を通知する制限ベースのフロー制御スキームを採用している。これにより、QUIC では 2 つのレベルのデータフロー制御が行われる。

- ストリームフロー制御。各ストリームで送信できるデータの量を制限することで、1 つのストリームがコネクションの全受信バッファを消費しないようにする。
- コネクションフロー制御。すべてのストリームの STREAM フレームで送信されるストリームデータの合計バイト数を制限することで、送信側がコネクションの受信側のバッファ容量を超えないようにする。

送信側は、いずれの制限を超えてデータを送信してはならない[MUST NOT]。

受信側は、ハンドシェイク中にトランスポートパラメータを使用してすべてのストリームの初期制限を設定する(第 7.4 節)。その後、受信側は MAX_STREAM_DATA フレーム (第 19.10 節) または MAX_DATA フレーム(第 19.9 節)を送信側に送信して、より大きな制限を通知する。

受信側は、対応するストリーム ID を持つ MAX_STREAM_DATA フレームを送信することによって、ストリームのより大きな制限を通知できる。MAX_STREAM_DATA フレームは、ストリームの明確な最大バイトオフセットを示す。受信側は、そのストリームで消費されるデータの現在のオフセットに基づいて、通知するフロー制御オフセットを決定できる。

受信側は、すべてのストリームの明確なバイトオフセットの合計の最大値を示す MAX_DATA フレームを送信することによって、コネクションのより大きな制限を通知できる。受信側は、すべてのストリームで受信したバイトの累積合計を保持する。これは、通知されたコネクションまたはストリームのデータ制限の違反をチェックするために使用される。受信側は、すべてのストリームで消費されたバイトの合計に基づいて、通知する最大データ制限を決定できる。

受信側がコネクションまたはストリームの制限を通知すると、それより小さい制限を通知してもエラーにはならないが、その制限は効果がない。

送信側が通知されたコネクションまたはストリームのデータ制限に違反した場合、受信側は FLOW_CONTROL_ERROR タイプのエラーでコネクションを閉じなければならない[MUST]。エラー処理の詳細については、第 11 章を参照。

送信側は、フロー制御の制限を増やさない MAX_STREAM_DATA または MAX_DATA フレームを無視しならない[MUST]。

送信側が制限を超えてデータを送信した場合、新しいデータを送信できず、ブロックされていると見なされる。送信側は、書き込むデータがあるが、フロー制御の制限によってブロックされていることを受信側に示すために、STREAM_DATA_BLOCKED または DATA_BLOCKED フレームを送信する必要がある。送信側がアイドルタイムアウト(第 10.1 節)より長い期間ブロックされている場合、送信側に送信可能なデータがある場合でも、受信側はコネクションを閉じる可能性がある。コネクションが閉じないようにするには、フロー制御が制限されている送信側は、送信中の ack-eliciting パケットがない場合に、定期的に STREAM_DATA_BLOCKED または DATA_BLOCKED フレームを送信する必要がある。

4.2 フロー制御制限の増加

実装では、MAX_STREAM_DATA および MAX_DATA フレームで通知するクレジットのタイミングと量を決定するが、この節ではいくつかの考慮事項を示す。

送信側のブロックを回避するために、受信側はラウンドトリップ内で MAX_STREAM_DATA または MAX_DATA フレームを複数回送信するか、フレームの損失とその後の回復のための時間を確保するために

十分に早く送信してもよい[MAY]。

制御フレームは、コネクションのオーバーヘッドに寄与する。したがって、小さな変更でMAX_STREAM_DATA および MAX_DATA フレームを頻繁に送信することは望ましくない。一方、更新の頻度が低い場合は、送信側のブロックを回避するために制限の増分を大きくする必要があり、受信側でより大きなリソースコミットメントが必要になる。通知される制限の大きさを決定する際には、リソースコミットメントとオーバーヘッドの間にトレードオフがある。

受信側は、自動チューニングメカニズムを使用して、一般的な TCP 実装と同様に、ラウンドトリップ時間の推定値と、受信側アプリケーションがデータを消費するレートに基づいて、通知された追加クレジットの頻度と量を調整できる。最適化として、エンドポイントは、送信する他のフレームがある場合にのみフロー制御に関連するフレームを送信し、フロー制御によって余分なパケットが送信されないようにすることができる。

ブロックされた送信側は、STREAM_DATA_BLOCKED または DATA_BLOCKED フレームを送信する必要はない。したがって、受信側は、MAX_STREAM_DATA または MAX_DATA フレームを送信する前に、STREAM_DATA_BLOCKED または DATA_BLOCKED フレームを待機してはならない[MUST NOT]。これを行うと、送信側がコネクションの残りの部分でブロックされる可能性がある。送信側がこれらのフレームを送信した場合でも、それらを待機すると、少なくともラウンドトリップ全体で送信側がブロックされる。

送信側がブロックされた後にクレジットを受信すると、応答で大量のデータを送信できる可能性があり、その結果、短期的な輻輳が発生する。送信側がこの輻輳を回避する方法については、[QUIC-RECOVERY] の第 7.7 節を参照。

4.3 フロー制御のパフォーマンス

エンドポイントが、コネクションでピアの帯域幅遅積よりも大きい使用可能なフロー制御クレジットをピアが常に持っていることを保証できない場合、その受信スループットはフロー制御によって制限される。

パケット損失によって受信バッファにギャップが発生し、アプリケーションがデータを消費して受信バッファ領域を解放できなくなる可能性がある。

フロー制御制限のタイムリーな更新を送信すると、パフォーマンスが向上する。フロー制御の更新を提供するためだけにパケットを送信すると、ネットワークの負荷が増加し、パフォーマンスに悪影響を及ぼす可能性がある。ACK フレームなどの他のフレームと共にフロー制御の更新を送信すると、これらの更新のコストが削減される。

4.4 ストリームのキャンセルの処理

エンドポイントは、コネクションレベルのフロー制御のすべてのバイトを考慮できるようにするために、すべてのストリームで消費されたフロー制御クレジットの量について最終的に合意する必要がある。

RESET_STREAM フレームを受信すると、エンドポイントは一致するストリームの状態を破棄し、そのストリームに到着するそれ以降のデータを無視する。

RESET_STREAM フレームは、ストリームの一方向を強制終了する。双方向ストリームの場合、RESET_STREAM フレームは反対方向のデータフローには影響しない。両方のエンドポイントは、その方向が終了状態になるまで、終了していない方向のストリームのフロー制御状態を維持する必要がある[MUST]。

4.5 ストリームの最終的なサイズ

最終的なサイズは、ストリームによって消費されるフロー制御クレジットの量である。ストリーム上の連続するすべてのバイトが 1 回送信されたと仮定すると、最終的なサイズは送信されたバイト数になる。一般に、これはストリームで送信されたオフセットが最大のバイトのオフセットよりも 1 大きい値になる。バイトが送信されなかった場合は 0 になる。

送信側は、ストリームの終了方法に関係なく、常にストリームの最終的なサイズを受信側に確実に伝達する。最終的なサイズは、FIN フラグを持つ STREAM フレームの”Offset”フィールドと”Length”フィールドの合計である。これらのフィールドは暗黙的な場合があることに注意すること。代わりに、RESET_STREAM フレームの”Final Size”フィールドにこの値が格納される。これにより、そのストリームで送信側が消費したフロー制御クレジットの量について、両方のエンドポイントが同意することが保証される。

ストリームの受信部が"Size Known"状態または"Reset Recvd"状態になると、エンドポイントはストリームの最終的なサイズを認識する(第 3 章) 。受信側は、コネクションレベルのフローコントローラのストリームで送信されたすべてのバイトを考慮するために、ストリームの最終的なサイズを使用する必要がある[MUST]。

エンドポイントは、最終的なサイズより大きいストリームでデータを送信してはならない[MUST NOT]。

ストリームの最終的なサイズがわかったら、変更することはできない。ストリームの最終サイズの変更を示す RESET_STREAM または STREAM フレームを受信した場合､エンドポイントは FINAL_SIZE_ERROR タイプのエラーで応答する必要がある[SHOULD]。エラー処理の詳細については、第 11 章を参照。受信側は、ストリームが閉じられた後でも、最終サイズ以上のデータの受信を FINAL_SIZE_ERROR タイプのエラーとして扱う必要がある[SHOULD]。これらのエラーの生成は必須ではない。エンドポイントがこれらのエラーを生成することを要求することは、エンドポイントが閉じられたストリームの最終サイズの状態を維持する必要があることを意味する。これは、重大なステートコミットメントを意味する可能性がある。

4.6 同時実行の制御

エンドポイントは、ピアが開くことができる受信ストリームの累積数を制限する。ストリーム ID が (max_streams*4+first_stream_id_of_type) 未満のストリームのみを開くことができる（表 1 を参照）。最初の制限は、トランスポートパラメータで設定される（18.2 節を参照）。その後の制限は、MAX_STREAMS フレームを使用して通知される（第 19.11 節を参照）。単方向ストリームと双方向ストリームには異なる制限が適用される。

max_streams トランスポートパラメータまたは MAX_STREAMS フレームが 2^{60} より大きい値で受信された場合、可変長整数として表現できない最大ストリーム ID が許可される（第 16 章を参照）。いずれかを受信した場合､問題の値がトランスポートパラメータで受信された場合は TRANSPORT_PARAMETER_ERROR タイプのコネクションエラーで、フレームで受信された場合は FRAME_ENCODING_ERROR タイプのコネクションエラーでコネクションを直ちに閉じなければならない[MUST]（第 10.2 節を参照）。

エンドポイントは、ピアによって設定された制限を超えてはならない[MUST NOT]。送信した制限を超えるストリーム ID を持つフレームを受信したエンドポイントは、これを STREAM_LIMIT_ERROR タイプのコネクションエラーとして処理しなければならない[MUST]。エラー処理の詳細については、第 11 章を参照。

受信側が MAX_STREAMS フレームを使用してストリーム制限を通知した場合、より小さい制限を通知しても効果はない｡ストリーム制限を増やさない MAX_STREAMS フレームは無視しなければならない[MUST]。

ストリームおよびコネクションフロー制御と同様に、この文書では、MAX_STREAMS フレームを介してピアに通知するストリームのタイミングと数を決定する実装を残している。実装では、ピアが使用できるストリームの数をほぼ一定に保つために、ストリームが閉じられたときに制限を増やすことを選択できる。

ピアの制限のために新しいストリームを開くことができないエンドポイントは、STREAMS_BLOCKED フレーム (19.14 節) を送信する必要がある[SHOULD]。このシグナルはデバッグに役立つと考えられる。エンドポイントは、追加のクレジットを通知する前にこのシグナルの受信を待機してはならない[MUST NOT]。これは､ピアが少なくともラウンドトリップ全体でブロックされることを意味し､ピアが STREAMS_BLOCKED フレームを送信しないことを選択した場合は無期限にブロックされる可能性があるドポイントは終了状態からドレイン状態に入ることができる。この場合、ドレイン状態は終了状態が終了したときに終了する。つ

まり、エンドポイントは同じ終了時刻を使用するが、このコネクションでパケットの送信を停止する。

５．　コネクション

QUIC コネクションは、クライアントとサーバ間の共有状態である。

各コネクションはハンドシェイクフェーズから始まる。このフェーズでは、2 つのエンドポイントが暗号ハンドシェイクプロトコル (QUIC-TLS) を使用して共有秘密を確立し、アプリケーションプロトコルをネゴシエートする。ハンドシェイク (第 7 章) は、両方のエンドポイントが通信できることを確認し(第 8.1 節)、コネクションのパラメータを確立する(第 7.4 節)。

アプリケーションプロトコルは、いくつかの制限付きでハンドシェイクフェーズ中にコネクションを使用できる。0-RTT を使用すると、サーバからの応答を受信する前に、クライアントがアプリケーションデータを送信できる。ただし、0-RTT はリプレイ攻撃に対する保護を提供しない。[QUIC-TLS] のセクション 9.2 参照。サーバは、クライアントの身元と活性を確認できる最終的な暗号化ハンドシェイクメッセージを受信する前に、クライアントにアプリケーションデータを送信することもできる。これらの機能により、アプリケーション プロトコルは、待ち時間の短縮とセキュリティ保証をトレードするオプションを提供できる。

コネクション ID (第 5.1 節)を使用すると、エンドポイントを直接選択する場合と、中間ボックスの変更によって強制された場合の両方で、コネクションを新しいネットワークパスに移行できる。第 9 章では、移行に関連するセキュリティとプライバシーの問題の軽減策について説明する。

不要になったコネクションについては、第 10 章で説明するように、クライアントとサーバがコネクションを終了する方法がいくつかある。

5.1　コネクション ID

各コネクションは、コネクションを識別できる一連のコネクション識別子またはコネクション ID を持つ。コネクション ID はエンドポイントによって独立して選択される。各エンドポイントは、ピアが使用するコネクション ID を選択する。

コネクション ID の主な機能は、下位プロトコル層(UDP、IP)でのアドレス指定の変更によって、QUIC コネクションのパケットが間違ったエンドポイントに配信されないようにすることである。各エンドポイントは、実装固有の (おそらくデプロイメント固有の) 方法を使用してコネクション ID を選択する。これにより、そのコネクション ID を持つパケットがエンドポイントにルーティングされ、受信時にエンドポイントによって識別される。

複数のコネクション ID が使用されるのは、エンドポイントが、エンドポイントからの協力なしに、オブザーバが同じコネクション用として識別できないパケットを送信できるようにするためである(第 9.5 節を参照。)。

コネクション ID には、外部オブザーバ(つまり、発行者と協力しないもの)が同じコネクションの他のコネクション ID と関連付けるために使用できる情報を含めてはならない[MUST NOT]。簡単な例として、これは同じコネクションで同じコネクション ID が複数回発行されてはならないことを意味する[MUST NOT]。

ロングヘッダを持つパケットには、Source Connection ID フィールドと Destination Connection ID フィールドが含まれる。これらのフィールドは、新しいコネクションのコネクション ID を設定するために使用される(詳細は第 7.2 節を参照。)。

ショートヘッダ(第 17.3 節)を持つパケットには、Destination Connection ID のみが含まれ、明示的な長さは省略される。Destination Connection ID フィールドの長さは、エンドポイントに認識されていることが想定される。コネクション ID に基づいてルーティングするロードバランサーを使用するエンドポイントは、コネクション ID の固定長についてロードバランサーと合意したり、エンコード方式について合意したりできる。固定部分では、明示的な長さをエンコードできる。これにより、コネクション ID 全体の長さが変化しても、ロードバランサーによって使用される。

Version Negotiation (第 17.2.1 項) パケットは、クライアントへの正しいルーティングを保証し、パケットがクライアントによって送信されたパケットへの応答であることを示すために、クライアントによって選択されたコネクション ID をエコーする。

正しいエンドポイントにルーティングするためにコネクション ID が必要ない場合は、長さゼロのコネクション ID を使用できる。ただし、長さゼロのコネクション ID を使用しているときに同じローカル IP アドレスとポートでコネクションを多重化すると、ピアコネクションの移行、NAT 再バインド、およびクライアントポートの再利用がある場合にエラーが発生する。エンドポイントは、これらのプロトコル機能が使用されていないことが確実でない限り、長さゼロのコネクション ID を持つ複数の同時コネクションに同じ IP アドレスとポートを使用してはならない[MUST NOT]。

エンドポイントが長さゼロ以外のコネクション ID を使用する場合は、ピアがエンドポイントに送信されるパケットに対して選択できるコネクション ID の供給を確保する必要がある。これらのコネクション ID は、NEW_CONNECTION_ID フレームを使用してエンドポイントによって提供される(第 19.15 節)。

5.1.1 コネクション ID の発行

各コネクション ID には、NEW_CONNECTION_ID または RETIRE_CONNECTION_ID フレームが同じ値を参照する場合の検出に役立つシーケンス番号が関連付けられている。エンドポイントによって発行された初期コネクション ID は、ハンドシェイク中にロングパケットヘッダ(第 17.2 節)の Source Connection ID フィールドに送信される。初期コネクション ID のシーケンス番号は 0 である。preferred_address トランスポートパラメータが送信された場合、指定されたコネクション ID のシーケンス番号は 1 である。

追加のコネクション ID は、NEW_CONNECTION_ID フレーム (第 19.15 節) を使用してピアに伝達される。新しく発行された各コネクション ID のシーケンス番号は、1 ずつ増加する必要がある[MUST]。クライアントが送信する最初の Destination Connection ID フィールドに対して選択するコネクション ID と、Retry パケットによって提供されるコネクション ID には、シーケンス番号が割り当てられない。

エンドポイントがコネクション ID を発行する場合、コネクションの間、またはピアが RETIRE_CONNECTION_ID フレーム (第 19.16 節) を使用してコネクション ID を無効にするまで、このコネクション ID を持つパケットを受け入れる必要がある[MUST]。発行され、破棄されていないコネクション ID はアクティブと見なされる。アクティブなコネクション ID は、任意のパケットタイプで、いつでも現在のコネクションで使用できる。これには、preferred_address トランスポートパラメータを介してサーバによって発行されたコネクション ID が含まれる。

エンドポイントは、ピアに十分な数の使用可能なコネクション ID と未使用のコネクション ID があることを確認する必要がある[SHOULD]。エンドポイントは、active_connection_id_limit トランスポートパラメータを使用して、維持するアクティブなコネクション ID の数を通知する。エンドポイントは、ピアの制限を超えるコネクション ID を提供してはならない [MUST NOT]。エンドポイントは、NEW_CONNECTION_ID フレームでも、Retire Prior To フィールドに十分に大きな値を含めることによって、超過の破棄が必要な場合に、ピアの制限を一時的に超えるコネクション ID を送信することがある[MAY]。

NEW_CONNECTION_ID フレームによって、エンドポイントは、いくつかのアクティブなコネクション ID を追加し、Retire Prior To フィールドの値に基づいて他のコネクションを破棄することがある。NEW_CONNECTION_ID フレームを処理し、アクティブなコネクション ID を追加して破棄した後、アクティブなコネクション ID の数が active_connection_id_limit トランスポートパラメータで通知された値を超えた場合、エンドポイントは CONNECTION_ID_LIMIT_ERROR タイプのエラーでコネクションを閉じなければならない[MUST]。

エンドポイントは、ピアがコネクション ID を破棄するときに新しいコネクション ID を提供する必要がある[SHOULD]。エンドポイントがピアの active_connection_id_limit よりも少ないコネクション ID を提供した

場合、以前に未使用のコネクション ID を持つパケットを受信したときに新しいコネクション ID を提供してもよい[MAY]。エンドポイントは、コネクション ID が不足するリスクを回避するために、各コネクションに対して発行されるコネクション ID の合計数を制限してもかまわない[MAY](第 10.3.2 項を参照。)。エンドポイントは、発行されたコネクション ID と同じ数のパスを介してピアと対話する可能性があるため、パス検証ステータスなど、エンドポイントが維持するパスごとの状態の量を減らすために、コネクション ID の発行を制限してもかまわない[MAY]

移行を開始し、長さがゼロ以外のコネクション ID を必要とするエンドポイントは、ピアが使用できるコネクション ID のプールによって、ピアが移行時に新しいコネクション ID を使用できるようにする必要がある[SHOULD]。これは、プールが使い果たされた場合にピアが応答できなくなるためである。

ハンドシェイク中に長さがゼロのコネクション ID を選択するエンドポイントは、新しいコネクション ID を発行できない。長さがゼロの t Destination Connection ID フィールドは、任意のネットワークパスを介してこのようなエンドポイントに送信されるすべてのパケットで使用される。

5.1.2 コネクション ID の消費と破棄

エンドポイントは、コネクション中にいつでも、ピアに使用するコネクション ID を使用可能な別のコネクション ID に変更できる。エンドポイントは、ピアの移行に応答してコネクション ID を消費する。 (詳細については、第 9.5 節を参照。)

エンドポイントは、ピアから受信したコネクション ID のセットを保持する。そのいずれかをパケットの送信時に使用できる。エンドポイントがコネクション ID を使用から削除する場合、RETIRE_CONNECTION_ID フレームをピアに送信する。RETIRE_CONNECTION_ID フレームを送信することは、そのコネクション ID が再度使用されないことを示し、ピアが NEW_CONNECTION_ID フレームを使用して新しいコネクション ID に置き換えることを要求する。

第 9.5 節で説明したように、エンドポイントは、単一のローカルアドレスから単一の宛先アドレスに送信されるパケットにコネクション ID の使用を制限する。エンドポイントは、コネクション ID が使用されていたローカルアドレスまたは宛先アドレスをアクティブに使用しなくなった場合に、コネクション ID を破棄する必要がある[SHOULD]。

エンドポイントは、特定の状況で以前に発行されたコネクション ID の受け入れを停止する必要がある場合がある。このようなエンドポイントは、増加した Retire Prior To フィールドを含む NEW_CONNECTION_ID フレームを送信することによって、ピアにコネクション ID を破棄させることができる。エンドポイントは、ピアによって破棄されるまで、以前に発行されたコネクション ID を引き続き受け入れる必要がある[SHOULD]。エンドポイントが指定されたコネクション ID を処理できなくなった場合、コネクションを閉じてもかまわない[MAY]。

増加した Retire Prior To フィールドを受信すると、ピアは対応するコネクション ID の使用を停止し、RETIRE_CONNECTION_ID フレームでそれらを破棄してから、新しく提供されたコネクション ID をアクティブなコネクション ID のセットに追加しなければならない[MUST]。この順序により、エンドポイントは、ピアが使用可能なコネクション ID を持たず、ピアが active_connection_id_limit トランスポートパラメータで設定した制限を超えることなく、すべてのアクティブなコネクション ID を置き換えることができる(第 18.2 節を参照。)。要求時にコネクション ID の使用を停止しないと、発行側のエンドポイントがアクティブなコネクションでコネクション ID を使用し続けることができないため、コネクションエラーが発生する可能性がある。

エンドポイントは、RETIRE_CONNECTION_ID フレームがまだ確認されていないローカルで破棄されたコネクション ID の数を制限する必要がある[SHOULD]。エンドポイントは、active_connection_id_limit トランスポートパラメータの値の少なくとも 2 倍の数の RETIRE_CONNECTION_ID フレームを送信および追跡でき

るようにする必要がある[SHOULD]。エンドポイントは、コネクション ID を破棄せずに忘れてはならない[MUST NOT]。ただし、この制限を超える破棄が必要なコネクション ID を持つ場合は、CONNECTION_ID_LIMIT_ERROR タイプのコネクションエラーとして処理することを選択できる[MAY]。

エンドポイントは、RETIRE_CONNECTION_ID フレームを受信する前に、Retire Prior To フィールドの更新を発行してはならない[SHOULD NOT]。このフレームは、前の Retire Prior To 値によって示されるすべてのコネクション ID を破棄する。

5.2 パケットとコネクションの照合

着信パケットは受信時に分類される。パケットは、既存のコネクションに関連付けることも、サーバの場合は新しいコネクションを作成することもできる。

エンドポイントは、パケットを既存のコネクションに関連付ける。パケットに既存のコネクションに対応するゼロ長以外の Destination Connection ID がある場合、QUIC はそのパケットを適切に処理する。複数のコネクション ID をコネクションに関連付けることができることに注意すること(第 5.1 節を参照。) Destination Connection ID の長さがゼロで、パケット内のアドレッシング情報が、ゼロ長のコネクション ID を持つコネクションを識別するためにエンドポイントが使用するアドレッシング情報と一致する場合、QUIC はそのコネクションの一部としてパケットを処理する。エンドポイントは、宛先 IP とポートのみ、または送信元と宛先の両方のアドレスを識別に使用できるが、第 5.1 節で説明するようにコネクションが脆弱になる。

エンドポイントは、既存のコネクションに起因しないパケットに対してステートレスリセット(第 10.3 節)を送信できる。ステートレスリセットを使用すると、ピアはコネクションが使用できなくなったときにすばやく識別できる。

既存のコネクションに一致するパケットは、そのコネクションの状態と一致しない場合に破棄される。たとえば、パケットは、コネクションのプロトコルバージョンとは異なるプロトコルバージョンを示している場合、または必要なキーが使用可能になったときにパケット保護の削除が失敗した場合に破棄される。

Initial、Retry、Version Negotiation などの強力な整合性保護がない無効なパケットは、破棄される可能性がある[MAY]。エンドポイントは、エラーを検出する前にこれらのパケットの内容を処理する場合は、コネクションエラーを生成する必要がある[MUST]。または、その処理中に行われた変更を完全に元に戻す必要がある。

5.2.1 クライアントパケットの処理

クライアントに送信される有効なパケットには、クライアントが選択した値と一致する Destination Connection ID が常に含まれている。長さゼロのコネクション ID を受信することを選択したクライアントは、ローカルアドレスとポートを使用してコネクションを識別できる。既存のコネクションに一致しないパケット (Destination Connection ID、またはこの値が長さゼロの場合はローカル IP アドレスとポート) は破棄される。

パケットの並べ替えまたはロスにより、クライアントは、まだ計算されていないキーで暗号化されたコネクションのパケットを受信する可能性がある。クライアントはこれらのパケットを破棄してもよい[MAY]。あるいは、キーの計算を可能にする後のパケットを見越してそれらをバッファしてもよい[MAY]。

クライアントは、最初に選択したものとは異なるバージョンを使用するパケットを受信した場合、そのパケットを破棄しなければならない[MUST]。

5.2.2 サーバパケットの処理

サーバがサポートされていないバージョンを示すパケットを受信し、そのパケットがサポートされているバージョンの新しいコネクションを開始するのに十分な大きさである場合、サーバは第 6.1 節で説明されているように Version Negotiation パケットを送信する必要がある[SHOULD]。サーバは、Version Negotiation パ

ケットで応答するパケットの数を制限してもよい[MAY]。サーバは、サポートされていないバージョンを指定するより小さなパケットを破棄しなければならない[MUST]。

サポートされていないバージョンの最初のパケットは、バージョン固有のフィールドに対して異なるセマンティクスとエンコーディングを使用できる。特に、異なるバージョンに対して異なるパケット保護キーが使用される可能性がある。特定のバージョンをサポートしていないサーバは、パケットのペイロードを復号化したり、結果を適切に解釈したりできない可能性がある。データグラムが十分に長い場合、サーバは Version Negotiation パケットで応答する必要がある[SHOULD]。

サポートされているバージョンを持つパケット、またはバージョンフィールドがないパケットは、コネクション ID を使用してコネクションと照合される。長さゼロのコネクション ID を持つパケットの場合は、ローカルアドレスとポートを使用する。これらのパケットは、選択されたコネクションを使用して処理される。それ以外の場合、サーバは以下で説明するように続行する。

パケットが仕様に完全に準拠した Initial パケットである場合、サーバはハンドシェイクを続行する(第 7 章)。これにより、サーバはクライアントが選択したバージョンにコミットされる。

サーバが新しいコネクションの受け入れを拒否する場合、エラーコード CONNECTION_REFUSED を含む CONNECTION_CLOSE フレームを含む Initial パケットを送信する必要がある[SHOULD]。

パケットが 0-RTT パケットの場合、サーバは、遅延到着の Initial パケットを見越して、これらのパケットの数を制限してバッファすることができる[MAY]。クライアントは、サーバの応答を受信する前に Handshake パケットを送信できないため、サーバはそのようなパケットを無視する必要がある[SHOULD]。

サーバは、他のすべての状況で受信パケットを削除する必要がある[MUST]。

5.2.3 単純なロードバランサーに関する考慮事項

サーバデプロイメントでは、送信元と宛先の IP アドレスとポートのみを使用して、サーバ間で負荷を分散できる。クライアントの IP アドレスまたはポートを変更すると、パケットが間違ったサーバに転送される可能性がある。このようなサーバデプロイメントでは、クライアントのアドレスが変更されたときにコネクションの継続性を維持するために、次のいずれかの方法を使用できる。

- サーバは、帯域外メカニズムを使用して、コネクション ID に基づいて正しいサーバにパケットを転送できる。
- サーバが、クライアントが最初にコネクションしたもの以外の専用サーバの IP アドレスまたはポートを使用できる場合は、preferred_address トランスポートパラメータを使用して、クライアントがその専用アドレスにコネクションを移動するように要求できる。クライアントが優先アドレスを使用しないことを選択できることに注意すること。

クライアントアドレスが変更されたときにコネクションの継続性を維持するソリューションを実装していないデプロイメント内のサーバは、disable_active_migration トランスポートパラメータを使用して、移行がサポートされていないことを示す必要がある[SHOULD]。disable_active_migration トランスポートパラメータは、クライアントが preferred_address トランスポートパラメータを操作した後のコネクションの移行を禁止しない。

この単純な形式の負荷分散を使用するサーバデプロイメントでは、ステートレスリセットオラクルの作成を回避しなければならない[MUST](第 21.11 節を参照。)。

5.3 コネクションに対する操作

このドキュメントは QUIC 用の API を定義しない。代わりに、アプリケーションプロトコルが依存できる QUIC コネクション用の関数セットを定義する。アプリケーションプロトコルは、QUIC の実装が本節で説明する操作を含むインタフェースを提供すると想定することができる。特定のアプリケーションプロトコルで

使用するために設計された実装は、そのプロトコルで使用される操作のみを提供する可能性がある。

クライアントロールを実装する場合、アプリケーションプロトコルは以下を行うことができる。

- コネクションを開き、第 7 章で説明する交換を開始する。
- 利用可能な場合は Early Data を有効にする。
- Early Data がサーバによって受け入れられたか拒否されたかを通知する。

サーバロールを実装すると、アプリケーションプロトコルは次のことができる。

- 着信コネクションをリッスンし、第 7 章で説明されている交換の準備をする。
- Early Data がサポートされている場合は、クライアントに送信される TLS 再開チケットにアプリケーションが制御するデータを埋め込む。
- Early Data がサポートされている場合は、クライアントの再開チケットからアプリケーションが制御するデータを取得し、その情報に基づいて Early Data を受け入れるか拒否する。

どちらのロールでも、アプリケーションプロトコルは次のことができる。

- トランスポートパラメータで伝達されるように、各タイプの許可ストリームの初期数の最小値を設定する(第 7.4 節);
- ストリームとコネクションの両方のフロー制御制限を設定して、受信バッファのリソース割り当てを制御する;
- ハンドシェイクが正常に完了したか、まだ進行中かを識別する;
- PING フレームを生成する(第 19.2 節)か、アイドルタイムアウトが期限切れになる前にトランスポートが追加のフレームを送信するように要求する(第 10.1 節)かのいずれかによって、コネクションが自動的に閉じないようにする;
- コネクションを直ちに閉じる(第 10.2 節)。

６．　バージョンネゴシエーション

バージョンネゴシエーションにより、サーバはクライアントが使用していたバージョンをサポートしていないことを示すことができる。サーバは、新しいコネクションを開始する可能性のある各パケットへの応答として Version Negotiation パケットを送信する(詳細は第 5.2 節を参照。)。

クライアントが送信する最初のパケットのサイズによって、サーバが Version Negotiation パケットを送信するかどうかが決まる。複数の QUIC バージョンをサポートするクライアントは、必要に応じて PADDING フレーム(第 19.1 節)を使用して、送信する最初の UDP データグラムが、サポートするすべてのバージョンの最小データグラムサイズのうち最大のサイズになるようにする必要がある[SHOULD]。これにより、相互にサポートされているバージョンがある場合にサーバが応答することが保証される。サーバは、受信したデータグラムが別のバージョンで指定されている最小サイズよりも小さい場合、Version Negotiation パケットを送信しないことがある(第 14.1 節を参照。)。

6.1　Version Negotiation パケットの送信

クライアントが選択したバージョンがサーバに受け入れられない場合、サーバは Version Negotiation パケットで応答する(第 17.2.1 項を参照。)。これには、サーバが受け入れるバージョンのリストが含まれる。エンドポイントは、Version Negotiation パケットの受信に応答して Version Negotiation パケットを送信してはならない[MUST NOT]。

このシステムにより、サーバは状態を保持することなく、サポートされていないバージョンのパケットを処理できる。応答で送信される Initial パケットまたは Version Negotiation パケットのいずれかが失われる可能性がある、クライアントは、応答を正常に受信するか、コネクション試行を破棄するまで、新しいパケッ

トを送信する。

サーバは、送信する Version Negotiation パケットの数を制限することができる[MAY]。たとえば、パケットを 0-RTT として認識できるサーバは、最終的に Initial パケットを受信することを期待して、0-RTT パケットへの応答として Version Negotiation パケットを送信しないことを選択できる。

6.2 Version Negotiation パケットの処理

Version Negotiation パケットは、QUIC がコネクションに使用する QUIC のバージョンをネゴシエートできるようにする機能を将来定義できるように設計されている。将来の標準トラックの仕様では、複数のバージョンの QUIC をサポートする実装が、このバージョンを使用してコネクションを確立しようとしたときに受信した Version Negotiation パケットにどのように反応するかが変更される可能性がある。

このバージョンの QUIC のみをサポートするクライアントは、Version Negotiation パケットを受信した場合、次の 2 つの例外を除き、現在のコネクション試行を破棄しなければならない[MUST]。クライアントは、以前の Version Negotiation パケットを含む他のパケットを受信して正常に処理した場合、Version Negotiation パケットを破棄しなければならない[MUST]。クライアントは、クライアントが選択した QUIC バージョンをリストする Version Negotiation パケットを破棄しなければならない[MUST]。

バージョンネゴシエーションをどのように実行するかは、将来の標準トラック仕様で定義される将来の作業として残されている。特に、将来の作業によって、バージョンダウングレード攻撃に対する堅牢性が保証される(第 21.12 節を参照。)。

6.3 予約バージョンの使用

サーバが将来新しいバージョンを使用するためには、クライアントはサポートされていないバージョンを正しく処理する必要がある。一部のバージョン番号(第 15 章に定義されている 0x?a?a?a?a)は、バージョン番号を含むフィールドに含めるために予約されている。

エンドポイントは、ピアが値を正しく無視することをテストするために、不明またはサポートされていないバージョンが無視されるフィールドに予約バージョンを追加してもかまわない[MAY]。たとえば、エンドポイントは予約バージョンを Version Negotiation パケットに含めることができる (第 17.2.1 項を参照。)。エンドポイントは、ピアがパケットを正しく破棄することをテストするために、予約バージョンを含むパケットを送信してもかまわない[MAY]。

7. 暗号ハンドシェイクとトランスポートハンドシェイク

QUIC は、コネクション確立の待機時間を最小限に抑えるために、暗号ハンドシェイクとトランスポートハンドシェイクの組み合わせに依存している。QUIC は、暗号ハンドシェイクを送信するために CRYPTO フレーム(第 19.6 節)を使用する。このドキュメントで定義されている QUIC のバージョンは 0x00000001 として識別され、[QUIC-TLS] で説明されているように TLS を使用する。異なる QUIC バージョンは、異なる暗号ハンドシェイクプロトコルが使用されていることを示している可能性がある。

QUIC は、暗号ハンドシェイクデータの信頼性の高い順序付き配信を提供する。QUIC パケット保護は、可能な限り多くのハンドシェイクプロトコルを暗号化するために使用される。暗号ハンドシェイクは、次のプロパティを提供しなければならない[MUST]。

- 認証されたキー交換。
  - サーバは常に認証される。
  - クライアントはオプションで認証される。
  - すべてのコネクションで個別の関連のないキーが生成される。
  - キー情報は、0-RTT パケットと 1-RTT パケットの両方のパケット保護に使用できる。

- 両方のエンドポイントのトランスポートパラメータの値の認証済み交換、およびサーバトランスポートパラメータの機密保護(第 7.4 節を参照)。
- アプリケーションプロトコルの認証済みネゴシエーション (TLS はこの目的でアプリケーション層プロトコルネゴシエーション (ALPN) [ALPN] を使用する) 。

CRYPTO フレームは、異なるパケット番号空間(第 12.3 節)で送信できる。暗号化ハンドシェイクデータの順序どおりの配信を保証するために CRYPTO フレームによって使用されるオフセットは、各パケット番号空間でゼロから始まる。

図 4 は、簡略化されたハンドシェイクと、ハンドシェイクを進めるために使用されるパケットとフレームの交換を示している。可能な場合は、ハンドシェイク中のアプリケーションデータの交換が有効になる(アスタリスク ("*") で示されている)。ハンドシェイクが完了すると、エンドポイントはアプリケーションデータを自由に交換できるようになる。

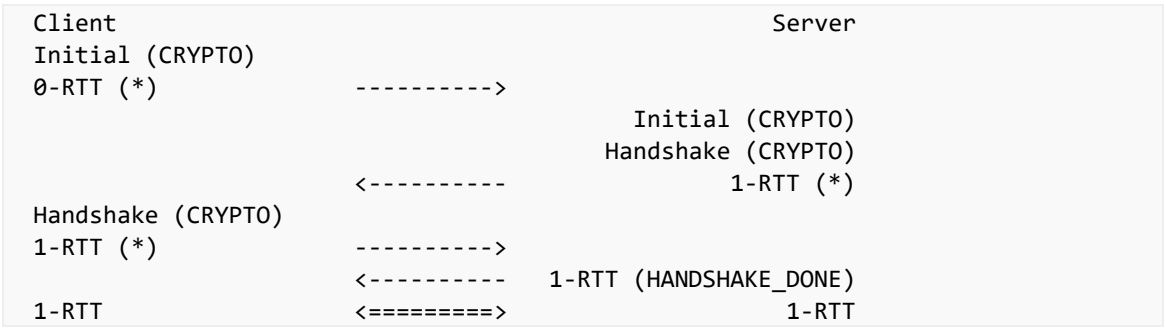

図 4　簡略化された QUIC ハンドシェイク

エンドポイントは、ハンドシェイク中に送信されたパケットを使用して、明示的輻輳通知 (ECN) のサポートをテストできる (第 13.4 節を参照。)。エンドポイントは、第 13.4.2 項で説明されているように、送信する最初のパケットを確認する ACK フレームが ECN カウントを運ぶかどうかを観察することによって、ECN のサポートを検証する。

エンドポイントは、アプリケーションプロトコルを明示的にネゴシエートしなければならない。これにより、使用中のプロトコルについて不一致がある状況を回避できる[MUST]。

7.1　ハンドシェイクフローの例

TLS が QUIC と統合される方法の詳細は、[QUIC-TLS] で提供されるが、ここではいくつかの例を示す。クライアントアドレス検証をサポートするためのこの交換の拡張は、第 8.1.2 項に示されている。

アドレス検証の交換が完了すると、暗号化ハンドシェイクを使用して暗号化キーに同意する。暗号ハンドシェイクは、Initial (第 17.2.2 項) および Handshake (第 17.2.4 項) パケットで実行される。

図 5 は、1-RTT ハンドシェイクの概要を示している。各行は、最初にパケットタイプとパケット番号が示されている QUIC パケットを示し、その後に通常これらのパケットに含まれるフレームが続く。たとえば、最初のパケットは Initial タイプで、パケット番号は 0 で、ClientHello を運ぶ CRYPTO フレームが含まれている。

複数の QUIC パケットは、パケットタイプが異なっていても、単一の UDP データグラムにまとめることができる(第 12.2 節を参照。)。結果として、このハンドシェイクは、最小で 4 つの UDP データグラム、または任意の数(輻輳制御や増幅防止などのプロトコル固有の制限に従う)で構成される。たとえば、サーバの最初のフライトには、Initial パケット、Handshake パケット、および 1-RTT パケット内の"0.5-RTT データ"が含まれる。

```
Client                                                  Server
Initial[0]: CRYPTO[CH] ->
                                  Initial[0]: CRYPTO[SH] ACK[0]
                        Handshake[0]: CRYPTO[EE, CERT, CV, FIN]
                                  <- 1-RTT[0]: STREAM[1, "..."]
Initial[1]: ACK[0]
Handshake[0]: CRYPTO[FIN], ACK[0]
1-RTT[0]: STREAM[0, "..."], ACK[0] ->
                                             Handshake[1]: ACK[0]
          <- 1-RTT[1]: HANDSHAKE_DONE, STREAM[3, "..."], ACK[0]
```

図 5　1-RTT ハンドシェイクの例

図 6 は、0-RTT ハンドシェイクと 0-RTT データの 1 つのパケットを持つコネクションの例を示している。第 12.3 節で説明したように、サーバは 0-RTT データを 1-RTT パケットで受信確認し、クライアントは同じパケット番号空間で 1-RTT パケットを送信することに注意すること。

```
Client                                                  Server
Initial[0]: CRYPTO[CH]
0-RTT[0]: STREAM[0, "..."] ->
                                  Initial[0]: CRYPTO[SH] ACK[0]
                                   Handshake[0] CRYPTO[EE, FIN]
                             <- 1-RTT[0]: STREAM[1, "..."] ACK[0]
Initial[1]: ACK[0]
Handshake[0]: CRYPTO[FIN], ACK[0]
1-RTT[1]: STREAM[0, "..."] ACK[0] ->
                                             Handshake[1]: ACK[0]
          <- 1-RTT[1]: HANDSHAKE_DONE, STREAM[3, "..."], ACK[1]
```

図 6　0-RTT ハンドシェイクの例

7.2　コネクション ID のネゴシエーション

第 5.1 節で説明されているように、コネクション ID はパケットの一貫したルーティングを保証するために使用される。ロングヘッダには 2 つのコネクション ID が含まれている。Destination Connection ID はパケットの受信側によって選択され、一貫したルーティングを提供するために使用される。Source Connection ID は、ピアによって使用される Destination Connection ID を設定するために使用される。

ハンドシェイク中に、ロングヘッダを持つパケット(第 17.2 節)は、両方のエンドポイントによって使用されるコネクション ID を確立するために使用される。各エンドポイントは、Source Connection ID フィールドを使用して、送信されるパケットの Destination Connection ID フィールドで使用されるコネクション ID を指定する。最初の Initial パケットを処理した後、各エンドポイントは、送信する後続のパケットの Destination Connection ID フィールドを、受信した Source Connection ID フィールドの値に設定する。

以前にサーバから Initial パケットまたは Retry パケットを受信していないクライアントによって Initial パケットが送信されると、クライアントは Destination Connection ID フィールドに予測できない値を入力する。この Destination Connection ID は、少なくとも 8 バイトの長さでなければならない[MUST]。パケットがサーバから受信されるまで、クライアントはこのコネクションのすべてのパケットで同じ Destination Connection ID 値を使用しなければならない[MUST]。

クライアントによって送信された最初の Initial パケットの Destination Connection ID フィールドは、Initial パケットのパケット保護キーを決定するために使用される。これらのキーは、Retry パケットの受信後に変更される ([QUIC-TLS] のセクション 5.2 を参照。)。

クライアントは、Source Connection ID フィールドに選択した値を入力し、Source Connection ID の長さフィールドを設定して長さを示す。

最初のフライトの 0-RTT パケットは、クライアントの最初の Initial パケットと同じ Destination Connection ID と Source Connection ID の値を使用する。

最初にサーバから Initial パケットまたは Retry パケットを受信すると、クライアントは、サーバによって提供された Source Connection ID を、すべての 0-RTT パケットを含む後続のパケットの Destination Connection ID として使用する。これは、クライアントがコネクションの確立中に 2 回、Destination Connection ID フィールドに設定したコネクション ID を変更する必要があることを意味する。1 回は Retry パケットに応答し、もう 1 回はサーバからの Initial パケットに応答する。クライアントは、サーバから有効な Initial パケットを受信すると、そのコネクションで受信した後続のパケットを異なる Source Connection ID で破棄する必要がある[MUST]。

クライアントは、最初に受信した Initial パケットまたは Retry パケットに応答してパケットを送信するために使用する Destination Connection ID を変更する必要がある[MUST]。サーバは、最初に受信した Initial パケットに基づいてパケットを送信するために使用する Destination Connection ID を設定する必要がある[MUST]。Destination Connection ID に対するそれ以上の変更は、値が NEW_CONNECTION_ID フレームから取得された場合にのみ許可される。後続の Initial パケットに異なる Source Connection ID が含まれている場合、それらは破棄されなければならない[MUST]。これにより、異なる送信元 Source ConnectiID を持つ複数の Initial パケットのステートレス処理によって生じる可能性のある予測不能な結果を回避できる。

エンドポイントが送信する Destination Connection ID は、特にコネクションの移行 (第 9 章) に応じて、コネクションの存続期間中に変更される可能性がある(詳細は第 5.1.1 項を参照。)。

7.3 コネクション ID の認証

各エンドポイントがハンドシェイク中に行うコネクション ID の選択は、トランスポートパラメータにすべての値を含めることによって認証される(第 7.4 節を参照。)。これにより、ハンドシェイクに使用されるすべてのコネクション ID も暗号ハンドシェイクによって認証される。

各エンドポイントは、送信した最初の Initial パケットの Source Connection ID フィールドの値を initial_source_connection_id トランスポートパラメータに含める (第 18.2 節を参照。)。サーバは、クライアントから受信した最初の Initial パケットの Destination Connection ID フィールドを original_destination_connection_id トランスポートパラメータに含める。サーバが Retry パケットを送信した場合、これは Retry パケットを送信する前に受信した最初の Initial パケットを指す。Retry パケットを送信する場合、サーバは Retry パケットの Source Connection ID フィールドも retry_source_connection_id トランスポートパラメータに含める。

ピアがこれらのトランスポートパラメータに対して提供する値は、エンドポイントが送信した(サーバの場合は受信した)Initial パケットの宛先および Source Connection ID フィールドで使用した値と一致する必要がある[MUST]。エンドポイントは、受信したトランスポートパラメータが受信したコネクション ID 値と一致することを検証する必要がある[MUST]。トランスポートパラメータにコネクション ID 値を含めて検証することにより、攻撃者がハンドシェイク中、自身が選択したコネクション ID を含むパケットを挿入することで、成功したコネクションのためのコネクション ID の選択に影響を与えることができなくなる。

エンドポイントは、いずれかのエンドポイントから initial_source_connection_id トランスポートパラメータがない場合、またはサーバから original_destination_connection_id トランスポートパラメータがない場合に、TRANSPORT_PARAMETER_ERROR タイプのコネクションエラーとして処理する必要がある[MUST]。

エンドポイントは、次のものを TRANSPORT_PARAMETER_ERROR タイプまたは PROTOCOL_VIOLATION タイプのコネクションエラーとして処理する必要がある[MUST]。

- Retry パケットを受信した後にサーバから retry_source_connection_id トランスポートパラメータがない場合。

- Retry パケットが受信されなかったときに retry_source_connection_id トランスポートパラメータが存在する場合。
- これらのトランスポートパラメータでピアから受信した値と、Initial パケットの対応する宛先または Source Connection ID フィールドで送信された値が一致しない場合。

長さゼロのコネクション ID が選択されている場合、対応するトランスポートパラメータは長さゼロの値に含まれる。

図 7 は、完全なハンドシェイクで使用されるコネクション ID (DCID= Destination Connection ID、SCID= Source Connection ID)を示している。Initial パケットの交換と、その後のハンドシェイク中に確立されたコネクション ID を含む 1-RTT パケットの交換が示されている。

```
Client                                                  Server
Initial: DCID=S1, SCID=C1 ->
                                  <- Initial: DCID=C1, SCID=S3
                                ...
1-RTT: DCID=S3 ->
                                             <- 1-RTT: DCID=C1
```

図 7　ハンドシェイクでのコネクション ID の使用

図 8 は、Retry パケットを含む同様のハンドシェイクを示している。

```
Client                                                  Server
Initial: DCID=S1, SCID=C1 ->
                                    <- Retry: DCID=C1, SCID=S2
Initial: DCID=S2, SCID=C1 ->
                                  <- Initial: DCID=C1, SCID=S3
                                ...
1-RTT: DCID=S3 ->
                                             <- 1-RTT: DCID=C1
```

図 8　再試行を伴うハンドシェイクでのコネクション ID の使用

どちらの場合も (図 7 と図 8)、クライアントは initial_source_connection_id トランスポートパラメータの値を C1 に設定する。

ハンドシェイクに再試行が含まれていない場合 (図 7)、サーバは original_destination_connection_id を S1 に設定し (この値はクライアントによって選択されることに注意すること)、initial_source_connection_id を S3 に設定する。この場合、サーバは retry_source_connection_id トランスポートパラメータを含めない。

ハンドシェイクに再試行が含まれている場合 (図 8)、サーバは original_destination_connection_id を S1 に、retry_source_connection_id を S2 に、initial_source_connection_id を S3 に設定する。

7.4　トランスポートパラメータ

コネクションの確立中に、両方のエンドポイントがトランスポートパラメータの認証済み宣言を行う。エンドポイントは、各パラメータが定義する制限に準拠する必要がある。各パラメータの説明には、その処理に関する規則が含まれている。

トランスポートパラメータは、各エンドポイントによって一方的に行われる宣言である。各エンドポイントは、ピアによって選択された値とは無関係に、トランスポートパラメータの値を選択できる。

トランスポートパラメータのエンコーディングについては、第 18 章で詳しく説明する。

QUIC は、暗号ハンドシェイクにエンコードされたトランスポートパラメータを含める。ハンドシェイクが完了すると、ピアによって宣言されたトランスポートパラメータが使用可能になる。各エンドポイントは、ピアによって提供された値を検証する。

定義された各トランスポートパラメータの定義は、第 18.2 節に含まれる。

エンドポイントは、無効な値を持つトランスポートパラメータの受信を TRANSPORT_PARAMETER_ERROR タイプのコネクションエラーとして処理しなければならない[MUST]。

エンドポイントは、特定のトランスポートパラメータ拡張でパラメータを複数回送信してはならない[MUST NOT]。エンドポイントは、重複するトランスポートパラメータの受信を TRANSPORT_PARAMETER_ERROR タイプのコネクションエラーとして処理する必要がある[SHOULD]。

エンドポイントは、ハンドシェイク中にコネクション ID のネゴシエーションを認証するためにトランスポートパラメータを使用する(第 7.3 節を参照。)。

ALPN ([ALPN] を参照) を使用すると、クライアントはコネクションの確立中に複数のアプリケーションプロトコルを提供できる。ハンドシェイク中にクライアントが含めるトランスポートパラメータは、クライアントが提供するすべてのアプリケーションプロトコルに適用される。アプリケーションプロトコルは、初期フロー制御制限などのトランスポートパラメータの値を推奨できる。ただし、トランスポートパラメータの値に制約を設定するアプリケーションプロトコルでは、これらの制約が競合する場合に、クライアントが複数のアプリケーションプロトコルを提供できなくなる可能性がある。

7.4.1 0-RTT のトランスポートパラメータの値

0-RTT の使用は、以前のコネクションからネゴシエートされたプロトコルパラメータを使用するクライアントとサーバの両方に依存する。0-RTT を有効にするために、エンドポイントは、コネクションで受信したセッションチケットと共にサーバのトランスポートパラメータの値を格納する。エンドポイントは、アプリケーションプロトコルまたは暗号ハンドシェイクに必要な情報も格納する([QUIC-TLS] のセクション 4.6 を参照。)。格納されているトランスポートパラメータの値は、セッションチケットを使用して 0-RTT を試行するときに使用される。

記憶されているトランスポートパラメータは、ハンドシェイクが完了し、クライアントが 1-RTT パケットの送信を開始するまで、新しいコネクションに適用される。ハンドシェイクが完了すると、クライアントはハンドシェイクで確立されたトランスポートパラメータを使用する。将来のコネクションに適用されないものや、0-RTT の使用に影響しないものもあるため、すべてのトランスポートパラメータが記憶されるわけではない。

新しいトランスポートパラメータの定義 (第 7.4.2 項) では、0-RTT のトランスポートパラメータの格納が必須、オプション、または禁止されているかどうかを指定しなければならない[MUST]。クライアントは、処理できないトランスポートパラメータを格納する必要はない。

クライアントは、ack_delay_exponent、max_ack_delay、initial_source_connection_id、original_destination_connection_id、preferred_address、retry_source_connection_id、および stateless_reset_token のパラメータに記憶された値を使用してはならない[MUST NOT]。クライアントは、代わりにハンドシェイクでサーバの新しい値を使用しなければならない[MUST]。サーバが新しい値を提供しない場合は、デフォルト値が使用される。

0-RTT データを送信しようとするクライアントは、処理できるサーバによって使用される他のすべてのトランスポートパラメータを記憶しておく必要がある[MUST]。サーバは、これらのトランスポートパラメータを記憶することも、チケット内の値の整合性保護されたコピーを格納し、0-RTT データを受け入れるときに情報を復元することもできる。サーバは、0-RTT データを受け入れるかどうかを決定する際にトランスポートパラメータを使用する。

0-RTT データがサーバによって受け入れられた場合、サーバは、その 0-RTT データでクライアントによって違反される可能性のある制限を減らしたり、値を変更したりしてはならない[MUST NOT]。特に、0-RTT データを受け入れるサーバは、パラメータの記憶されている値よりも小さい次のパラメータ(第 18.2 節)の値

を設定してはならない[MUST NOT]。

- active_connection_id_limit
- initial_max_data
- initial_max_stream_data_bidi_local
- initial_max_stream_data_bidi_remote
- initial_max_stream_data_uni
- initial_max_streams_bidi
- initial_max_streams_uni

特定のトランスポートパラメータに対してゼロの値を省略または設定すると、0-RTT データが有効になっても使用できなくなる可能性がある。アプリケーションデータの送信を許可するトランスポートパラメータの該当するサブセットは、0-RTT に対してゼロ以外の値に設定する必要がある[SHOULD]。これには、initial_max_data と (1) initial_max_streams_bidi と initial_max_stream_data_bidi_remote または (2) initial_max_streams_uni と initial_max_stream_data_uni のいずれかが含まれる。

サーバは、0-RTT を送信するときにクライアントが適用する記憶された値よりも大きなストリームの初期ストリームフロー制御制限を提供する場合がある。ハンドシェイクが完了すると、クライアントは、initial_max_stream_data_bidi_remote および initial_max_stream_data_uni の更新された値を使用して、すべての送信ストリームのフロー制御制限を更新する。

サーバは、max_idle_timeout、max_udp_payload_size、および disable_active_migration パラメータの以前に送信された値を格納して復元し、小さい値を選択した場合は 0-RTT を拒否することができる[MAY]。0-RTT データを受け入れながらこれらのパラメータの値を下げると、コネクションのパフォーマンスが低下する可能性がある。具体的には、max_udp_payload_size を下げると、パケットが破棄され、0-RTT データを完全に拒否する場合と比較してパフォーマンスが低下する可能性がある。

トランスポートパラメータの復元された値をサポートできない場合、サーバは 0-RTT データを拒否しなければならない[MUST]。

0-RTT パケットでフレームを送信する場合、クライアントは記憶されているトランスポートパラメータのみを使用しなければならない[MUST]。重要なのは、サーバの更新されたトランスポートパラメータまたは 1-RTT パケットで受信したフレームから学習した更新された値を使用してはならないということである[MUST NOT]。ハンドシェイクからのトランスポートパラメータの更新された値は、1-RTT パケットにのみ適用される。たとえば、記憶されているトランスポートパラメータからのフロー制御制限は、ハンドシェイクまたは 1-RTT パケットで送信されたフレームによって値が増加した場合でも、すべての 0-RTT パケットに適用される。サーバは、0-RTT で更新されたトランスポートパラメータの使用を PROTOCOL_VIOLATION タイプのコネクションエラーとして扱うことができる[MAY]。

7.4.2 新しいトランスポートパラメータ

新しいトランスポートパラメータは、新しいプロトコルの動作をネゴシエートするために使用できる。エンドポイントは、サポートしていないトランスポートパラメータを無視しなければならない[MUST]。したがって、トランスポートパラメータがないと、そのパラメータを使用してネゴシエートされるオプションのプロトコル機能が無効になる。第 18.1 節で説明されているように、一部の識別子はこの要件を実行するために予約されている。

トランスポートパラメータを理解しないクライアントは、それを破棄し、後続のコネクションで 0-RTT を試行できる。ただし、クライアントが破棄されたトランスポートパラメータのサポートを追加した場合、0-RTT を試行すると、トランスポートパラメータによって確立される制約に違反するリスクがある。新しいトランスポートパラメータは、最も保守的な値のデフォルトを設定することで、この問題を回避できる。クラ

イアントは、現在サポートされていないものも含め、すべてのパラメータを記憶することで、この問題を回避できる。

新しいトランスポートパラメータは、第 22.3 節の規則に従って登録できる。

7.5 暗号メッセージバッファ

実装は、順不同で受信した CRYPTO データのバッファを維持する必要がある。CRYPTO フレームのフロー制御がないため、エンドポイントはピアに対して無制限のデータ量のバッファを強制する可能性がある。

実装では、順序の異なる CRYPTO フレームで受信されたデータの少なくとも 4096 バイトのバッファをサポートする必要がある[MUST]。エンドポイントは、ハンドシェイク中により多くのデータをバッファできるようにすることを選択できる[MAY]。ハンドシェイク中に制限を大きくすると、より大きなキーまたは資格情報を交換できるようになる。エンドポイントのバッファサイズは、コネクションの有効期間中に一定である必要はない。

ハンドシェイク中に CRYPTO フレームをバッファできないと、コネクションエラーが発生する可能性がある。ハンドシェイク中にエンドポイントのバッファが超過した場合は、ハンドシェイクを完了するために一時的にバッファを拡張できる。エンドポイントがバッファを拡張しない場合は、CRYPTO_BUFFER_EXCEEDED エラーコードを使用してコネクションを閉じる必要がある[MUST]。

ハンドシェイクが完了すると、エンドポイントが CRYPTO フレーム内のすべてのデータをバッファできない場合は、その CRYPTO フレームと今後受信するすべての CRYPTO フレームを破棄するか[MAY]、CRYPTO_BUFFER_EXCEEDED エラーコードを使用してコネクションを閉じることができる[MAY]。破棄された CRYPTO フレームを含むパケットは、CRYPTO フレームが破棄された場合でも、トランスポートによってパケットが受信および処理されているため、受信確認を行う必要がある[MUST]。

８． アドレス検証

アドレス検証により、エンドポイントをトラフィック増幅攻撃に使用できなくなる。このような攻撃では、被害者を識別するスプーフィングされた送信元アドレス情報を使用して、パケットがサーバに送信される。サーバがそのパケットに応答してより多くのパケットを生成した場合、攻撃者はサーバを使用して、独自に送信できるよりも多くのデータを被害者に送信できる。

増幅攻撃に対する主な防止策は、ピアが要求するトランスポートアドレスでパケットを受信できることを確認することである。したがって、まだ検証されていないアドレスからパケットを受信した後、エンドポイントは、未検証のアドレスに送信するデータの量を、そのアドレスから受信したデータの量の 3 倍に制限しなければならない[MUST] 。応答のサイズに対するこの制限は、増幅防止制限と呼ばれる。

アドレス検証は、コネクション確立中(第 8.1 節参照)とコネクション移行中(第 8.2 節参照)の両方で実行される。

8.1 コネクション確立中のアドレス検証

コネクション確立によって、両方のエンドポイントのアドレスが暗黙的に検証される。特に、Handshake キーで保護されたパケットの受信は、ピアが Initial パケットを正常に処理したことを確認する。エンドポイントは、ピアからの Handshake パケットを正常に処理すると、ピアアドレスが検証されたと見なすことができる。

さらに、ピアがエンドポイントによって選択されたコネクション ID を使用し、コネクション ID に少なくとも 64 ビットのエントロピが含まれている場合、エンドポイントはピアアドレスが検証されたと見なすことができる[MAY]。

クライアントの場合、最初の Initial パケットの Destination Connection ID フィールドの値によって、パケットの正常な処理の一部としてサーバアドレスを検証できる。サーバからの Initial パケットは、この値から派

生したキーで保護される ([QUIC-TLS] のセクション 5.2 参照) 。あるいは、その値は、Version Negotiation パケット (第 6 章参照) でサーバによってエコーされるか、Retry パケットの Integrity Tag ([QUIC-TLS] のセクション 5.8 参照) に含まれる。

クライアントアドレスを検証する前に、サーバは受信したバイト数の 3 倍を超えるバイトを送信してはならない[MUST NOT]。これにより、スプーフィングされた送信元アドレスを使用してマウントできる増幅攻撃の規模が制限される。アドレス検証前に増幅を回避するために、サーバは、単一のコネクションに一意に帰属するデータグラムで受信したすべてのペイロードバイトをカウントしなければならない[MUST]。これには、正常に処理されたパケットを含むデータグラムと、すべて破棄されたパケットを含むデータグラムが含まれる。

クライアントは、Initial パケットを含む UDP データグラムに少なくとも 1200 バイトの UDP ペイロードがあることを確認し、必要に応じて PADDING フレームを追加しなければならない[MUST]。埋め込みデータグラムを送信するクライアントは、アドレス検証を完了する前に、サーバがより多くのデータを送信できるようにする。

サーバからの Initial パケットまたは Handshake パケットが失われると、クライアントが追加の Initial パケットまたは Handshake パケットを送信しない場合にデッドロックが発生する可能性がある。デッドロックは、サーバが増幅防止制限に達し、クライアントが送信したすべてのデータの受信確認を受信した場合に発生する可能性がある。この場合、クライアントが追加のパケットを送信する理由がない場合、サーバはクライアントのアドレスを検証していないため、それ以上のデータを送信できない。このデッドロックを回避するには、クライアントはプローブタイムアウト (Probe Timeout：PTO) でパケットを送信しなければならない[MUST]([QUIC-RECOVERY] 」のセクション 6.2 参照)。具体的には、クライアントは、Handshake キーがない場合は少なくとも 1200 バイトを含む UDP データグラムで Initial パケットを送信し、それ以外の場合は Handshake パケットを送信しなければならない[MUST]。

サーバは、暗号化ハンドシェイクを開始する前にクライアントアドレスを検証したい場合がある。QUIC は、ハンドシェイクを完了する前にアドレス検証を提供するために、Initial パケット内のトークンを使用する。このトークンは、Retry パケット (第 8.1.2 項参照) を使用したコネクション確立中に、または NEW_TOKEN フレーム (第 8.1.3 項参照) を使用した以前のコネクションでクライアントに配信される。

アドレス検証の前に課された送信制限に加えて、サーバは、輻輳コントローラによって設定された制限によって送信できる内容にも制限される。クライアントは、輻輳コントローラによってのみ制限される。

8.1.1 トークンの構築

NEW_TOKEN フレームまたは Retry パケットで送信されるトークンは、サーバがクライアントに提供された方法を識別できるように構築されなければならない[MUST]。これらのトークンは同じフィールドで伝達されるが、サーバとは異なる処理が必要である。

8.1.2 再試行パケットを使用したアドレス検証

クライアントの Initial パケットを受信すると、サーバはトークンを含む Retry パケット (第 17.2.5 項参照) を送信することでアドレス検証を要求できる。このトークンは、Retry パケットを受信した後にそのコネクションに対して送信するすべての Initial パケットでクライアントによって繰り返されなければならない[MUST]。

Retry パケットで提供されたトークンを含む Initial パケットの処理に応答して、サーバは別の Retry パケットを送信することはできない。コネクションを拒否するか、続行を許可することしかできない。

攻撃者が自身のアドレスに対して有効なトークンを生成できず (第 8.1.4 項参照)、クライアントがそのトークンを返すことができる限り、サーバに対してトークンを受信したことを証明する。

サーバは、Retry パケットを使用して、コネクション確立の状態と処理コストを延期することもできる。第

18.2 節で定義している original_destination_connection_id トランスポートパラメータと共に別のコネクション ID を提供するようにサーバに要求すると、サーバは、サーバまたはサーバが連携するエンティティがクライアントから元の Initial パケットを受信したことを証明する必要がある。別のコネクション ID を提供すると、サーバは後続のパケットのルーティング方法を制御することもできる。これは、別のサーバインスタンスにコネクションを転送するために使用できる。

サーバは、無効な Retry トークンを含むがそれ以外は有効なクライアントの Initial パケットを受信した場合、クライアントが別の Retry トークンを受け入れないことを認識する。サーバはこのようなパケットを破棄し、ハンドシェイクの失敗を検出するためにクライアントをタイムアウトさせることができるが、クライアントに重大な待機時間のペナルティを課す可能性がある。代わりに、サーバは INVALID_TOKEN エラーでコネクションを直ちに閉じる必要がある[SHOULD] (第 10.2 節参照)。サーバはこの時点でコネクションの状態を確立していないため、終了期間に入らないことに注意する。

Retry パケットの使用を示すフローを図 9 に示す。

```
Client                                                  Server
Initial[0]: CRYPTO[CH] ->
                                                <- Retry+Token
Initial+Token[1]: CRYPTO[CH] ->
                                 Initial[0]: CRYPTO[SH] ACK[1]
                          Handshake[0]: CRYPTO[EE, CERT, CV, FIN]
                                 <- 1-RTT[0]: STREAM[1, "..."]
```

図 9　再試行を伴うハンドシェイクの例

8.1.3　将来のコネクションのためのアドレス検証

サーバは、1 つのコネクション中に、後続のコネクションで使用できるアドレス検証トークンをクライアントに提供する場合がある[MAY]。アドレス検証は、サーバが 0-RTT データに応答して大量のデータをクライアントに送信する可能性があるため、0-RTT で特に重要である。

サーバは、NEW_TOKEN フレーム(第 19.7 節参照)を使用して、将来のコネクションを検証するために使用できるアドレス検証トークンをクライアントに提供する。将来のコネクションでは、クライアントはこのトークンを Initial パケットに含めてアドレス検証を提供する。クライアントは、再試行によってトークンが新しいものに置き換えられない限り、送信するすべての Initial パケットにトークンを含めなければならない[MUST]。クライアントは、再試行で提供されたトークンを将来のコネクションに使用してはならない[MUST NOT]。サーバは、予期されたトークンを伝送しない Initial パケットを破棄することがある[MAY]。

すぐに使用される Retry パケット用に作成されたトークンとは異なり、NEW_TOKEN フレームで送信されたトークンは、一定の期間が経過した後に使用できる。したがって、トークンには有効期限が必要である[SHOULD]。これは、明示的な有効期限または有効期限を動的に計算するために使用できる発行済みのタイムスタンプのいずれかになる。サーバは、有効期限を格納するか、暗号化された形式でトークンに含めることができる。

NEW_TOKEN フレームで発行されたトークンには、オブザーバが発行されたコネクションに値をリンクできるようにする情報を含めてはならない[MUST NOT]。たとえば、値が暗号化されていない限り、以前のコネクション ID またはアドレス指定情報を含めることはできない。サーバは、以前に送信された NEW_TOKEN フレームの損失を修復するために送信されたものを除き、送信するすべての NEW_TOKEN フレームがすべてのクライアントで一意であることを確認しなければならない[MUST]。サーバが再試行と NEW_TOKEN フレームからのトークンを区別できるようにする情報は、サーバ以外のエンティティからアクセスできる可能性がある[MAY]。

クライアントポート番号が 2 つの異なるコネクションで同じであることはほとんどない。したがって、ポートの検証が成功する可能性は低い。

NEW_TOKEN フレームで受信されたトークンは、コネクションが権限のあるものと見なされるすべてのサーバに適用できる(例:証明書に含まれるサーバ名)。クライアントが適用可能で未使用のトークンを保持しているサーバにコネクションする場合は、Initial パケットの Token フィールドにそのトークンを含める必要がある[SHOULD]。トークンを含めると、サーバは追加のラウンドトリップなしでクライアントアドレスを検証できる可能性がある。クライアントは、トークンを発行したサーバとクライアントがコネクションしているサーバがトークンを共同で管理していることを認識していない限り、コネクション先のサーバに適用されないトークンを含めてはならない[MUST NOT]。クライアントは、そのサーバへの以前のコネクションからトークンを使用してもよい[MAY]。

トークンを使用すると、サーバは、トークンが発行されたコネクションとそれが使用されているコネクションの間のアクティビティを関連付けることができる。サーバとの ID の連続性を解除したいクライアントは、NEW_TOKEN フレームを使用して提供されたトークンを破棄できる。これに対して、Retry パケットで取得されたトークンは、コネクション試行中に直ちに使用されなければならず[MUST]、その後のコネクション試行では使用できない。

クライアントは、異なるコネクション試行のために NEW_TOKEN フレームからトークンを再利用すべきではない[SHOULD NOT]。トークンを再利用すると、ネットワークパス上のエンティティによってコネクションがリンクされる(第 9.5 節参照)。

クライアントは、単一のコネクションで複数のトークンを受け取る場合がある。リンク可能性を防ぐ以外に、任意のトークンを任意のコネクション試行で使用できる。サーバは、追加のトークンを送信して、複数のコネクション試行のアドレス検証を有効にするか、無効になる可能性のある古いトークンを置き換えることができる。クライアントにとって、この曖昧さは、最新の未使用のトークンを送信することが最も効果的であることを意味する。古いトークンを保存して使用しても悪影響はないが、クライアントは、古いトークンがアドレス検証のためにサーバに役立つ可能性が低いと見なすことができる。

サーバは、アドレス検証トークンを含む Initial パケットを受信すると、アドレス検証が既に完了していない限り、トークンの検証を試行しなければならない[MUST]。トークンが無効である場合、サーバは、Retry パケットを送信する可能性を含め、クライアントが検証済みアドレスを持っていないかのように処理する必要がある[SHOULD]。NEW_TOKEN フレームで提供されるトークンと Retry パケットは、サーバによって区別でき (第 8.1.1 項参照)、後者はより厳密に検証できる。検証が成功した場合、サーバはハンドシェイクの進行を許可する必要がある[SHOULD]。

> 注意:パケットを破棄するのではなく、クライアントを未検証として扱う理論的根拠は、クライアントが以前のコネクションで NEW_TOKEN フレームを使用してトークンを受信している可能性があり、サーバが状態を失った場合、トークンを検証することができず、パケットが破棄された場合にコネクションエラーにつながる可能性があるためである。

ステートレス設計では、サーバは暗号化および認証されたトークンを使用して、サーバが後で復元してクライアントアドレスの検証に使用できる情報をクライアントに渡すことができる。トークンは暗号化ハンドシェイクに統合されていないため、認証されない。たとえば、クライアントはトークンを再利用できる可能性がある。このプロパティを悪用する攻撃を回避するために、サーバはトークンの使用をクライアントアドレスの検証に必要な情報のみに制限できる。

クライアントは、同じバージョンを使用するすべてのコネクション試行に対して、1 つのコネクションで取得したトークンを使用できる[MAY]。使用するトークンを選択する場合、クライアントは、可能なアプリケーションプロトコル、セッションチケット、またはその他のコネクションプロパティの選択など、試行されているコネクションの他のプロパティを考慮する必要はない。

8.1.4 アドレス検証トークンの整合性

アドレス検証トークンは推測が困難でなければならない[MUST]。トークンに少なくとも 128 ビットのエントロピを持つランダム値を含めることで十分であるが、これはサーバがクライアントに送信する値を記憶しているかどうかに依存する。

トークンベースのスキームを使用すると、サーバは検証に関連付けられているすべての状態をクライアントにオフロードできる。この設計を機能させるには、クライアントによる変更または改ざんに対する整合性保護によってトークンを保護しなければならない[MUST]。整合性保護がないと、悪意のあるクライアントがサーバによって受け入れられるトークンの値を生成または推測する可能性がある。トークンの整合性保護キーにアクセスする必要があるのはサーバだけである。

トークンを生成するサーバもトークンを使用するため、トークンの単一の明確に定義された形式は必要ない。Retry パケットで送信されるトークンには、クライアントパケットの送信元 IP アドレスとポートが一定であることをサーバが確認できるようにする情報を含める必要がある[SHOULD]。

NEW_TOKEN フレームで送信されるトークンには、トークンが発行されたときからクライアント IP アドレスが変更されていないことをサーバが確認できるようにする情報を含めなければならない[MUST]。サーバは、クライアントアドレスが変更された場合でも、Retry パケットを送信しないことを決定する際に NEW_TOKEN フレームのトークンを使用できる。クライアントの IP アドレスが変更された場合、サーバは増幅防止制限に従わなければならない[MUST](第 8 章参照)。NAT が存在する場合、NAT を共有する他のホストを増幅攻撃から保護するには、この要件が不十分である可能性があることに注意する。

攻撃者はトークンを再生して、DDoS 攻撃の増幅器としてサーバを使用する可能性がある。このような攻撃から保護するために、サーバはトークンの再生を防止または制限しなければならない[MUST]。サーバは、Retry パケットで送信されたトークンがクライアントによってすぐに返されるため、短時間だけ受け入れられるようにする必要がある[SHOULD]。NEW_TOKEN フレーム(第 19.7 節参照)で提供されるトークンは、より長い期間有効である必要があるが、複数回受け入れてはならない[SHOULD NOT]。サーバは、可能であれば、トークンを 1 回だけ使用できるようにすることを推奨する。トークンには、適用範囲または再利用をさらに絞り込むために、クライアントに関する追加情報を含めることができる[MAY]。

8.2 パス検証

パス検証は、アドレス変更後の到達可能性を検証するために、コネクションの移行中に両方のピアによって使用される (第 9 章を参照)。パス検証では、エンドポイントは特定のローカルアドレスと特定のピアアドレスの間の到達可能性をテストする。ここで、アドレスは IP アドレスとポートの 2 タプルである。

パス検証では、ピアへのパスで送信されたパケットがそのピアで受信されることをテストする。パス検証は、移行ピアから受信したパケットにスプーフィングされた送信元アドレスが含まれていないことを確認するために使用される。

パス検証では、ピアが戻り方向に送信できることは検証されない。確認応答は、エントロピが不十分でスプーフィングされる可能性があるため、戻りパス検証には使用できない。エンドポイントは、パスの各方向の到達可能性を個別に決定するため、戻りの到達可能性はピアによってのみ確立できる。

パス検証は、どちらのエンドポイントでもいつでも使用できる。たとえば、エンドポイントは、休止期間の後もピアがアドレスを保持していることを確認できる。

パス検証は、NAT トラバーサルメカニズムとして設計されていない。ここで説明するメカニズムは、NAT トラバーサルをサポートする NAT バインディングの作成には有効であるが、1 つのエンドポイントが、最初にそのパスでパケットを送信しなくてもパケットを受信できることが想定される。効果的な NAT トラバーサルには、ここでは提供されていない追加の同期メカニズムが必要である。

エンドポイントには、パスの検証に使用される PATH_CHALLENGE フレームおよび PATH_RESPONSE フ

レームを持つ他のフレームを含めることができる[MAY]。特に、エンドポイントには、Path Maximum Transmission Unit Discovery (PMTUD) 用の PATH_CHALLENGE フレームを持つ PADDING フレームを含めることができる(第 14.2.1 項参照)。エンドポイントは、PATH_RESPONSE フレームを送信するときに独自の PATH_CHALLENGE フレームを含めることもできる。

エンドポイントは、新しいローカルアドレスから送信されるプローブに新しいコネクション ID を使用する(第 9.5 節参照)。新しいパスをプローブするとき、エンドポイントは、ピアが応答に使用できる未使用のコネクション ID を持っていることを確認できる。ピアの active_connection_id_limit が許可されている場合、NEW_CONNECTION_ID フレームと PATH_CHALLENGE フレームを同じパケットで送信すると、応答を送信するときにピアが未使用のコネクション ID を使用できることが保証される。

エンドポイントは、複数のパスを同時にプローブすることを選択できる。プローブに使用される新しいローカルアドレスごとに、以前に未使用のコネクション ID が必要になるため、プローブに使用される同時パスの数は、ピアが以前に提供した追加のコネクション ID の数によって制限される。

8.2.1 パス検証の開始

パスの検証を開始するために、エンドポイントは、検証するパス上の予測できないペイロードを含む PATH_CHALLENGE フレームを送信する。

エンドポイントは、パケット損失を防ぐために複数の PATH_CHALLENGE フレームを送信する場合がある[MAY]。ただし、エンドポイントは、単一のパケットで複数の PATH_CHALLENGE フレームを送信することはできない[SHOULD NOT]。

エンドポイントは、Initial パケットを送信するよりも頻繁に PATH_CHALLENGE フレームを含むパケットで新しいパスをプローブすることはできない[SHOULD NOT]。これにより、コネクションの移行が新しいパスで新しいコネクションを確立するよりも負荷がかからないことが保証さる。

エンドポイントは、ピアの応答を対応する PATH_CHALLENGE フレームに関連付けることができるように、すべての PATH_CHALLENGE フレームで予測不可能なデータを使用しなければならない[MUST]。

エンドポイントは、パスの増幅防止制限がこのサイズのデータグラムの送信を許可しない場合を除き、PATH_CHALLENGE フレームを含むデータグラムを少なくとも最小の許容最大データグラムサイズである 1200 バイトに拡張しなければならない[MUST]。このサイズの UDP データグラムを送信すると、エンドポイントからピアへのネットワークパスを QUIC に使用できる(第 14 章参照)。

エンドポイントが増幅防止制限のためにデータグラムサイズを 1200 バイトに拡張できない場合、パス MTU は検証されない。パス MTU が十分に大きいことを確認するには、エンドポイントは、少なくとも 1200 バイトのデータグラムで PATH_CHALLENGE フレームを送信することによって、2 番目のパス検証を実行しなければならない[MUST]。この追加の検証は、PATH_RESPONSE フレームが正常に受信された後、または大きなデータグラムを送信しても増幅防止制限を超えないように、パス上で十分なバイトが受信された後に実行できる。

データグラムが展開される他のケースとは異なり、エンドポイントは、PATH_CHALLENGE フレームまたは PATH_RESPONSE フレームが含まれている場合に小さすぎると思われるデータグラムを破棄してはならない[MUST NOT]。

8.2.2 パス検証の応答

PATH_CHALLENGE フレームを受信すると、エンドポイントは PATH_RESPONSE フレーム内の PATH_CHALLENGE フレームに含まれるデータをエコーすることによって応答しなければならない[MUST]。エンドポイントは、輻輳制御によって制約されていない限り、PATH_RESPONSE フレームを含むパケットの送信を遅延させてはならない[MUST NOT]。

PATH_RESPONSE フレームは、PATH_CHALLENGE フレームが受信されたネットワークパスで送信され

なければならない[MUST]。これにより、パスが両方向で機能している場合にのみ、ピアによるパス検証が成功することが保証される。この要件は、パス検証を開始するエンドポイントによって強制されてはならない[MUST NOT](第 9.3.3 項参照)。

エンドポイントは、PATH_RESPONSE フレームを含むデータグラムを、少なくとも許可されている最小の最大データグラムサイズである 1200 バイトに拡張しなければならない[MUST]。これにより、パスがこのサイズのデータグラムを両方向に伝送できることが確認される。ただし、結果のデータが増幅防止制限を超える場合、エンドポイントは PATH_RESPONSE フレームを含むデータグラムを拡張してはならない[MUST NOT]。これは、受信した PATH_CHALLENGE フレームが拡張されたデータグラムで送信されなかった場合にのみ発生すると想定される。

エンドポイントは、一つの PATH_CHALLENGE フレームに応答して複数の PATH_RESPONSE フレームを送信してはならない[MUST NOT](第 13.3 節参照)。ピアは、追加の PATH_RESPONSE フレームを呼び出すために、必要に応じてさらに PATH_CHALLENGE フレームを送信することが想定されている。

8.2.3 パス検証の成功

パス検証は、前の PATH_CHALLENGE フレームで送信されたデータを含む PATH_RESPONSE フレームを受信すると成功する。任意のネットワークパスで受信された PATH_RESPONSE フレームは、PATH_CHALLENGE フレームが送信されたパスを検証する。

エンドポイントが少なくとも 1200 バイトに展開されていないデータグラムで PATH_CHALLENGE フレームを送信し、それに対する応答でピアアドレスが検証された場合、パスは検証されるが、パス MTU は検証される。その結果、エンドポイントは受信したデータ量の 3 倍以上を送信できるようになる。ただし、エンドポイントは、パスが必要な MTU をサポートしていることを確認するために、拡張されたデータグラムで別のパス検証を開始しなければならない[MUST]。

PATH_CHALLENGE フレームを含むパケットの受信確認は、悪意のあるピアによってスプーフィングされる可能性があるため、適切な検証ではない。

8.2.4 パス検証の失敗

パス検証は、パスを検証しようとしているエンドポイントがパス検証の試行を放棄した場合にのみ失敗する。

エンドポイントは、タイマに基づいてパス検証を放棄する必要がある[SHOULD]。このタイマを設定する場合、実装では新しいパスのラウンドトリップ時間が元のパスより長くなる可能性があることに注意が必要である。現在の PTO または新しいパスの PTO ([QUIC-RECOVERY] で定義している kInitialRtt を使用)の 3 倍の値を指定することを推奨する[RECOMMENDED]。

このタイムアウトにより、パス検証に失敗する前に複数の PTO が期限切れになるため、1 つの PATH_CHALLENGE フレームまたは PATH_RESPONSE フレームが失われても、パス検証の失敗は発生しない。

エンドポイントは、新しいパス上の他のフレームを含むパケットを受信する可能性があるが、パス検証を成功させるには、適切なデータを含む PATH_RESPONSE フレームが必要であることに注意する。

エンドポイントがパス検証を放棄すると、パスが使用できないと判断される。これは必ずしもコネクションの失敗を意味するわけではない。エンドポイントは、必要に応じて他のパス経由でパケットを送信し続けることができる。使用可能なパスがない場合、エンドポイントは新しいパスが使用可能になるまで待機するか、コネクションを閉じることができる。ピアへの有効なネットワークパスがないエンドポイントは、NO_VIABLE_PATH コネクションエラーを使用してこれを通知する可能性がある[MAY]。これは、ネットワークパスが存在するが、必要な MTU (第 14 章参照) をサポートしていない場合にのみ可能であることに注意する。

パス検証は、失敗以外の理由で放棄される可能性がある。主に、古いパスでのパス検証が進行中に、新しいパスへのコネクションの移行が開始された場合に発生する。

９．　コネクション移行

コネクション ID を使用すると、エンドポイントが新しいネットワークに移行することによって発生するような、エンドポイントアドレス (IP アドレスとポート) の変更にもコネクションを維持できる。この章では、エンドポイントが新しいアドレスに移行するプロセスについて説明する。

QUIC の設計は、ハンドシェイクの間、安定したアドレスを保持するエンドポイントに依存している。エンドポイントは、[QUIC-TLS] のセクション 4.1.2 で定義しているように、ハンドシェイクが確認される前にコネクションの移行を開始してはならない[MUST NOT]。

ピアが disable_active_migration トランスポートパラメータを送信した場合、エンドポイントは、ピアからの preferred_address トランスポートパラメータを処理していない限り、ハンドシェイク中にピアが使用したアドレスに別のローカルアドレスからパケット(プローブパケットを含む;第 9.1 節参照)を送信してはならない[MUST NOT]。ピアがこの要件に違反した場合、エンドポイントは、ステートレスリセットを生成せずにそのパス上の着信パケットを削除するか、パスの検証を続行してピアの移行を許可しなければならない[MUST]。ステートレスリセットを生成するか、コネクションを閉じると、ネットワーク内のサードパーティが、監視しているトラフィックをスプーフィングまたは操作することによってコネクションを閉じることができる。

ピアアドレスのすべての変更が意図的またはアクティブな移行であるとは限らない。ピアでは、NAT 再バインドが発生する可能性がある。これは、ミドルボックス (通常は NAT) によるアドレスの変更であり、新しい発信ポートまたはフローの新しい発信 IP アドレスが割り当てられる。エンドポイントは、ピアのアドレスの変更を検出した場合、以前にそのアドレスを検証していない限り、パス検証(第 8.2 節参照)を実行しなければならない[MUST]。

エンドポイントにパケットを送信するための検証済みパスがない場合、コネクション状態を破棄してもよい[MAY]。コネクション移行が可能なエンドポイントは、コネクション状態を破棄する前に新しいパスが使用可能になるまで待機してもよい[MAY]。

このドキュメントでは、第 9.6 節で説明している場合を除き、新しいクライアントアドレスへのコネクションの移行を制限する。クライアントはすべての移行を開始する責任がある。サーバは、クライアントアドレスからの非プローブパケットを確認するまで、クライアントアドレスに向けて非プローブパケット(第 9.1 節参照)を送信しない。クライアントが未知のサーバアドレスからパケットを受信した場合、クライアントはこれらのパケットを破棄しなければならない[MUST]。

9.1　新しいパスのプローブ

エンドポイントは、コネクションを新しいローカルアドレスに移行する前に、パス検証(第 8.2 節参照)を使用して、新しいローカルアドレスからピアの到達可能性をプローブすることができる[MAY]。パス検証の失敗は、単に新しいパスがこのコネクションで使用できないことを意味する。パスの検証に失敗しても、有効な代替パスが使用できない場合を除き、コネクションが終了することはない。

PATH_CHALLENGE フレーム、PATH_RESPONSE フレーム、NEW_CONNECTION_ID フレーム、および PADDING フレームは"プローブフレーム"であり、その他のすべてのフレームは"非プローブフレーム"である。プローブフレームのみを含むパケットは"プローブパケット"であり、その他のフレームを含むパケットは"非プローブパケット"である。

9.2　コネクション移行の開始

エンドポイントは、そのアドレスから非プローブフレームを含むパケットを送信することで、新しいロー

カルアドレスにコネクションを移行できる。

各エンドポイントは、コネクション確立中にピアのアドレスを検証する。したがって、移行中のエンドポイントは、ピアがピアの現在のアドレスで受信できることを知っているピアに送信できる。したがって、エンドポイントは、最初にピアのアドレスを検証することなく、新しいローカルアドレスに移行できる。

新しいパスでの到達可能性を確立するために、エンドポイントは新しいパスでパス検証(第 8.2 節参照)を開始する。エンドポイントは、ピアが次の非プローブフレームを新しいアドレスに送信するまで、パス検証を延期することができる[MAY]。

移行時に、新しいパスがエンドポイントの現在の送信レートをサポートしていない可能性がある。そのため、エンドポイントは、第 9.4 節で説明しているように、輻輳コントローラと RTT 推定値をリセットする。

新しいパスが同じ ECN 機能を持っていない可能性がある。そのため、エンドポイントは、第 13.4 節で説明しているように ECN 機能を検証する。

9.3 コネクション移行への応答

非プローブフレームを含む新しいピアアドレスからのパケットの受信は、ピアがそのアドレスに移行したことを示す。

受信側が移行を許可する場合は、新しいピアアドレスに後続のパケットを送信しなければならず[MUST]、検証がまだ進行中でない場合は、ピアのアドレスの所有権を確認するために、パス検証(第 8.2 節参照)を開始しなければならない[MUST]。受信側がピアからの未使用のコネクション ID を持っていない場合、ピアが提供するまで、新しいパスで何も送信できない(第 9.5 節参照)。

エンドポイントは、最大番号の非プローブパケットに応答して、パケットを送信するアドレスのみを変更する。これにより、再順序付けされたパケットを受信した場合に、エンドポイントが古いピアアドレスにパケットを送信しないことが保証される。

エンドポイントは、未検証のピアアドレスにデータを送信してもよいが[MAY]、第 9.3.1 項および第 9.3.2 項で説明しているように、潜在的な攻撃から保護しなければならない[MUST]。エンドポイントは、そのアドレスが最近検出された場合、ピアアドレスの検証をスキップしてもよい[MAY]。特に、何らかの形式の不正な移行を検出した後に、エンドポイントが以前に検証されたパスに戻った場合、アドレス検証をスキップし、損失検出と輻輳状態を復元することで、攻撃のパフォーマンスへの影響を軽減できる。

非プローブパケットを送信するアドレスを変更した後、エンドポイントは他のアドレスのパス検証を放棄できる。

新しいピアアドレスからのパケットの受信は、ピアでの NAT 再バインドの結果である可能性がある。

新しいクライアントアドレスを検証した後、サーバは新しいアドレス検証トークン (第 8 章参照) をクライアントに送信する必要がある[SHOULD]。

9.3.1 ピアアドレスのスプーフィング

ピアが送信元アドレスを偽装して、エンドポイントが不本意なホストに大量のデータを送信する可能性がある。エンドポイントがスプーフィングピアよりもはるかに多くのデータを送信する場合、コネクション移行を使用して、攻撃者が被害者に対して生成できるデータの量を増やすことができる。

第 9.3 節で説明したように、エンドポイントは、ピアが新しいアドレスを所有していることを確認するために、ピアの新しいアドレスを検証する必要がある。ピアのアドレスが有効と見なされるまで、エンドポイントはそのアドレスに送信するデータの量を制限する(第 8 章参照)。この制限がない場合、エンドポイントは、不審な被害者に対するサービス拒否攻撃に使用されるリスクがある。

前述のようにエンドポイントがピアアドレスの検証をスキップする場合、送信速度を制限する必要はない。

9.3.2 オンパスアドレススプーフィング

オンパス攻撃者は、スプーフィングされたアドレスを持つパケットをコピーして転送し、元のパケットよりも前に到着するようにすることで、不正なコネクション移行を引き起こす可能性がある。スプーフィングされたアドレスを持つパケットは移行中のコネクションから送信されたものと見なされ、元のパケットは重複していると見なされて破棄される。不正な移行の後、送信元アドレスのエンティティには、必要な場合でも、送信された PATH_CHALLENGE フレームの読み取りまたは応答に必要な暗号化キーがないため、送信元アドレスの検証は失敗する。

このような不正な移行によってコネクションが失敗しないようにするには、新しいピアアドレスの検証が失敗したときに、エンドポイントは最後に検証されたピアアドレスを使用するように戻さなければならない[MUST]。さらに、正当なピアアドレスからより高いパケット番号を持つパケットを受信すると、別のコネクション移行を引き起こす。これにより、不正な移行のアドレスの検証が破棄され、攻撃者が 1 つのパケットを挿入することによって開始された移行を抑制する。

エンドポイントに最後に検証されたピアアドレスに関する状態がない場合は、すべてのコネクション状態を破棄して、コネクションをサイレントに閉じなければならない[MUST]。これにより、コネクション上の新しいパケットが汎用的に処理される。たとえば、エンドポイントは、それ以降の受信パケットに応答してステートレスリセットを送信することがある[MAY]。

9.3.3 オフパスパケット転送

パケットを監視できるオフパス攻撃者は、正規のパケットのコピーをエンドポイントに転送する可能性がある。コピーされたパケットが正規のパケットの前に到着した場合、これは NAT 再バインドとして表示される。正規のパケットは重複として破棄される。攻撃者がパケットの転送を継続できる場合は、攻撃者を介してパスへの移行を引き起こす可能性がある。これにより、攻撃者はパス上に配置され、後続のすべてのパケットを監視または破棄できるようになる。

このスタイルの攻撃は、エンドポイント間の直接パスとほぼ同じ特性を持つパスを攻撃者が使用することに依存する。送信されるパケットが比較的少ない場合、またはパケット損失が試行された攻撃と一致する場合、攻撃の確度が高くなる。

元のパスで受信されたプローブ以外のパケットによって受信パケットの最大数が増加すると、エンドポイントはそのパスに戻る。このパスでパケットを引き出すと、攻撃が失敗する可能性が高くなる。したがって、この攻撃の軽減策は、パケットの交換を引き起こすことに依存する。

明らかな移行に応答して、エンドポイントは PATH_CHALLENGE フレームを使用して以前のアクティブなパスを検証しなければならない[MUST]。これにより、そのパスで新しいパケットが送信される。パスが実行可能でなくなった場合、検証の試行はタイムアウトして失敗する。パスが実行可能であっても不要になった場合、検証は成功するが、そのパスで送信されるパケットをプローブするだけになる。

アクティブなパスでPATH_CHALLENGEフレームを受信するエンドポイントは、応答で非プローブパケットを送信する必要がある[SHOULD]。攻撃者によって作成されたコピーの前に非プローブパケットが到着した場合、コネクションは元のパスに移行される。その後、別のパスに移行すると、このプロセス全体が再開される。

この防止策は不完全であるが、深刻な問題とは見なされない。元のパスの使用が複数回試行されたにもかかわらず、攻撃によるパスが元のパスよりも確実に高速である場合、攻撃とルーティングの改善を区別することはできない。

エンドポイントは、ヒューリスティックを使用して、このスタイルの攻撃の検出を改善することもできる。たとえば、パケットが古いパスで最近受信された場合、NAT の再バインドはあり得ない。同様に、IPv6 パスでの再バインドはまれである。エンドポイントは、重複したパケットを検索することもできる。逆に、コネ

クション ID の変更は、攻撃ではなく意図的な移行を示す可能性が高くなる。

9.4 損失検出と輻輳制御

新しいパスで使用可能な容量は、古いパスと同じではない可能性がある。古いパスで送信されたパケットは、新しいパスの輻輳制御または RTT 推定に寄与してはならない[MUST NOT]。

ピアの新しいアドレスの所有権を確認すると、エンドポイントは、ピアのアドレスの唯一の変更がポート番号である場合を除き、新しいパスの輻輳コントローラとラウンドトリップ時間推定値をすぐに初期値([QUIC-RECOVERY] の付録 A.3 および B.3 参照)にリセットしなければならない[MUST]。ポートのみの変更は通常、NAT の再バインドまたはその他のミドルボックスアクティビティの結果であるため、エンドポイントは、初期値に戻すのではなく、その輻輳制御状態とラウンドトリップ推定値を保持することができる[MAY]。古いパスから保持されている輻輳制御状態が、実質的に異なる特性を持つ新しいパスで使用される場合、輻輳コントローラと RTT 推定値が適応するまで、送信側は積極的に送信する可能性がある。一般に、新しいパスで以前の値を使用する場合は、実装に注意することを推奨する。

移行期間中にエンドポイントが複数のアドレスとの間でデータとプローブを送信すると、受信側で明らかに順序が変更される可能性がある。これは、結果として得られる 2 つのパスのラウンドトリップ時間が異なる可能性があるためである。複数のパス上のパケットの受信側は、受信したすべてのパケットをカバーする ACK フレームを送信する。

コネクション移行中に複数のパスが使用される可能性があるが、([QUIC-RECOVERY] で説明しているように) 単一の輻輳制御コンテキストと単一の損失回復コンテキストで十分である。たとえば、エンドポイントは、古いパスが不要になったことが確認されるまで、新しい輻輳制御コンテキストへの切り替えを遅らせる可能性がある(第 9.3.3 項で説明している場合など) 。

送信側は、損失検出が独立していて、輻輳コントローラが送信速度を過度に低下させないように、プローブパケットに例外を設定できる。エンドポイントは、PATH_CHALLENGE フレームが送信されたときに個別のタイマを設定する場合がある。これは、対応する PATH_RESPONSE フレームが受信された場合に取り消される。PATH_RESPONSE フレームが受信される前にタイマが起動した場合、エンドポイントは新しい PATH_CHALLENGE フレームを送信し、より長い期間タイマを再起動する可能性がある。このタイマは、[QUIC-RECOVERY] のセクション 6.2.1 で説明しているように設定する必要があり[SHOULD]、より積極的にしてはならない[MUST NOT]。

9.5 コネクション移行のプライバシーへの影響

複数のネットワークパスで安定したコネクション ID を使用すると、パッシブなオブザーバがそれらのパス間のアクティビティを関連付けることができる。ネットワーク間を移動するエンドポイントは、ピア以外のエンティティによってアクティビティが関連付けられることを望まない場合があるため、第 5.1 節で説明しているように、異なるローカルアドレスから送信するときに異なるコネクション ID が使用される。これを有効にするには、エンドポイントが提供するコネクション ID が他のエンティティによってリンクされないようにする必要がある。

エンドポイントは、いつでも、Destination Connection ID を別のパスで使用されていない値に変更することができる[MAY]。

エンドポイントは、複数のローカルアドレスから送信するときにコネクション ID を再利用してはならない[MUST NOT]。たとえば、第 9.2 節で説明しているようにコネクション移行を開始するとき、または第 9.1 節で説明しているように新しいネットワークパスをプローブするときである。

同様に、複数の宛先アドレスに送信する場合、エンドポイントはコネクション ID を再利用してはならない[MUST NOT]。ピアの制御外のネットワーク変更により、エンドポイントは同じ Destination Connection ID フィールド値を持つ新しい送信元アドレスからパケットを受信する可能性がある。この場合、同じローカル

アドレスから送信しながら、新しいリモートアドレスで現在のコネクション ID を引き続き使用してもよい[MAY]。

コネクション ID の再利用に関するこれらの要件は、パケットの送信にのみ適用される。コネクション ID を変更せずにパスを意図せず変更する可能性があるためである。たとえば、ネットワークが非アクティブな期間の後、NAT 再バインドによって、クライアントが送信を再開したときにパケットが新しいパスで送信される可能性がある。エンドポイントは、第 9.3 節で説明しているように、このようなイベントに応答する。

各新しいネットワークパス上で両方向に送信されるパケットに異なるコネクション ID を使用すると、異なるネットワークパス間で同じコネクションからのパケットをリンクするためにコネクション ID を使用する必要がなくなる。ヘッダ保護によって、パケット番号を使用してアクティビティを関連付けることができないことが保証される。これによって、タイミングやサイズなどのパケットの他のプロパティを使用してアクティビティを関連付けることが妨げられることはない。

新しいパス上のトラフィックが古いパス上のトラフィックに簡単にリンクできる可能性があるため、エンドポイントは、長さ 0 のコネクション ID を要求したピアとの移行を開始してはならない[SHOULD NOT]。サーバが長さ 0 のコネクション ID を持つパケットを正しいコネクションに関連付けることができる場合は、サーバが他の情報を使用してパケットを分離していることを意味する。たとえば、サーバは、HTTP 代替サービス [ALTSVC] などを使用して、すべてのクライアントに一意のアドレスを提供する場合がある。複数のネットワークパスにまたがるパケットの正しいルーティングを可能にする情報は、ピア以外のエンティティによってそれらのパスのアクティビティをリンクすることもできる。

クライアントは、非アクティブな期間の後にトラフィックを送信するときに、新しいコネクション ID、送信元 UDP ポート、または IP アドレス ([RFC8981] を参照) に切り替えることによって、リンク可能性を減らすことを望む場合がある。同時にパケットを送信するアドレスを変更すると、サーバがコネクションの移行を検出する可能性がある。これにより、NAT の再バインドや真の移行を経験しないクライアントに対しても、移行をサポートするメカニズムが確実に実行される。アドレスを変更すると、ピアが輻輳制御状態(第 9.4 節参照)をリセットする可能性があるため、アドレスは頻繁に変更する必要がある[SHOULD]。

使用可能なコネクション ID を使い果たしたエンドポイントは、新しいパスをプローブしたり、移行を開始したりすることはできない。また、ピアによるプローブや移行の試行に応答することもできない。移行が可能であり、異なるパスで送信されたパケットを関連付けることができないようにするために、エンドポイントは、ピアが移行する前に新しいコネクション ID を提供する必要がある[SHOULD](第 5.1.1 項参照)。ピアが使用可能なコネクション ID を使い果たした可能性がある場合、移行するエンドポイントは、新しいネットワークパスで送信されるすべてのパケットに NEW_CONNECTION_ID フレームを含めることができる。

9.6 サーバの優先アドレス

QUIC を使用すると、サーバは 1 つの IP アドレスでコネクションを受け入れ、ハンドシェイクの直後にこれらのコネクションをより優先されるアドレスに転送しようとすることができる。これは、クライアントが最初に複数のサーバで共有されているアドレスにコネクションし、コネクションの安定性を確保するためにユニキャストアドレスを使用したい場合に特に便利である。この節では、優先サーバアドレスにコネクションを移行するためのプロトコルについて説明する。

このドキュメントで指定している QUIC のバージョンでは、コネクション中の新しいサーバアドレスへのコネクションの移行はサポートされていない。クライアントがそのアドレスへの移行を開始していないときに、クライアントが新しいサーバアドレスからパケットを受信した場合、クライアントはこれらのパケットを破棄する必要がある[SHOULD]。

9.6.1 優先アドレスの通信

サーバは、TLS ハンドシェイクに preferred_address トランスポートパラメータを含めることによって、優

先アドレスを伝達する。

サーバは、各アドレスファミリ (IPv4 および IPv6) の優先アドレスを通信して、クライアントがネットワークコネクションに最も適したものを選択できるようにすることができる[MAY]。

ハンドシェイクが確認されると、クライアントはサーバによって提供された 2 つのアドレスのいずれかを選択し、パスの検証を開始する必要がある[SHOULD] (第 8.2 節参照)。クライアントは、preferred_address トランスポートパラメータまたは NEW_CONNECTION_ID フレームから取得した、以前に使用されていないアクティブなコネクション ID を使用してパケットを構築する。

パス検証が成功するとすぐに、クライアントは新しいコネクション ID を使用して新しいサーバアドレスへの今後のすべてのパケットの送信を開始し、古いサーバアドレスの使用を中止する必要がある[SHOULD]。パス検証が失敗した場合、クライアントはサーバの元の IP アドレスへの今後のすべてのパケットの送信を継続しなければならない[MUST]。

9.6.2 優先アドレスへの移行

優先アドレスに移行するクライアントは、移行前に選択したアドレスを検証しなければならない[MUST](第 21.5.3 項参照)。

サーバは、コネクションを受け入れた後、いつでも優先 IP アドレス宛てのパケットを受信する可能性がある。このパケットに PATH_CHALLENGE フレームが含まれている場合、サーバは第 8.2 節に従って PATH_RESPONSE フレームを含むパケットを送信する。サーバは、優先アドレスでクライアントから非プローブパケットを受信し、サーバが新しいパスを検証するまで、元のアドレスから非プローブパケットを送信しなければならない[MUST]。

サーバは、優先アドレスからクライアントへのパスをプローブしなければならない[MUST]。これは、攻撃者によって開始された誤った移行を防ぐのに役立つ。

サーバがパスの検証を完了し、優先アドレスで新しい最大のパケット番号を持つ非プローブパケットを受信すると、サーバは優先 IP アドレスからのみ、非プローブパケットをクライアントに送信し始める。サーバは、古い IP アドレスで受信されたこのコネクションの新しいパケットを破棄する必要がある[SHOULD]。サーバは、古い IP アドレスで受信された遅延パケットを処理し続けることができる[MAY]。

サーバが preferred_address トランスポートパラメータで提供するアドレスは、それらが提供されているコネクションに対してのみ有効である。クライアントは、現在のコネクションから再開されるコネクションを含む、他のコネクションにこれらを使用してはならない[MUST NOT]。

9.6.3 クライアントの移行と優先アドレスの相互作用

クライアントは、サーバの優先アドレスに移行する前に、コネクションの移行を実行する必要がある場合がある。この場合、クライアントは、クライアントの新しいアドレスから元のサーバアドレスと優先サーバアドレスの両方へのパス検証を同時に実行する必要がある[SHOULD]。

サーバの優先アドレスのパス検証が成功した場合、クライアントは元のアドレスの検証を放棄し、サーバの優先アドレスを使用して移行しなければならない[MUST]。サーバの優先アドレスのパス検証が失敗したが、サーバの元のアドレスの検証が成功した場合、クライアントは新しいアドレスに移行し、サーバの元のアドレスへの送信を継続してもよい[MAY]。

サーバの優先アドレスで受信されたパケットが、ハンドシェイク中にクライアントから観察されたものとは異なる送信元アドレスを持つ場合、サーバは第 9.3.1 項および第 9.3.2 項で説明している潜在的な攻撃から保護しなければならない[MUST]。意図的な同時移行に加えて、クライアントのアクセスネットワークがサーバの優先アドレスに対して異なる NAT バインディングを使用したためにも発生する可能性がある。

サーバは、別のアドレスからプローブパケットを受信したときに、クライアントの新しいアドレスへのパス検証を開始する必要がある[SHOULD](第 8 章参照)。

新しいアドレスに移行するクライアントは、サーバと同じアドレスファミリの優先アドレスを使用する必要がある[SHOULD]。

preferred_address トランスポートパラメータで提供されるコネクション ID は、提供されるアドレスに固有ではない。このコネクション ID は、クライアントが移行に使用できるコネクション ID を持つことを保証するために提供されるが、クライアントは任意のパスでこのコネクション ID を使用してもよい[MAY]。

9.7 IPv6 フローラベルの使用と移行

IPv6 を使用してデータを送信するエンドポイントは、ローカル API が IPv6 フローラベルの設定を許可していない場合を除き、[RFC6437] に準拠して IPv6 フローラベルを適用する必要がある[SHOULD]。

フローラベルの生成は、以前に使用されたフローラベルとのリンク可能性を最小限に抑えるように設計されなければならない[MUST](第 9.5 節参照)。

[RFC6437] は、フローラベルを生成するために擬似乱数関数を使用して値を導出することを提案している。フローラベルを生成するときに、送信元アドレスと宛先アドレスに加えて Destination Connection ID フィールドを含めることで、変更が他の監視可能な識別子の変更と同期されることを保証する。これらの入力をローカルシークレットと組み合わせる暗号化ハッシュ関数は、これを実装する 1 つの方法である。

１０． コネクションの終了

確立された QUIC コネクションは、次の 3 つの方法のいずれかで終了できる。

- アイドルタイムアウト(第 10.1 節)
- 即時クローズ(第 10.2 節)
- ステートレスリセット(第 10.3 節)

エンドポイントは、パケットを送信できる検証済みパスを持たない場合、コネクション状態を破棄してもよい[MAY]（第 8.2 節）。

10.1 アイドルタイムアウト

いずれかのエンドポイントがトランスポートパラメータ(第 18.2 節)で max_idle_timeout を指定した場合、両方のエンドポイントによって通知された max_idle_timeout の値の最小値よりも長くアイドル状態が続くと、コネクションは静かに閉じられ、その状態は破棄される。

各エンドポイントは max_idle_timeout を通知するが、エンドポイントでの有効値は、通知された 2 つの値の最小値として計算される (1 つのエンドポイントだけが非ゼロ値を通知した場合は、その唯一の通知された値が適用される)。max_idle_timeout を通知することにより、エンドポイントは、その有効値の前にコネクションを破棄する場合、即時クローズ(第 10.2 節)を開始することを約束する。

エンドポイントは、ピアからのパケットが正常に受信および処理されると、アイドルタイマを再起動する。また、エンドポイントは、最後にパケットを受信して処理した後に他の ack-eliciting パケットが送信されていない場合に、ack-eliciting パケットを送信するときにアイドルタイマを再起動する。パケットを送信するときにこのタイマを再起動すると、新しいアクティビティが開始された後にコネクションが閉じられなくなることを保証する。

アイドルタイムアウト期間が極端に短くなることを防ぐため、エンドポイントはアイドルタイムアウト期間を現在のプローブタイムアウト (PTO) の少なくとも 3 倍に増やさなければならない [MUST]。これによって、複数の PTO が期限切れとなり、アイドルタイムアウト前に複数のプローブを送信して破棄することが可能となる。

10.1.1 活性テスト

有効タイムアウトに近いパケットを送信するエンドポイントは、これらのパケットが到着する前にピアで

アイドルタイムアウト期間が期限切れになっている可能性があるため、ピアで破棄されるリスクがある。

エンドポイントは、PTO 内などでピアがすぐにタイムアウトする可能性がある場合に、PING または別の ack-eliciting フレームを送信してコネクションの活性状態をテストできる ([QUIC-RECOVERY] セクション 6.2)。これは、使用可能なアプリケーションデータを安全に再試行できない場合に特に有用である。なお、どのデータを安全に再試行するかはアプリケーションによって決定される。

10.1.2 アイドルタイムアウトの延期

エンドポイントは、応答データを期待しているが、アプリケーションデータを持っていない、または送信できない場合に、アイドルタイムアウトを回避するために ack-eliciting パケットを送信する必要がある場合がある。

QUIC の実装によって、アプリケーションにアイドルタイムアウトを延期するオプションが提供される場合がある。この機能は、アプリケーションが、開いているコネクションに関連付けられている状態の損失を回避したいが、しばらくの間アプリケーションデータを交換することを想定していない場合に使用できる。このオプションを使用すると、エンドポイントは PING フレーム(第 19.2 節)を定期的に送信でき、ピアはアイドルタイムアウト期間を再開始する。PING フレームを含むパケットを送信すると、このエンドポイントのアイドルタイムアウトが再開される。これは、パケットを受信してから送信された最初の ack-eliciting パケットである場合も同様である。PING フレームを送信すると、ピアは確認応答で応答する。これにより、エンドポイントのアイドルタイムアウトも再開される。

QUIC を使用するアプリケーションプロトコルは、アイドルタイムアウトを延期することが適切な状況に関するガイダンスを提供するべきである [SHOULD]。不要な PING フレームの送信は、パフォーマンスに悪影響を及ぼす可能性がある。

max_idle_timeout トランスポートパラメータを使用してネゴシエートされた時間を超えてパケットが送信または受信されなかった場合、コネクションはタイムアウトする (第 10 章を参照)。しかし、ミドルボックスの状態はそれよりも早くタイムアウトする可能性がある。[RFC4787] の REQ-5 では 2 分間のタイムアウト間隔を推奨しているが、経験的に、ミドルボックスで UDP フローの状態が失われないようにするには、30 秒ごとにパケットを送信する必要が示されている ([GATEWAY])。

10.2 即時クローズ

エンドポイントは CONNECTION_CLOSE フレーム (第 19.19 節) を送信して、コネクションを直ちに終了させる。CONNECTION_CLOSE フレームにより、すべてのストリームを直ちに閉じられる。オープン状態のストリームは暗黙的にリセットされたものとみなすことができる。

CONNECTION_CLOSE フレームを送信した後、エンドポイントは直ちにクローズ状態になる (第 10.2.1 項)。CONNECTION_CLOSE フレームを受信した後、エンドポイントはドレイン状態に入る (第 10.2.2 項)。

プロトコルの違反は即時クローズにつながる。

即時クローズは、アプリケーションプロトコルがコネクションを閉じるように調整した後に使用される可能性がある。これは、アプリケーションプロトコルが正常終了(グレースフルシャットダウン)をネゴシエートした後である場合がある。アプリケーションプロトコルは、両方のアプリケーションエンドポイントがコネクションを閉じることができることに同意するために必要なメッセージを交換可能である。その後、アプリケーションが QUIC に対してコネクションを閉じることを要求する。これに伴い QUIC がコネクションを閉じる際、アプリケーションから提供されたエラーコードを含む CONNECTION_CLOSE フレームがピアに終了を通知するために使用される。

クローズ状態およびドレイン状態は、コネクションが正常に終了させ、遅延または順序が入れ替わったパケットを適切に破棄することを保証するために存在する。これらの状態は、[QUIC-RECOVERY] で定義されている現在の PTO 間隔の少なくとも 3 倍の期間、持続維持されるべきである [SHOULD]。

クローズ状態またはドレイン状態を終了する前にコネクションの状態を破棄すると、遅延到着したパケットを受信したときにエンドポイントが不必要な Stateless Reset を生成してしまう可能性がある。UDP ソケットを閉じることができるエンドポイントなど、遅延到着パケットが応答を誘発しないような代替手段を持つエンドポイントは、リソースの早急な回復を可能にするために、これらの状態を早期に終了することが可能である [MAY]。なお、新しいコネクションを受け入れるためのソケットを開いたまま保持するサーバは、クローズ状態またはドレイン状態を早期に終了すべきではない [SHOULD NOT]。

クローズ状態またはドレイン状態が終了した後、エンドポイントはすべてのコネクションの状態を破棄すべきである [SHOULD]。エンドポイントは、このコネクションに属する追加の着信パケットへの応答として、Stateless Reset を送信することができる [MAY]。

10.2.1 クローズ状態のコネクション

エンドポイントは、即時クローズを開始した後、クローズ状態に入る。

クローズ状態では、エンドポイントは、CONNECTION_CLOSE フレームを含むパケットを生成し、コネクションに属するものとして識別するのに十分な情報のみを保持する。クローズ状態のエンドポイントは、コネクションに関連づけられたすべての受信パケットに応じて、CONNECTION_CLOSE フレームを含むパケットを送信する。

エンドポイントは、クローズ状態においてパケットを生成する頻度を制限するべきである [SHOULD]。たとえば、エンドポイントは、受信したパケットに応答するまでの間隔を次第に増やすように調整するか、または応答前に一定の時間を待機することが考えられる。

エンドポイントが選択したコネクション ID と QUIC バージョンは、クローズ状態のコネクションに関連付けられるパケットを識別するのに十分な情報であり、エンドポイントは、他のすべてのコネクション状態を破棄してもよい [MAY]。終了中のエンドポイントは、受信したフレームを処理する必要はない。エンドポイントは、受信パケットのパケット保護キーを保持することで、CONNECTION_CLOSE フレームを読み取り、処理できるようにすることが可能である [MAY]。

エンドポイントは、クローズ状態に入るときにパケット保護キーを破棄し、受信した UDP データグラムに応じて CONNECTION_CLOSE フレームを含むパケットを送信できる [MAY]。しかし、パケット保護キーを破棄したエンドポイントは、無効なパケットを識別および破棄することができない。そのため、エンドポイントが増幅攻撃に利用されることを防ぐには、このようなエンドポイントが送信するパケットの累積サイズを、受信してそのコネクションに帰属するパケットの累積サイズの 3 倍に制限しなければならない[MUST]。クローズ状態のコネクションにおいてエンドポイントがコネクション切断のために維持する状態を最小限にするために、エンドポイントは、受信したパケットごとにまったく同じパケットを送信してもよい[MAY]。

> 注意: クローズパケットの再送信を許可することは、各パケットに新しいパケット番号を使用するという要件(第 12.3 節)の例外である。新しいパケット番号を送信することは、主にロス回復と輻輳制御に利点があり、これらは閉じられたコネクションと関連することは想定されていない。最終パケットを再送信する場合の必要な状態は少なくて済む。

クローズ状態の間、エンドポイントは新しい送信元アドレスからのパケットを受信する可能性があり、これはコネクションの移行(第 9 章)を示している可能性がある。クローズ状態のエンドポイントは、未検証のアドレスから受信したパケットを破棄するか、未検証のアドレスに送信するパケットの累積サイズを、そのアドレスから受信するパケットのサイズの 3 倍に制限しなければならない[MUST]。

エンドポイントは、クローズ状態にキー更新を処理することは想定されていない（[QUIC-TLS] セクション 6）。キー更新は、エンドポイントがクローズ状態からドレイン状態に移行するのを妨げる可能性がある。これはエンドポイントがその後に受信したパケットを処理できなくなるためであるが、それ以外に影響ない。

10.2.2 ドレイン状態のコネクション

エンドポイントが CONNECTION_CLOSE フレームを受信すると、ドレイン状態に入る。これは、ピアがクローズ状態もしくはドレイン状態であることを示す。それ以外はクローズ状態と同一であるが、ドレイン状態のエンドポイントはパケットを送信してはならない [MUST NOT]。コネクションがドレイン状態になると、パケット保護キーを保持する必要はない。

CONNECTION_CLOSE フレームを受信したエンドポイントは、必要に応じて NO_ERROR コードを使用し、ドレイン状態に入る前に CONNECTION_CLOSE フレームを含む 1 つのパケットを送信してもよい [MAY]。ただし、エンドポイントはそれ以上のパケットを送信してはならない [MUST NOT]。そのようなことをすると、いずれかのエンドポイントがクローズ状態を終了するまで、CONNECTION_CLOSE フレームが無限に交換され続ける可能性がある。

ピアがクローズ状態またはドレイン状態であることを示す CONNECTION_CLOSE フレームを受信した場合、エンドポイントはクローズ状態からドレイン状態に移行することができる [MAY]。この場合、ドレイン状態は、クローズ状態が終了したときに終了する。つまり、エンドポイントは同じ終了時刻を使用するが、このコネクション上でのパケットの送信は停止する。

10.2.3 ハンドシェイク中の即時クローズ

CONNECTION_CLOSE フレームを送信するときの目標は、ピアがそのフレームを確実に処理することである。一般に、これはパケットが破棄されないよう、最も高いパケット保護レベルを持つパケットでフレームを送信することを意味する。ハンドシェイクが確認された後 ([QUIC-TLS] セクション 4.1.2)、エンドポイントは 1- RTT パケット内で CONNECTION_CLOSE フレームを送信しなければならない [MUST]。ただし、ハンドシェイクが確認される前は、ピアがより高度なパケット保護キーを使用できない可能性があるため、より低いパケット保護レベルのパケットで別の CONNECTION_CLOSE フレームを送信することができる [MAY]。より具体的には次のとおりである:

- クライアントは、サーバが Handshake キーを持っているかどうかを常に認識している(第 17.2.2.1 項)が、サーバはクライアントが Handshake キーを持っているかどうかを認識していない可能性がある。このような状況では、サーバは少なくとも 1 つのパケットがクライアントによって処理可能であることを保証するために、Handshake パケットと Initial パケットの両方で CONNECTION_CLOSE フレームを送信するべきである [SHOULD]。
- RTT パケットで CONNECTION_CLOSE フレームを送信するクライアントは、サーバが 0-RTT を受け入れたかどうかを保証できない。Initial パケットで CONNECTION_CLOSE フレームを送信することで、アプリケーションエラーコードが受信されない場合でも、サーバが close シグナルを受信する可能性が高まる。
- ハンドシェイクが確認される前に、ピアが 1-RTT パケットを処理できない可能性があるため、エンドポイントは Handshake パケットと 1-RTT パケットの両方で CONNECTION_CLOSE フレームを送信するべきである [SHOULD]。また、サーバは Initial パケットで CONNECTION_CLOSE フレームも送信するべきである [SHOULD]。

Initial パケットまたは Handshake パケットでタイプ 0x1d の CONNECTION_CLOSE を送信すると、アプリケーションの状態が公開されたり、アプリケーションの状態を変更するために使用されたりする可能性がある。Initial パケットまたは Handshake パケットでフレームを送信するときは、タイプ 0x1d の CONNECTION_CLOSE をタイプ 0x1c の CONNECTION_CLOSE に置き換えなければならない [MUST]。そうしない場合、アプリケーションの状態に関する情報が公開される可能性がある。エンドポイントは、Reason Phrase フィールドの値をクリアしなければならず[MUST]、タイプ 0x1c の CONNECTION_CLOSE に変換す

る際には APPLICATION_ERROR コードを使用するべきである [SHOULD]。

複数のパケットタイプで送信された CONNECTION_CLOSE フレームは、単一の UDP データグラムにまとめることができる(第 12.2 節)。

エンドポイントは、Initial パケットで CONNECTION_CLOSE フレームを送信できる。このフレームは、Initial パケットまたは Handshake パケット内で受信された未認証情報への応答である可能性がある。このような即時クローズは、正当なコネクションがサービス拒否にさらされる可能性がある。QUIC には、ハンドシェイク中の中間者攻撃に対する防御手段は含まれていない(第 21.2 節)。ただし、正当なピアに対するエラーのフィードバックを減らす代わりに、エンドポイントが CONNECTION_CLOSE によるコネクション終了するのではなく、不正なパケットを破棄することで、いくつかの形のサービス拒否を攻撃者にとってより困難にすることができる。このため、認証のないパケットでエラーが検出された場合、エンドポイントが即時に閉じるのではなく、パケットを破棄することができる[MAY]。

Initial パケットでエラーを検出したサーバように、状態が確立されていないエンドポイントは、クローズ状態に移行しない。コネクション状態がないエンドポイントは、CONNECTION_CLOSE フレームのした場合でも、クローズ期間またはドレイン期間に入らない。

10.3 ステートレスリセット

ステートレスリセットは、コネクションの状態にアクセスできないエンドポイントの最後の手段として提供される。クラッシュまたは停電が発生した場合、ピアがコネクションを適切に継続できないエンドポイントに対してデータを送信し続ける可能性がある。エンドポイントは、アクティブなコネクションに関連付けることができないパケットの受信に応じて、ステートレスリセットを送信することができる [MAY]。

ステートレスリセットは、アクティブなコネクションのエラーを示すことには適していない。致命的なコネクションエラーを通信したいエンドポイントは、可能であれば CONNECTION_CLOSE フレームを使用しなければならない[MUST]。

このプロセスをサポートするために、エンドポイントはステートレスリセットトークンを発行する。このトークンは推測しにくい 16 バイトの値である。その後、ピアがステートレスリセットを受信した場合 (そのステートレスリセットトークンで終わる UDP データグラム)、そのピアは直ちにコネクションを終了する。

ステートレスリセットトークンはコネクション ID に固有である。エンドポイントは、NEW_CONNECTION_ID フレームにおける Stateless Reset Token フィールドに値を含めるで、ステートレスリセットトークンを発行する。サーバは、ハンドシェイク中に選択したコネクション ID に適用される stateless_reset_token トランスポートパラメータを発行することもできる。これらの交換は暗号化によって保護されるため、クライアントとサーバのみがその値を認識する。クライアント側のトランスポートパラメータには機密保護がないため、クライアントは stateless_reset_token トランスポートパラメータを使用することはできないことに注意する必要がある。

トークンは、関連付けられているコネクション ID が RETIRE_CONNECTION_ID フレーム (第 19.16 節) を通じて破棄された場合に無効となる。

処理できないパケットを受信したエンドポイントは、次のレイアウトでパケットを送信する(第 1.3 節)。

```
Stateless Reset {
  Fixed Bits (2) = 1,
  Unpredictable Bits (38..),
  Stateless Reset Token (128),
}
```

図 10　ステートレスリセット

この設計により、ステートレスリセットは、可能な限り、ショートヘッダを持つ通常のパケットと区別できない形となることを保証する。

ステートレスリセットは、パケットヘッダの最初の 2 ビットから始まる UDP データグラム全体を使用する。最初のバイトの残りの部分と、それに続く任意の数のバイトは、ランダムと区別できない値に設定されるべきである [SHOULD]。データグラムの最後の 16 バイトには、ステートレスリセットトークンが含まれる。

意図された受信側以外のエンティティには、ステートレスリセットはショートヘッダを持つパケットとして表示される。ステートレスリセットが有効な QUIC パケットとして表示されるためには、予測不能ビットフィールドに少なくとも 38 ビットのデータ(または 5 バイトから固定の 2 ビットを引いた値)が含まれている必要がある。

結果として得られる最小サイズ 21 バイトは、受信側がコネクション ID の使用を必要とする場合に、ステートレスリセットを他のパケットと区別されにくいことを保証しない。この目的を達成するために、エンドポイントは、送信するすべてのパケットが、必要に応じて PADDING フレームを追加して、ピアにパケットに含めるように要求する最小コネクション ID 長よりも少なくとも 22 バイト長いことを確認する必要がある[SHOULD]。これにより、ピアによって送信されたステートレスリセットは、エンドポイントに送信された有効なパケットと区別できなくなる。43 バイト以下のパケットに応答してステートレスリセットを送信するエンドポイントは、応答するパケットよりも 1 バイト短いステートレスリセットを送信する必要がある[SHOULD]。

これらの値は、ステートレスリセットトークンがパケット保護 AEAD の最小拡張と同じ長さであることを前提としている。エンドポイントがより大きな最小拡張でパケット保護スキームをネゴシエートできた場合は、追加の予測できないバイトが必要である。

エンドポイントは、増幅に使用されないように、受信したパケットの 3 倍以上のサイズのステートレスリセットを送信してはなりらない [MUST NOT]。第 10.3.3 項では、ステートレスリセットのサイズに関する追加の制限について説明する。

エンドポイントは、有効な QUIC パケットにするには小さすぎるパケットを破棄しなければならない [MUST]。例を挙げると、[QUIC-TLS] で定義されている一連の AEAD 関数では、21 バイト未満のショートヘッダパケットは決して有効なものとはならない。

エンドポイントは、ショートヘッダを持つパケットとしてフォーマットされたステートレスリセットを送信しなければならない [MUST]。ただし、他の QUIC バージョンではロングヘッダの使用が許可される場合があるため、エンドポイントは、有効なステートレスリセットトークンで終わるすべてのパケットをステートレスリセットとして扱わなければならない [MUST]。

エンドポイントは、ロングヘッダを持つパケットへの応答としてステートレスリセットを送信してもよい [MAY]。ステートレスリセットトークンをピアが使用できるようになる前に、ステートレスリセットを送信しても効果がない。この QUIC バージョンでは、ロングヘッダを持つパケットはコネクション確立中にのみ使用される。ステートレスリセットトークンは、コネクションの確立が完了するか、または完了に近づくまで利用できないため、ロングヘッダを持つ不明なパケットを無視することは、ステートレスリセットを送信するのと同じくらい効果的である。

エンドポイントは、ショートヘッダを持つパケットから Source Connection ID を特定することはできない。そのため、ステートレスリセットで Destination Connection ID を設定することもできない。したがって、Destination Connection ID は、以前のパケットで使用された値とは異なるものになる。ランダムな Destination Connection ID を使用すると、コネクション ID が NEW_CONNECTION_ID フレームを使用して提供された新しいコネクション ID に移動した結果であるように見せることができる(第 19.15 節)。

ランダムなコネクション ID を使用すると、次の 2 つの問題が発生する。

- パケットがピアに到達しない可能性がある。Destination Connection ID がピアへのルーティングに重要な場合、このパケットが誤ってルーティングされる可能性がある。これは、応答として別のステート

レスリセットがトリガされる可能性もある(10.3.3 節)。正しくルーティングされないステートレスリセットは、無効なエラー検出および回復メカニズムである。この場合、エンドポイントは、コネクションが失敗したことを検出するために、タイマなどの他の方法に頼る必要がある。

- ランダムに生成されたコネクション ID は、ピア以外のエンティティがステートレスリセットの可能性を識別するために使用することができる。時折異なるコネクション ID を使用することがあるエンドポイントでは、この点について不確実性が生じる可能性がある。

このステートレスリセットの設計は、QUIC バージョン 1 に特有のものである。複数のバージョンの QUIC をサポートするエンドポイントは、エンドポイントがサポートする可能性のある (または状態を失う前にサポートしていた可能性のある) 任意のバージョンをサポートするピアによって受け入れられるステートレスリセットを生成する必要がある。新しいバージョンの QUIC の設計者はこのことを認識し、(1) この設計を再利用するか、または(2) データの伝送にパケットの最後の 16 バイト以外のパケットの部分を使用する必要がある。

10.3.1 ステートレスリセットの検出

エンドポイントは、UDP データグラムの末尾 16 バイトを使用して、ステートレスリセットの可能性を検出する。エンドポイントは、最近送信したデータグラムのコネクション ID とリモートアドレスに関連付けられているすべてのステートレスリセットトークンを記憶する。これには、NEW_CONNECTION_ID フレームとサーバのトランスポートパラメータから取得したステートレスリセット Token フィールドの値が含まれるが、未使用または廃止されたコネクション ID に関連付けられたステートレスリセットトークンは含まない。エンドポイントは、データグラムの最後の 16 バイトを、そのデータグラムを受信したリモートアドレスに関連付けられた全てのステートレスリセットトークンと比較することにより、受信したデータグラムをステートレスリセットとして識別する。

この比較は、すべての受信データグラムに対して実行できる。エンドポイントは、データグラムのいずれかのパケットが正常に処理された場合、このチェックをスキップしてもよい[MAY]。しかし、受信データグラムの最初のパケットがコネクションに関連付けられていない場合、または復号化できない場合は、この比較を実行しなければならない [MUST]。

エンドポイントは、使用していないコネクション ID や廃止されたコネクション ID に関連付けられているステートレスリセットトークンを確認してはならない [MUST NOT]。

データグラムをステートレスリセットトークンの値と比較するとき、エンドポイントはトークンの値に関する情報を漏らさすことなく比較を実行しなければならない[MUST]。たとえば、この比較を一定時間で実行することで、個々のステートレスリセットトークンの値が、タイミングサイドチャネルを介した情報漏洩から保護される。別の方法としては、未加工のトークン値の代わりに、変換されたステートレスリセットトークンの値を格納して比較する方法が考えられる。この場合、変換は、秘密キー(ブロック暗号、HMAC (Hashed Message Authentication Code) [RFC2104] など)を使用する暗号的に安全な擬似ランダム関数として定義される。エンドポイントは、パケットが正常に復号化されたかどうかや、または有効なステートレスリセットトークンの数に関する情報を保護することは期待されていない。

データグラムの最後の 16 バイトの値がステートレスリセットトークンと同じである場合、エンドポイントはドレイン期間に入り、このコネクションでそれ以上のパケットを送信してはならない [MUST]。

10.3.2 ステートレスリセットトークンの計算

ステートレスリセットトークンは推測が困難でなければならない[MUST]。ステートレスリセットトークンを作成するために、エンドポイントは、作成するすべてのコネクションごとにランダムに [RANDOM] シークレットを生成することができる。ただし、クラスター内に複数のインスタンスがある場合や、状態が失わ

れる可能性のあるエンドポイントのストレージに問題がある場合、調整の問題が発生する。ステートレスリセットは、特に状態が失われたケースを処理するために存在するため、このアプローチは最適ではない。

静的キーと、エンドポイントによって選択されたコネクション ID (第 5.1 節)を入力とする擬似乱数関数を使用して証明を生成することで、同じエンドポイントへのすべてのコネクションで単一の静的キーを使用できる。エンドポイントでは、HMAC[RFC2104] (例: HMAC (static_key, connection_id))または HMAC ベースの鍵導出関数 (HKDF) [RFC5869] (例: 静的キーを入力キーとして使用し、コネクション ID をソルトとして使用)を使用できる。この関数の出力は 16 バイトに切り詰められ、そのコネクションのステートレスリセットトークンを生成する。

状態を失ったエンドポイントは、同じ方法を使用して有効なステートレスリセットトークンを生成できる。コネクション ID は、エンドポイントが受信するパケットから取得される。

この設計は、エンドポイントがパケットのコネクション ID を使用してコネクションをリセットできるように、ピアは常にそのパケットでコネクション ID を送信することに依存する。この設計を使用するエンドポイントは、すべてのコネクションで同じコネクション ID の長さを使用するか、状態なしで復元できるようにコネクション ID の長さをエンコードしなければならない [MUST]。また、長さがゼロのコネクション ID を提供することはできない。

ステートレスリセットトークンを公開すると、どのエンティティもコネクションを終了できるため、値は一度しか使用できない。ステートレスリセットトークンを選択するためのこの方法は、コネクション ID と静的キーの組み合わせを別のコネクションに使用してはならない[MUST NOT]。静的キーを共有するインスタンスによって同じコネクション ID が使用されている場合や、攻撃者が状態を持たないが同じ静的キーを持つインスタンスにパケットをルーティングできる場合、サービス拒否攻撃が可能となる (第 21.11 節)。ステートレスリセットトークンを公開することでリセットされたコネクションのコネクション ID は、静的キーを共有するノードでの新しいコネクションに再利用してはならない [MUST NOT]。

同じステートレスリセットトークンを複数のコネクション ID に使用してはならない [MUST NOT]。エンドポイントは新しい値を以前のすべての値と比較する必要はないが、重複する値は PROTOCOL_VIOLATION タイプのコネクションエラーとして扱ってもよい[MAY]。

ステートレスリセットには暗号化による保護がないことに注意すること。

10.3.3 ループ

ステートレスリセットの設計では、ステートレスリセットトークンがわからないと、有効なパケットと区別できないようになっている。たとえば、サーバが別のサーバにステートレスリセットを送信した場合、その応答として別のステートレスリセットを受信する可能性があり、これにより無限の交換が発生する可能性がある。

エンドポイントは、ループを防ぐのに十分な状態を維持しない限り、送信するすべてのステートレスリセットが、それをトリガしたパケットよりも小さいことを確認しなければならない [MUST]。ループが発生した場合、最終的にはパケットが小さくなりすぎて応答をトリガできなくなる。

エンドポイントは、自分が送信したステートレスリセットの数を記憶し、上限に達すると新しいステートレスリセットの生成を停止することができる。異なるリモートアドレスごとに個別の上限を使用するとで、他のピアやコネクションが上限を達した場合でも、ステートレスリセットを使用してコネクションを閉じることができる。

41 バイトより小さいステートレスリセットは、ピアのコネクション ID の長さによっては、オブザーバによってステートレスリセットとして識別される場合がある。逆に、小さなパケットに応答してステートレスリセットを送信しない場合、ステートレスリセットは、非常に小さなパケットしか送信されない破損したコネクションを検出には役に立たない場合がある。このような障害は、タイマなどの他の手段によってのみ検

出される場合がある。

１１．　エラー処理

エラーを検出したエンドポイントは、そのエラーの存在をピアに通知する必要がある [SHOULD]。トランスポートレベルおよびアプリケーションレベルの両方のエラーがコネクション全体に影響を与える可能性がある (第 11.1 節)。アプリケーションレベルのエラーのみが単一のストリームに分離できる(第 11.2 節)。

エラーを通知するフレームには、最も適切なエラーコード (第 20 章) を含める必要がある [SHOULD]。この仕様がエラー状態を特定する場合は、使用されるエラーコードも特定される。これらは要件として表現されているが、異なる実装戦略が異なるエラーの報告を導く場合もある。特に、エンドポイントは、エラー状態を検出したときに、適用可能なエラーコードを使用してもよい [MAY]。一般的なエラーコード (PROTOCOL_VIOLATION や INTERNAL_ERROR など) は、特定のエラーコードの代わりに常に使用できる。

ステートレスリセット(第 10.3 節)は、CONNECTION_CLOSE フレームまたは RESET_STREAM フレームで通知されるエラーには適さない。ステートレスリセットは、コネクションでフレームを送信するために必要な状態を持つエンドポイントによって使用されてはならない [MUST NOT]。

11.1　コネクションエラー

プロトコルセマンティクスの明らかな違反や、コネクション全体に影響を与える状態の破損など、コネクションが使用できなくなるエラーは、CONNECTION_CLOSE フレーム(第 19.19 節) を使用して通知しなければならない [MUST]。

アプリケーション固有のプロトコルエラーは、フレームタイプが 0x1d の CONNECTION_CLOSE フレームを使用して通知される。トランスポートに固有のエラーは、このドキュメントで説明されているすべてのエラーを含めて、フレームタイプが 0x1c の CONNECTION_CLOSE フレームで伝達される。

CONNECTION_CLOSE フレームは、ロストしたパケットの中で送信される可能性がある。エンドポイントは、終了したコネクションでさらにパケットを受信した場合に、CONNECTION_CLOSE フレームを含むパケットを再送信する準備をしておく必要がある[SHOULD]。再送信の数と、この最終パケットが送信される時間を制限することは、終了したコネクションで費やされる労力を抑えることができる。

CONNECTION_CLOSE フレームを含むパケットを再送信しないことを選択したエンドポイントは、ピアがそのような最初のパケットを見逃すリスクがある。終了したコネクションのデータを受信し続けるエンドポイントで利用できる唯一のメカニズムは、ステートレスリセット処理(第 10.3 節)を試行することである。

Initial パケットの AEAD は強力な認証を提供しないため、エンドポイントは無効な Initial パケットを破棄してもよい[MAY]。この仕様においてコネクションエラーが必須である場合でも、Initial パケットの破棄は許可される。エンドポイントは、パケット内のフレームを処理しないか、または処理の効果を元に戻す場合にのみ、パケットを破棄することができる。無効な Initial パケットの破棄することは、サービス拒否にさらされる可能性を減らすために利用されることがある(第 21.2 節)。

11.2　ストリームエラー

アプリケーションレベルのエラーが単一のストリームに影響するが、それ以外の場合はコネクションが回復可能な状態である場合、エンドポイントは適切なエラーコードを指定した RESET_STREAM フレーム(第 19.4 節)を送信し、影響を受けたストリームだけを終了させることができる。

アプリケーションプロトコルの関与なしにストリームをリセットすると、アプリケーションプロトコルが回復不能な状態になる可能性がある。RESET_STREAM フレームは、QUIC を使用するアプリケーションプロトコルによってのみ開始されなければならない [MUST]。

RESET_STREAM フレームに含まれるアプリケーションエラーコードのセマンティクスは、アプリケーションプロトコルによって定義される。アプリケーションプロトコルのみがストリームを終了させることが

できる。アプリケーションプロトコルのローカルインスタンスは直接 API コールを使用し、リモートインスタンスは自動的な RESET_STREAM フレームをトリガする STOP_SENDING フレームを使用する。

アプリケーションプロトコルは、いずれかのエンドポイントによって途中で取り消されたストリームを処理するためのルールを定義する必要がある [SHOULD]。

１２．　パケットとフレーム

QUIC エンドポイントはパケットを交換することで通信を行う。パケットには機密性と整合性の保護がある(第 12.1 節)。パケットは UDP データグラムで伝送される(第 12.2 節)。

このバージョンの QUIC は、コネクションの確立時に長いパケットヘッダを使用する(第 17.2 節)。ロングヘッダを持つパケットは、Initial (第 17.2.2 項)、0-RTT (第 17.2.3 項)、Handshake (第 17.2.4 項)、および Retry (第 17.2.5 項) である。バージョンネゴシエーションは、ロングヘッダを持つバージョンに依存しないパケットを使用する(第 17.2.1 項)。

ショートヘッダを持つパケットは、最小限のオーバーヘッドにするように設計されており、コネクションが確立され、1-RTT キーが使用可能になった後に使用される(第 17.3 節)。

12.1　保護されたパケット

QUIC パケットには、パケットの種類に基づいて異なるレベルの暗号保護がある。パケット保護の詳細については、[QUIC-TLS]を参照のこと。この節では提供される保護の概要が含まれている。

Version Negotiation パケットは暗号保護を持たない([QUIC-INVARIANTS])。

Retry パケットは、偶発的な改変から保護するために AEAD 関数 [AEAD] を使用する。

Initial パケットは AEAD 関数を使用し、この関数のキーは経路上に表示される値を使用して導出される。そのため、Initial パケットには効果的な機密保護はない。Initial 保護は、パケットの送信側がネットワークパス上にいることを保証するために存在する。クライアントから Initial パケットを受信したエンティティは、パケットの内容の読み取りや、いずれかのエンドポイントでも正常に認証される Initial パケットの生成の両方を可能にするキーを復元できる。また、AEAD は、Initial パケットを偶発的な改変からも保護する。

その他のすべてのパケットは、暗号ハンドシェイクから導出したキーで保護される。暗号ハンドシェイクは、Handshake、0-RTT パケットおよび 1-RTT パケットの対応するキーを、通信するエンドポイントのみが受信することを保証する。0-RTT キーと 1-RTT キーで保護されたパケットは、強力な機密性と完全性の保護を持つ。

一部のパケットタイプに現れるパケット番号フィールドには、ヘッダ保護の一部として適用される別の機密保護がある ([QUIC-TLS] セクション 5.4)。基礎となるパケット番号は、与えられたパケット番号空間で送信されるパケットごとに増加する (第 12.3 節)。

12.2　パケットの結合

Initial パケット (第 17.2.2 項)、0-RTT パケット (第 17.2.3 項)、および Handshake パケット (第 17.2.4 項)には、パケットの終了位置を決定する長さフィールドが含まれる。この長さには、パケット番号フィールドとペイロードフィールドの両方が含まれ、どちらも機密保護されており、最初は未知の長さである。ペイロードフィールドの長さは、ヘッダ保護が削除されることで判明する。

長さフィールドを使用することで、送信側は複数の QUIC パケットを 1 つの UDP データグラムにまとめることができる。これにより、暗号ハンドシェイクを完了してデータの送信を開始するために必要な UDP データグラムの数を削減できる。また、これは、Path Maximum Transmission Unit (PMTU) プローブを作成するためにも使用できる (第 14.4.1 項)。受信側は結合されたパケットを処理できなければならない[MUST]。

パケットを暗号化レベルの低い方から順(Initial、0-RTT、Handshake、1-RTT([QUIC-TLS] のセクション 4.1.4))に結合すると、受信側がすべてのパケットを 1 回のパスで処理できる可能性が高くなる。ショートヘッダを

持つパケットには長さが含まれていないため、UDP データグラムに含まれる最後のパケットとしてのみ配置可能である。エンドポイントは、同じ暗号化レベルで複数のパケットを結合して送信するのではなく、同じ暗号化レベルで送信する場合は、1 つのパケットに複数のフレームを含めるべきである [SHOULD]。

受信側は、UDP データグラムに含まれる最初のパケットの情報に基づいてルーティングすることができる [MAY]。送信側は、異なるコネクション ID を持つ QUIC パケットを 1 つの UDP データグラムに結合してはならない [MUST NOT]。受信側は、データグラム中の最初のパケットとは異なる Destination Connection ID を持つ後続のパケットを無視するべきである [SHOULD]。

1 つの UDP データグラムに結合されるすべての QUIC パケットは、それぞれ独立した完全なものである。結合された QUIC パケットの受信側は、異なる UDP データグラムのペイロードとして受信されたかのように、各 QUIC パケットを個別に処理し、個別に認識しなければならない [MUST]。たとえば、復号化が失敗した場合 (キーが利用できない場合や、その他の理由により)、受信側は、パケットを破棄するか、後で処理のためにバッファリングしてもよい[MAY]。ただし、残りのパケットの処理を試みなければならない [MUST]。

Retry パケット (第 17.2.5 項)、Version Negotiation パケット (第 17.2.1 項))、およびショートヘッダを持つパケット(第 17.3 節)には Length フィールドが含まれていないため、同じ UDP データグラム内で他のパケットの後に続くことはできない。また、Retry パケットまたは Version Negotiation パケットが別のパケットと結合される状況は発生しないことに注意しなければならない。

12.3　パケット番号

パケット番号は、0~2^{62}-1 の範囲の整数である。この番号は、パケット保護のための暗号ナンスの決定に使用される。各エンドポイントは、送信用と受信用にそれぞれ別のパケット番号を保持する。

パケット番号は、ACK フレームの Largest Acknowledged フィールド(第 19.3 節)で全体を表現する必要があるため、この範囲に制限される。ただし、ロングヘッダまたはショートヘッダに存在する場合、パケット番号は縮小され、1~4 バイトでエンコードされる(第 17.1 節)。

Version Negotiation パケット (第 17.2.1 項) および Retry パケット(第 17.2.5 項)には、パケット番号は含まれない。

QUIC では、パケット番号は 3 つのスペースに分割される。

初期空間: すべての Initial パケット (第 17.2.2 項) はこの空間に含まれる。

ハンドシェイク空間: すべての Handshake パケット (第 17.2.4 項) はこの空間に含まれる。

アプリケーションデータ空間: すべての 0-RTT パケット (第 17.2.3 項) および 1-RTT (第 17.3.1 項) パケットはこの空間に含まれる。

[QUIC-TLS]で説明されているように、各パケットタイプでは異なる保護キーが使用される。

概念的には、パケット番号空間とは、パケットを処理および確認応答できるコンテキストである。Initial パケットは、Initial パケット保護キーを使用してのみ送信でき、Initial パケットでもあるパケットの中でのみ確認応答できる。同様に、Handshake パケットは Handshake 暗号化レベルで送信され、Handshake パケットでのみ確認応答できる。

これにより、異なるパケット番号空間で送信されるデータ間の暗号化による分離が強制される。各空間のパケット番号は、パケット番号 0 から始まる。同じパケット番号空間で送信される後続のパケットは、パケット番号を少なくとも 1 つ増やさなければならない [MUST]。

0-RTT データと 1-RTT データは、2 つのパケットタイプ間で損失回復アルゴリズムを実装しやすくするために、同じパケット番号空間に存在する。

QUIC エンドポイントは、あるコネクションで同じパケット番号空間内のパケット番号を再利用してはならない [MUST NOT]。送信用のパケット番号が 2^{62}-1 に達した場合、送信側は CONNECTION_CLOSE フレームまたはその他のパケットを送信せずにコネクションを閉じなければならない [MUST]。エンドポイントは、

受信したさらに他のパケットに応答して Stateless Reset パケット (第 10.3 節)を送信してもよい[MAY]。

受信側は、同じパケット番号空間から同じパケット番号を持つ別のパケットを処理していないことが確実でない限り、新しく保護されていないパケットを破棄しなければならない [MUST]。重複の抑制は、[QUIC-TLS] のセクション 9.5 で説明されている理由により、パケット保護を解除した後に行わなければならない [MUST]。

重複を検出する目的で個々のパケットをすべての追跡するエンドポイントは、過剰な状態を蓄積するリスクがある。重複を検出するために必要なデータは、すべてのパケットが直ちに破棄される最小パケット番号を維持することによって制限できる。どのような最小値であっても、ラウンドトリップ時間の大きな変動を考慮する必要があり、これには、ピアがより大きなラウンドトリップ時間を持つネットワークパスをプローブする可能性が含まれる(第 9 章)。

送信側でのパケット番号のエンコードと受信側でのデコードについては、第 17.1 節で説明されている。

12.4 フレームとフレームタイプ

パケット保護を削除した後の QUIC パケットのペイロードは、図 11 に示すように、完全なフレームのシーケンスで構成される。Version Negotiation パケット、Stateless Reset パケット、および Retry パケットにはフレームは含まれない。

```
Packet Payload {
  Frame (8..) ...,
}
```

図 11　QUIC ペイロード

フレームを含むパケットのペイロードには、少なくとも 1 つのフレームを含めなければならない[MUST]、また、複数のフレームと複数のフレームタイプが含んでもよい[MAY]。エンドポイントは、フレームを含まないパケットの受信を PROTOCOL_VIOLATION タイプのコネクションエラーとして処理しなければならない [MUST]。フレームは常に 1 つの QUIC パケット内に収まり、複数のパケットにまたがることはできない。

各フレームは、そのタイプを示す Frame Type フィールドで始まり、その後に追加のタイプ依存フィールドが続く。

```
Frame {
  Frame Type (i),
  Type-Dependent Fields (..),
}
```

図 12　汎用フレームレイアウト

表 3 は、この仕様で定義されている各フレームタイプに関する情報をリストし、要約したものである。この要約の説明は、表の後に記載されている。

表 3　フレームタイプ

タイプ値	フレームタイプ名	定義	Pkts	仕様
0x00	PADDING	第 19.1 節	IH01	NP
0x01	PING	第 19.2 節	IH01	
0x02-0x03	ACK	第 19.3 節	IH_1	NC
0x04	RESET_STREAM	第 19.4 節	__01	
0x05	STOP_SENDING	第 19.5 節	__01	
0x06	CRYPTO	第 19.6 節	IH_1	
0x07	NEW_TOKEN	第 19.7 節	___1	

0x08-0x0f	STREAM	第 19.8 節	__01	F
0x10	MAX_DATA	第 19.9 節	__01	
0x11	MAX_STREAM_DATA	第 19.10 節	__01	
0x12-0x13	MAX_STREAMS	第 19.11 節	__01	
0x14	DATA_BLOCKED	第 19.12 節	__01	
0x15	STREAM_DATA_BLOCKED	第 19.13 節	__01	
0x16-0x17	STREAMS_BLOCKED	第 19.14 節	__01	
0x18	NEW_CONNECTION_ID	第 19.15 節	__01	P
0x19	RETIRE_CONNECTION_ID	第 19.16 節	__01	
0x1a	PATH_CHALLENGE	第 19.17 節	__01	P
0x1b	PATH_RESPONSE	第 19.18 節	___1	P
0x1c-0x1d	CONNECTION_CLOSE	第 19.19 節	ih01	N
0x1e	HANDSHAKE_DONE	第 19.20 節	___1	

各フレームタイプのフォーマットとセマンティクスについては、第 19 章で詳しく説明する。本節の残りでは、重要かつ一般的な情報の概要を提供する。

ACK フレーム、STREAM フレーム、MAX_STREAMS フレーム、STREAMS_BLOCKED フレーム、および CONNECTION_CLOSE フレームの Frame Type フィールドは、他のフレーム固有のフラグを伝達するために使用される。その他のすべてのフレームでは、Frame Type フィールドは単にフレームを識別する。

表 3 の「Pkts」列には、各フレームタイプが出現する可能性のあるパケットのタイプが以下の文字で示されている。

I: Initial (第 17.2.2 項)

H: Handshake (第 17.2.4 項)

0: 0-RTT (第 17.2.3 項)

1: 1-RTT (第 17.3.1 項)

ih: Initial パケットまたは Handshake パケットに出現できるのは、タイプ 0x1c の CONNECTION_CLOSE フレームのみである。

これらの制限の詳細については、第 12.5 節を参照。すべてのフレームが 1-RTT パケットに出現できることに注意すること。エンドポイントは、許可されていないパケットタイプのフレームの受信を、PROTOCOL_VIOLATION タイプのコネクションエラーとして処理しなければならない [MUST]。

表 3 の「Spec」には、フレームタイプの処理または生成を制御する特別な規則が、以下の文字でまとめられている。

N: このマーキングの付いたフレームのみを含むパケットは、ack-eliciting パケットではない (第 13.2 節)。

C: このマーキングの付いたフレームのみを含むパケットは、輻輳制御のための bytes in flight にカウントされない [QUIC-RECOVERY]。

P: このマーキングが付いたフレームのみを含むパケットは、コネクション移行中に新しいネットワークパスをプローブするために使用できる(第 9.1 節)。

F: このマーキングが付いたフレームの内容はフロー制御される (第 4 章)。

表 3 の「Pkts」と「Spec」は IANA レジストリの一部ではない (第 22.4 節)。

エンドポイントは、不明なタイプのフレームの受信を FRAME_ENCODING_ERROR タイプのコネクションエラーとして処理しなければならない [MUST]。

このバージョンの QUIC では、すべてのフレームは冪等である。つまり、有効なフレームを複数回受信しても、望ましくない副作用やエラーは発生しない。

Frame Type フィールドは、可変長整数エンコーディングを使用する(第 16 章)が、例外がある。フレーム解析の単純かつ効率的な実装を保証するために、フレームタイプは可能な限り最短のエンコーディングを使用しなければならない [MUST]。この文書で定義されているフレームタイプでは、これらの値を 2 バイト、4 バイト、8 バイトの可変長整数としてエンコードすることが可能であっても、これはシングルバイトエンコーディングを意味する。たとえば、0x4001 は値が 1 の可変長整数の正当な 2 バイトエンコードであるが、PING フレームは常に値が 0x01 の値を持つ 1 バイトとしてエンコードされる。この規則は、現在および将来のすべての QUIC フレームタイプに適用される。エンドポイントは、必要以上に長いエンコードを使用するフレームタイプの受信を PROTOCOL_VIOLATION タイプのコネクションエラーとして扱うことができる [MAY]。

12.5　フレームと番号空間

一部のフレームは、異なるパケット番号空間で禁止されている。ここでの規則は TLS の規則を一般化したものであり、コネクションの確立に関連するフレームは通常、任意のパケット番号空間のパケットに出現可能であるが、データ転送に関連するフレームはアプリケーションデータパケット番号空間にのみ出現可能である。

- PADDING、PING、および CRYPTO フレームは、任意のパケット番号空間に出現する可能性がある [MAY]。
- QUIC 層でエラーを通知する CONNECTION_CLOSE フレーム (0x1c タイプ) は、任意のパケット番号空間に出現する可能性がある [MAY]。アプリケーションエラーを通知する CONNECTION_CLOSE フレーム (0x1d タイプ) は、アプリケーションデータパケット番号空間にのみ出現しなければならない [MUST]。
- ACK フレームは、任意のパケット番号空間に出現する可能性がある[MAY]が、そのパケット番号空間に出現したパケットのみを確認することしかできない。ただし、以下に示すように、0-RTT パケットには ACK フレームを含めることはできない。
- 他のすべてのフレームタイプは、アプリケーションデータパケット番号空間でのみ送信されなければならない [MUST]。

さまざまな理由により、ACK フレーム、CRYPTO フレーム、HANDSHAKE_DONE フレーム、NEW_TOKEN フレーム、PATH_RESPONSE フレーム、および RETIRE_CONNECTION_ID フレームを 0-RTT パケットで送信することはできないことに注意すること。サーバは、0-RTT パケットでこれらのフレームを受信した場合、PROTOCOL_VIOLATION タイプのコネクションエラーとして処理することができる [MAY]。

１３．　パケット化と信頼性

送信側は、QUIC パケットで 1 つ以上のフレームを送信する。第 12.4 節を参照。

送信側は、各 QUIC パケットに可能な限り多くのフレームを含めることによって、パケットごとの帯域幅と計算コストを最小限に抑えることができる。送信側は、大量の小さなパケットを送信しないように、最大にパックされていないパケットを送信する前に、複数のフレームを収集するために短時間待機してもかまわない[MAY]。実装は、アプリケーションの送信動作またはヒューリスティックに関する知識を使用して、待機するかどうかとその期間を決定してもかまわない[MAY]。この待機期間は実装の決定事項であり、遅延が発生するとアプリケーションに表示される待機時間が増加する可能性があるため、実装は控えめに遅延するように注意する必要がある。

ストリーム多重化は、複数のストリームからの STREAM フレームを 1 つ以上の QUIC パケットにインターリーブすることによって実現される。1 つの QUIC パケットには、1 つ以上のストリームからの複数の

STREAM フレームを含めることができる。

QUIC の利点の 1 つは、複数のストリームの先頭ブロックを回避することである。パケット損失が発生すると、そのパケット内のデータを含むストリームのみがブロックされ、再送信の受信を待機する。他のストリームは進行を続行できる。複数のストリームからのデータが 1 つの QUIC パケットに含まれている場合、そのパケットが失われると、すべてのストリームが進行をブロックされることに注意すること。実装では、不足しているパケットに送信効率を低下させることなく、送信パケットに必要な最小限のストリームを含めることを推奨する。

13.1 パケット処理

パケット保護が正常に削除され、パケットに含まれるすべてのフレームが処理されるまで、パケットは受信確認 してはならない[MUST NOT]。STREAM フレームの場合、これは、アプリケーションプロトコルによって受信される準備としてデータがエンキューされていることを意味するが、データを配信して消費する必要はない。

パケットが完全に処理されると、受信側は、受信したパケットのパケット番号を含む 1 つ以上の ACK フレームを送信することによって受信を確認する。

エンドポイントは、状態を検出できる場合、送信しなかったパケットの受信確認を PROTOCOL_VIOLATION タイプのコネクションエラーとして処理する必要がある。これを実現する方法の詳細については、第 21.4 節を参照。

13.2 確認応答の生成

エンドポイントは、受信して処理するすべてのパケットを確認する。ただし、ACK フレームが最大 ACK 遅延内で送信されるのは、ack-eliciting パケットのみである。ack-eliciting ではないパケットは、他の理由で ACK フレームが送信された場合にのみ確認される。

何らかの理由でパケットを送信する場合、エンドポイントは、ACK フレームが最近送信されていない場合は、ACK フレームを含めることを試みる必要がある。これは、ピアでのタイムリーな損失検出に役立つ。

一般に、受信側からの頻繁なフィードバックは損失と輻輳応答を改善するが、これは、すべての ack-eliciting パケットに応答して ACK フレームを送信する受信側によって生成される過剰な負荷とのバランスを取る必要がある。以下のガイダンスでは、このバランスを取ることを目指している。

13.2.1 ACK フレームの送信

すべてのパケットは、少なくとも一度は確認応答されるべきであり[SHOULD]、ack-eliciting パケットは、エンドポイントが max_ack_delay トランスポートパラメータを使用して通信する最大遅延内で、少なくとも一度は確認応答されなければならない[MUST]。第 18.2 節を参照。max_ack_delay は、明示的なコントラクトを宣言する。エンドポイントは、ACK 送信パケットの確認応答を、指定された値よりも意図的に遅延させないことを約束する。その場合、RTT 推定値に超過が発生し、ピアからの誤った再送信または遅延再送信が発生する可能性がある。送信側は、[QUIC-RECOVERY] の第 6.2 節で詳述されているように、受信側の max_ack_delay 値を使用して、タイマベースの再送信のタイムアウトを決定する。

エンドポイントは、通知された max_ack_delay 内のすべての ack-eliciting Initial パケットと Handshake パケット、およびすべての ack-eliciting 0-RTT および 1-RTT パケットを直ちに確認しなければならない[MUST]。ただし、次の例外がある。ハンドシェイク確認の前に、エンドポイントは、Handshake パケット、0-RTT パケット、または 1-RTT パケットを受信したときに復号化するためのパケット保護キーを持っていない可能性がある。したがって、必要なキーが使用可能になったときに、それらをバッファリングして確認応答する可能性がある。

ACK フレームのみを含むパケットは輻輳制御されないため、エンドポイントは、ack-eliciting パケットの

受信に応答して、そのようなパケットを複数送信してはならない[MUST NOT]。

エンドポイントは、受信したパケットの前にパケットギャップがある場合でも、non-ack-eliciting パケット以外のパケットを送信してはならない[MUST NOT]。これにより、コネクションがアイドル状態になることを防ぐ可能性がある、確認応答の無限フィードバックループが回避される。non-ack-eliciting パケットは、エンドポイントが他のイベントに応答して ACK フレームを送信するときに、最終的に確認応答される。

ACK フレームのみを送信するエンドポイントは、それらの確認応答が ack-eliciting フレームを含むパケットに含まれていない限り、ピアからの確認応答を受信しない。確認応答する新しい ACK 誘導パケットがある場合、エンドポイントは他のフレームと共に ACK フレームを送信する必要がある[SHOULD]。非 ACK 誘導パケットのみが確認応答される必要がある場合、エンドポイントは、ack-eliciting パケットが受信されるまで、送信フレームと共に ACK フレームを送信しないことを選択してもかまわない[MAY]。

non-ack-eliciting パケットのみを送信するエンドポイントは、確認応答を確実に受信するために、それらのパケットに時折 ack-eliciting フレームを追加することを選択してもかまわない。第 13.2.4 項を参照。その場合、エンドポイントは、確認応答の無限フィードバックループを回避するために、すべての non-ack-eliciting パケットで ack-eliciting フレームを送信してはならない[MUST NOT]。

送信側で損失検出を支援するために、エンドポイントは ack-eliciting パケットを受信したときに、遅延なく ACK フレームを生成して送信する必要がある[SHOULD]。

- 受信したパケットのパケット番号が、受信した別の ack-eliciting パケットよりも小さい場合。
- パケットのパケット番号が、受信した最大番号の ack-eliciting パケットよりも大きく、そのパケットとこのパケットの間に欠落しているパケットがある場合。

同様に、IP ヘッダの ECN Congestion Experienced (CE) コードポイントでマークされたパケットは、輻輳イベントに対するピアの応答時間を短縮するために、直ちに確認応答される必要がある[SHOULD]。

[QUIC-RECOVERY] のアルゴリズムは、上記のガイダンスに従わない受信側に対して回復力があることが期待される。ただし、実装は、エンドポイントによって行われたコネクションとネットワークの他のユーザーに対して、変更のパフォーマンスへの影響を慎重に考慮した後にのみ、これらの要件から逸脱する必要がある。

13.2.2 確認応答の頻度

受信側は、ack-eliciting パケットに応答して確認応答を送信する頻度を決定する。この決定には、トレードオフが含まれる。

エンドポイントは、損失を検出するためにタイムリーな確認応答に依存する。[QUIC-RECOVERY] の第 6 章を参照。 [QUIC-RECOVERY] の第 7 章で説明されているようなウィンドウベースの輻輳コントローラは、輻輳ウィンドウを管理するために確認応答に依存する。どちらの場合も、確認応答の遅延はパフォーマンスに悪影響を与える可能性がある。

一方、確認応答のみを伝送するパケットの頻度を減らすと、両方のエンドポイントでパケットの送信と処理のコストが削減される。非常に非対称なリンクでのコネクションスループットを向上させ、リターンパスの容量を使用して確認応答トラフィックの量を減らすことができる。[RFC3449] の第 3 章を参照。

受信側は、少なくとも 2 つの ack-eliciting パケットを受信した後に ACK フレームを送信すべきである[SHOULD]。この推奨事項は一般的であり、TCP エンドポイントの動作に関する推奨事項 [RFC5681] と一致している。ネットワーク条件の知識、ピアの輻輳コントローラの知識、またはさらなる調査と実験によって、より優れたパフォーマンス特性を持つ代替の確認応答戦略が提案される場合がある。

受信側は、応答で ACK フレームを送信するかどうかを決定する前に、複数の利用可能なパケットを処理してもよい[MAY]。

13.2.3 ACK Range の管理

ACK フレームが送信されると、確認済みパケットの 1 つ以上の範囲が含まれる。古いパケットの確認応答を含めると、以前に送信された ACK フレームが失われることによって、より大きな ACK フレームを犠牲にして誤った再送信が行われる可能性が低くなる。

ACK フレームは、最後に受信したパケットを常に確認する必要がある[SHOULD]。また、パケットの順序が正しくないほど、更新された ACK フレームを迅速に送信することが重要になる。これは、ピアがパケットを失われたと宣言し、含まれているフレームを誤って再送信することを防ぐためである。ACK フレームは、単一の QUIC パケット内に収まることが期待される。そうでない場合は、古い範囲 (パケット番号が最も小さい範囲) は省略される。

受信側は、ACK フレームのサイズを制限し、リソースの枯渇を回避するために、ACK フレームで記憶し送信する ACK Range (第 19.3.1 項) の数を制限する。ACK フレームの確認応答を受信した後、受信側は確認された ACK Range の追跡を停止すべきである[SHOULD]。送信側はほとんどのパケットの確認応答を期待できるが、QUIC は受信側が処理するすべてのパケットの確認応答の受信を保証しない。

多くの ACK Range を保持すると、ACK フレームが大きくなりすぎる可能性がある。受信側は、受信確認されていない ACK Range を破棄して、送信側からの再送信が増加するという代償を払って、ACK フレームのサイズを制限することができる。これは、ACK フレームが大きすぎてパケットに収まらない場合に必要である。受信側は、ACK フレームのサイズをさらに制限して、他のフレーム用のスペースを確保したり、受信確認が消費する容量を制限したりすることもできる[MAY]。

受信側は、その後その範囲内の番号を持つパケットを受け入れないことを保証できない限り、ACK Range を保持しなければならない[MUST]。範囲が破棄されるにつれて増加する最小のパケット番号を維持することは、最小の状態でこれを達成する一つの方法である。

受信側はすべての ACK Range を破棄することができるが、正常に処理された最大のパケット番号を保持しなければならない[MUST]。これは、後続のパケットからパケット番号を回復するために使用されるからである。第 17.1 節を参照。

受信側は、すべての ACK フレームに最大の受信パケット番号を含む ACK Range を含めるべきである[SHOULD]。Largest Acknowledged フィールドは送信側で ECN 検証に使用され、前の ACK フレームに含まれていたものよりも小さい値を含めると ECN が不必要に無効になる可能性がある。第 13.4.2 項を参照。

第 13.2.4 項では、各 ACK フレームでどのパケットを確認するかを決定するための例示的なアプローチについて説明する。このアルゴリズムの目標は、処理されるすべてのパケットに対して確認応答を生成することだが、確認応答が失われる可能性がある。

13.2.4 ACK フレームの追跡による範囲の制限

ACK フレームを含むパケットが送信されると、そのフレームの Largest Acknowledged フィールドを保存できる。ACK フレームを含むパケットが確認されると、受信側は、送信された ACK フレームの Largest Acknowledged フィールド以下のパケットの確認応答を停止できる。

ACK フレームなど、non-ack-eliciting パケットのみを送信する受信側は、長時間確認応答を受信しない可能性がある。これにより、受信側は長時間にわたって多数の ACK フレームの状態を維持し、送信する ACK フレームが不必要に大きくなる可能性がある。このような場合、受信側は、ピアから ACK を取得するために、PING またはその他の小さな ack-eliciting フレームを、ラウンドトリップごとに 1 回などの頻度で送信できる。

ACK フレームの損失がない場合、このアルゴリズムでは最低 1 RTT の並べ替えが可能である。ACK フレームの損失と並べ替えがある場合、この方法では、ACK フレームに含まれなくなる前に送信側がすべての受信確認を確認することは保証されない。パケットは順序どおりに受信されず、それらを含む後続のすべての ACK フレームが失われる可能性がある。この場合、損失回復アルゴリズムによって誤った再送信が発生する

可能性があるが、送信側は伝送を続ける。

13.2.5 ホスト遅延の測定と報告

エンドポイントは、最大のパケット番号を持つパケットが受信されてから確認応答が送信されるまでの間に意図的に導入された遅延を測定する。エンドポイントは、この確認応答遅延を ACK フレームの ACK Delay フィールドにエンコードする。第 19.3 節を参照。これにより、ACK フレームの受信側は意図的な遅延を調整できる。これは、確認応答が遅延した場合にパス RTT のより良い推定値を取得するために重要である。

パケットは、処理される前に OS カーネルまたはホスト上の他の場所に保持される可能性がある。エンドポイントは、ACK フレームの ACK Delay フィールドを設定するときに、制御しない遅延を含めてはならない[MUST NOT]。しかし、エンドポイントは、復号キーが利用できないことによるバッファリング遅延を含めるべきである。なぜなら、これらの遅延は大きくなる可能性があり、繰り返さない可能性が高いからである。

測定された確認応答遅延が max_ack_delay より大きい場合、エンドポイントは測定された遅延を報告すべきである[SHOULD]。この情報は、遅延が大きくなる可能性があるハンドシェイク中に特に有用である。第 13.2.1 項を参照。

13.2.6 ACK フレームとパケット保護

ACK フレームは、確認応答されるパケットと同じパケット番号空間を持つパケットでのみ伝送されなければならない[MUST]。第 12.1 節を参照。例えば、1-RTT キーで保護されているパケットは、同じく 1-RTT キーで保護されているパケットで確認応答されなければならない[MUST]。

クライアントが 0-RTT パケット保護で送信するパケットは、1-RTT キーで保護されているパケットでサーバによって確認応答されなければならない[MUST]。これは、サーバの暗号化ハンドシェイクメッセージが遅延または消失した場合、クライアントがこれらの確認応答を使用できないことを意味する。1-RTT キーで保護されているサーバによって送信される他のデータにも同じ制限が適用されることに注意すること。

13.2.7 PADDING フレーム消費輻輳ウィンドウ

PADDING フレームを含むパケットは、輻輳制御 [QUIC-RECOVERY] のために伝送中であるとみなされる。したがって、PADDING フレームのみを含むパケットは輻輳ウィンドウを消費するが、輻輳ウィンドウを開く確認応答を生成しない。デッドロックを回避するために、送信側は PADDING フレームに加えて、定期的に他のフレームを送信し、受信側から確認応答を引き出すように努めるべきである[SHOULD]。

13.3 情報の再送信

失われたと判断された QUIC パケットは、完全には再送信されない。失われたパケットに含まれるフレームにも同じことが当てはまる。代わりに、フレームで伝送される可能性のある情報は、必要に応じて新しいフレームで再送信される。

新しいフレームとパケットは、失われたと判断された情報を伝送するために使用される。一般に、情報を含むパケットが失われたと判断されたときに情報が再送信され、その情報を含むパケットが確認応答されたときに送信が停止される。

- CRYPTO フレームで送信されたデータは、すべてのデータが確認応答されるまで、[QUIC-RECOVERY] の規則に従って再送信される。Initial パケットおよび Handshake パケットの CRYPTO フレーム内のデータは、対応するパケット番号空間のキーが破棄された時点で破棄される。
- STREAM フレームで送信されたアプリケーションデータは、エンドポイントがそのストリームに対して RESET_STREAM フレームを送信していない限り、新しい STREAM フレームで再送信される。エンドポイントが RESET_STREAM フレームを送信すると、それ以上の STREAM フレームは必要ない。

- ACK フレームは、第 13.2.1 項で説明されているように、最新の確認応答のセットと、最大の確認応答パケットからの確認応答遅延を伝送する。ACK フレームを含むパケットの送信を遅らせたり、古い ACK フレームを再送信したりすると、ピアが RTT サンプルを大きくしたり、ECN を不必要に無効にしたりする可能性がある。
- RESET_STREAM フレームで伝送されるストリーム伝送の取り消しは、受信が確認されるまで、またはすべてのストリームデータがピアによって確認されるまで送信される(ストリームの送信側で「Reset Recvd」または「Data Recvd」のいずれかの状態に達した場合)。RESET_STREAM フレームの内容は、再送信時に変更してはならない[MUST NOT]。
- 同様に、STOP_SENDING フレームでエンコードされたストリーム伝送の取り消し要求は、ストリームの受信側が"Data Recvd"または"Reset Recvd"状態になるまで送信される。第 3.5 節を参照。
- CONNECTION_CLOSE フレームを含むパケットによるコネクションクローズ信号は、パケット損失が検出されても再送信されない。これらの信号の再送信については、第 10 章で説明する。
- 現在の最大コネクションデータは MAX_DATA フレームで送信される。最後に送信された MAX_DATA フレームを含むパケットが失われたと宣言された場合、またはエンドポイントが制限を更新することを決定した場合は、更新された値が MAX_DATA フレームで送信される。制限が頻繁に増加し、不必要に多数の MAX_DATA フレームが送信される可能性があるため、このフレームを頻繁に送信しないように注意する必要がある。第 4.2 節を参照。
- 現在の最大ストリームデータオフセットは、MAX_STREAM_DATA フレームで送信される。MAX_DATA と同様に、ストリームの最新の MAX_STREAM_DATA フレームを含むパケットが失われたとき、または制限が更新されたときに、フレームが頻繁に送信されないように注意して、更新された値が送信される。エンドポイントは、ストリームの受信部分が"Size Known"または"Reset Recvd"状態になったときに、MAX_STREAM_DATA フレームの送信を停止する必要がある[SHOULD]。
- 特定の種類のストリームの制限は、MAX_STREAMS フレームで送信される。MAX_DATA と同様に、ストリームの種類のフレームの最新の MAX_STREAMS を含むパケットが失われたと宣言されたとき、または制限が更新されたときに、フレームが頻繁に送信されないように注意して送信される。
- ブロックされたシグナルは、DATA_BLOCKED、STREAM_DATA_BLOCKED、および STREAMS_BLOCKED フレームで伝送される。DATA_BLOCKED フレームにはコネクションスコープがあり、STREAM_DATA_BLOCKED フレームにはストリームスコープがあり、STREAMS_BLOCKED フレームのスコープは特定のストリームの種類である。新しいフレームは、スコープの最新のフレームを含むパケットが失われた場合に送信されるが、エンドポイントが対応する制限でブロックされている間にのみ送信される。これらのフレームには、送信時にブロックの原因となっている制限が常に含まれる。
- PATH_CHALLENGE フレームを使用した死活監視またはパス検証は、一致する PATH_RESPONSE フレームが受信されるまで、または死活監視またはパス検証が不要になるまで、定期的に送信される。PATH_CHALLENGE フレームには、送信されるたびに異なるペイロードが含まれる。
- PATH_RESPONSE フレームを使用したパス検証に対する応答は、1 回だけ送信される。ピアは、追加の PATH_RESPONSE フレームを呼び出すために、必要に応じてさらに PATH_CHALLENGE フレームを送信する必要がある。
- 新しいコネクション ID は NEW_CONNECTION_ID フレームで送信され、それらを含むパケットが失われた場合に再送信される。このフレームの再送信では、同じシーケンス番号値が伝送される。同様に、廃止されたコネクション ID は RETIRE_CONNECTION_ID フレームで送信され、それらを含むパケットが失われた場合に再送信される。
- NEW_TOKEN フレームは、それらを含むパケットが失われた場合に再送信される。フレームの内容を

直接比較する以外に、並べ替えられた NEW_TOKEN フレームと重複した NEW_TOKEN フレームを検出するための特別なサポートはない。

- PING および PADDING フレームには情報が含まれていないため、失われた PING または PADDING フレームを修復する必要はない。
- HANDSHAKE_DONE フレームは、確認応答があるまで再送信されなければならない[MUST]。

アプリケーションによって指定された優先順位でない限り、エンドポイントは新しいデータの送信よりもデータの再送信を優先する必要がある[SHOULD]。第 2.3 節を参照。

送信側は、パケットを送信するたびに最新の情報を含むフレームを組み立てることが推奨されているが、失われたパケットからフレームのコピーを再送信することは禁止されていない。フレームのコピーを再送信する送信側は、パケット番号の長さ、コネクション ID の長さ、およびパス MTU の変更による使用可能なペイロードサイズの減少を処理する必要がある。受信側は、古いパケットで見つかった最大データ値よりも小さい MAX_DATA フレームなど、古いフレームを含むパケットを受け入れなければならない[MUST]。

送信側は、確認されたパケットからの情報の再送信を避ける必要がある[SHOULD]。これには、ネットワークの順序変更が発生した場合に、失われたと宣言された後に確認されたパケットが含まれる。これを行うには、パケットが失われたと宣言された後、送信側がパケットに関する情報を保持する必要がある。送信側は、PTO ([QUIC-RECOVERY] の第 6.2 節) などの並べ替えを十分に可能にする期間が経過した後、またはメモリ制限に到達するなどの他のイベントに基づいて、この情報を破棄できる。

損失を検出した場合、送信側は適切な輻輳制御アクションを実行する必要がある[MUST]。損失検出と輻輳制御の詳細については、[QUIC-RECOVERY] で説明されている。

13.4 明示的な輻輳通知

QUIC エンドポイントは、ECN [RFC3168] を使用してネットワーク輻輳を検出し、応答できる。ECN を使用すると、エンドポイントは IP パケットの ECN フィールドに ECN 対応トランスポート (ECT) コードポイントを設定できる。ネットワークノードは、パケットを破棄する代わりに ECN フィールドに ECN-CE コードポイントを設定することで輻輳を示すことができる [RFC8087] 。エンドポイントは、[QUIC-RECOVERY] で説明されているように、応答で送信レートを下げることによって、報告された輻輳に対応する。

ECN を有効にするために、送信 QUIC エンドポイントはまず、パスが ECN マーキングをサポートしているかどうか、およびピアが受信した IP ヘッダの ECN 値を報告するかどうかを判断する。第 13.4.2 項を参照。

13.4.1 ECN カウントの報告

ECN を使用するには、受信側エンドポイントが IP パケットから ECN フィールドを読み取る必要があるが、これはすべてのプラットフォームで可能ではない。エンドポイントが ECN サポートを実装していない場合、または受信した ECN フィールドにアクセスできない場合、受信したパケットの ECN カウントは報告されない。

エンドポイントが送信するパケットに ECT フィールドを設定していない場合でも、アクセス可能な場合、エンドポイントは受信した ECN マーキングに関するフィードバックを提供する必要がある[MUST]。ECN カウントを報告しないと、送信側はこのコネクションに対する ECN の使用を無効にする。

ECT (0)、ECT (1)、または ECN-CE コードポイントを含む IP パケットを受信すると、ECN 対応エンドポイントは ECN フィールドにアクセスし、対応する ECT (0)、ECT (1)、または ECN-CE カウントを増やす。これらの ECN カウントは後続の ACK フレームに含まれる。第 13.2 節および第 19.3 節を参照。

各パケット番号空間は、個別の確認応答状態と個別の ECN カウントを保持する。結合された QUIC パケット(第 12.2 節を参照)は同じ IP ヘッダを共有するため、ECN カウントは結合された QUIC パケットごとに 1 回ずつ増加する。

たとえば、Initial、Handshake、および 1-RTT QUIC パケットのそれぞれが単一の UDP データグラムに結合された場合、3 つのパケット番号空間すべての ECN カウントは、単一の IP ヘッダの ECN フィールドに基づいてそれぞれ 1 ずつ増加する。

ECN カウントは、受信した IP パケットからの QUIC パケットが処理されるときにのみ増加する。そのため、重複した QUIC パケットは処理されず、ECN カウントは増加しない。関連するセキュリティの問題については、第 21.10 節を参照。

13.4.2 ECN 検証

障害のあるネットワークデバイスが、ゼロ以外の ECN コードポイントを持つパケットを破損または誤って破棄する可能性がある。このようなデバイスが存在する場合のコネクションを保証するために、エンドポイントは各ネットワークパスの ECN カウントを検証し、エラーが検出された場合はそのパスでの ECN の使用を無効にする。

新しいパスの ECN 検証を実行するには、次の手順を実行する。

- エンドポイントは、ピアへの新しいパスで送信される初期の送信パケットの IP ヘッダに ECT (0) コードポイントを設定する。
- エンドポイントは、ECT コードポイントで送信されたすべてのパケットが最終的に失われたと見なされ ([QUIC-RECOVERY] の第 6 章)、ECN 検証が失敗したことを示すかどうかを監視する。

エンドポイントに、ECT コードポイントを持つ IP パケットが障害のあるネットワーク要素によって破棄される可能性があると予想される理由がある場合、エンドポイントは、パス上の最初の 10 個の発信パケットに対してのみ、または 3 つの PTO の期間に対して ECT コードポイントを設定できる ([QUIC-RECOVERY] の第 6.2 節を参照) 。ゼロ以外の ECN コードポイントでマークされたすべてのパケットがその後失われた場合、マーキングが損失の原因であると仮定してマーキングを無効にすることができる。

したがって、エンドポイントは ECN の使用を試み、サーバの優先アドレスへの切り替え時、および新しいパスへのアクティブなコネクションの移行時に、新しいコネクションごとにこれを検証する。付録 A.4 では、1 つの可能なアルゴリズムについて説明する。

ECN サポートのパスをプローブする他の方法も、さまざまなマーキング戦略と同様に可能である[MAY]。実装では、RFC で定義されている他の方法を使用してもかまわない。[RFC8311] を参照。ECT (1) コードポイントを使用する実装では、報告された ECT (1) カウントを使用して ECN 検証を実行する必要がある。

13.4.2.1 ECN カウントを含む ACK フレームの受信

ネットワークによる ECN-CE マーキングの誤った適用により、コネクションパフォーマンスが低下する可能性がある。したがって、ECN カウントを含む ACK フレームを受信するエンドポイントは、使用する前にカウントを検証する。この検証は、新しく受信したカウントと、最後に正常に処理された ACK フレームのカウントを比較することによって実行される。ECN カウントの増加は、ACK フレームで新しく確認されたパケットに適用された ECN マーキングに基づいて検証される。

エンドポイントが ECT (0) または ECT (1) コードポイントセットで送信したパケットを ACK フレームが新たに確認応答した場合、対応する ECN カウントが ACK フレームに存在しないと ECN 検証は失敗する。このチェックは、ECN フィールドをゼロにするネットワーク要素、または ECN マーキングを報告しないピアを検出する。

また、ECT (0) カウントと ECN-CE カウントの増加の合計が、最初に ECT (0) マーキングで送信された新たに確認応答されたパケットの数よりも少ない場合も、ECN 検証は失敗する。同様に、ECT (1) カウントと ECN-CE カウントの増加の合計が、ECT (1) マーキングで送信された新たに確認応答されたパケットの数よりも少ない場合、ECN 検証は失敗する。これらのチェックでは、ネットワークによる ECN-CE マーキングの

再マーキングを検出できる。

ACK フレームが失われた場合、エンドポイントはパケットの確認応答を見逃す可能性がある。したがって、ECT (0)、ECT (1)、および ECN-CE カウントの合計増加数が、ACK フレームによって新たに確認応答されたパケットの数よりも大きくなる可能性がある。これは、ECN カウントが確認応答されたパケットの合計数よりも大きくなることが許可される理由である。

並べ替えられた ACK フレームから ECN カウントを検証すると、失敗する可能性がある。エンドポイントは、確認応答された最大パケット数が増加しない ACK フレームを処理した結果として ECN 検証に失敗してはならない[MUST NOT]。

ECT (0) または ECT (1) の受信合計数が、対応する各 ECT コードポイントで送信されたパケットの合計数を超えた場合、ECN 検証は失敗する可能性がある。特に、エンドポイントが、適用されていない ECT コードポイントに対応する 0 以外の ECN カウントを受信した場合、検証は失敗する。このチェックは、パケットがネットワーク内で ECT (0) または ECT (1) に再マークされたことを検出する。

13.4.2.2 ECN 検証の結果

検証が失敗した場合、エンドポイントは ECN を無効にする必要がある[MUST]。ネットワークパスまたはピアが ECN をサポートしていないと想定して、送信する IP パケットの ECT コードポイントの設定を停止する。

検証が失敗した場合でも、エンドポイントは、コネクションの後の任意の時点で同じパスの ECN を再検証することができる[MAY]。エンドポイントは、定期的に検証を試行し続けることができる。

検証が成功した場合、エンドポイントは、パスが ECN 対応であることを想定して、送信する後続のパケットで ECT コードポイントを設定し続けることができる[MAY]。ネットワークルーティングとパス要素は接続中に変更可能;エンドポイントは、後で検証が失敗した場合に ECN を無効にしなければならない[MUST]。

１４． データグラムサイズ

UDP データグラムには、1 つ以上の QUIC パケットを含めることができる。データグラムサイズとは、QUIC パケットを伝送する単一の UDP データグラムの UDP ペイロードサイズの合計を指す。データグラムサイズには、1 つ以上の QUIC パケットヘッダと保護されたペイロードが含まれるが、UDP または IP ヘッダは含まれない。

最大データグラムサイズは、単一の UDP データグラムを使用してネットワークパスを介して送信できる UDP ペイロードの最大サイズとして定義される。ネットワークパスが少なくとも 1200 バイトの最大データグラムサイズをサポートできない場合は、QUIC を使用してはならない[MUST NOT]。

QUIC は、少なくとも 1280 バイトの最小 IP パケットサイズを想定している。これは IPv6 の最小サイズ[IPv6] であり、ほとんどの最新の IPv4 ネットワークでもサポートされている。IPv6 の最小 IP ヘッダサイズを 40 バイト、IPv4 の最小 IP ヘッダサイズを 20 バイトと仮定し、UDP ヘッダサイズを 8 バイトとすると、最大データグラムサイズは IPv6 で 1232 バイト、IPv4 で 1252 バイトになる。したがって、最新の IPv4 およびすべての IPv6 ネットワークパスで QUIC をサポートできることが期待される。

> 注意:1200 バイトの UDP ペイロードをサポートするためのこの要件により、IPv6 拡張ヘッダに使用できる領域が 32 バイトに制限され、パスが 1280 バイトの IPv6 最小 MTU のみをサポートする場合は IPv4 オプションが 52 バイトに制限される。これは、Initial パケットとパス検証に影響する。

1200 バイトを超える最大データグラムサイズは、Path Maximum Transmission Unit Discovery (PMTUD) (第 14.2.1 項を参照) または Datagram Packetization Layer PMTU Discovery (DPLPMTUD) (第 14.3 節を参照)を使用して検出できる。

max_udp_payload_size トランスポートパラメータ(第 18.2 節)の適用は、最大データグラムサイズに対する

追加の制限として機能する可能性がある。値がわかっていれば、送信側はこの制限を超えることを回避できる。ただし、トランスポートパラメータの値を学習する前に、エンドポイントが許容される最小最大データグラムサイズである 1200 バイトを超えるデータグラムを送信すると、データグラムが失われる危険がある。

UDP データグラムは、IP 層でフラグメント化されてはならない[MUST NOT]。IPv4 [IPv4] では、可能であれば、パスのフラグメント化を防ぐために、Don't Fragment (DF) ビットを設定しなければならない[MUST]。

QUIC では、データグラムが特定のサイズ以上であることが要求される場合がある。例として第 8.1 節を参照。ただし、データグラムのサイズは認証されない。つまり、エンドポイントが特定のサイズのデータグラムを受信した場合、送信側が同じサイズのデータグラムを送信したことを知ることはできない。したがって、サイズ制約を満たさないデータグラムを受信した場合、エンドポイントはコネクションを閉じてはならない[MUST NOT]。エンドポイントはそのようなデータグラムを破棄してもよい[MAY]。

14.1 初期データグラムサイズ

クライアントは、Initial パケットに PADDING フレームを追加するか、Initial パケットを結合することによって、Initial パケットを運ぶすべての UDP データグラムのペイロードを少なくとも最小の許容最大データグラムサイズである 1200 バイトに拡張しなければならない[MUST]。第 12.2 節を参照。Initial パケットは、受信側が破棄する無効なパケットと結合することさえできる。同様に、サーバは、ack-eliciting Initial パケットを運ぶすべての UDP データグラムのペイロードを少なくとも最小の許容最大データグラムサイズである 1200 バイトに拡張しなければならない[MUST]。

このサイズの UDP データグラムを送信すると、ネットワークパスが適切なパスをサポートすることが保証される。

両方向の最大伝送ユニット (PMTU) 、さらに Initial パケットを拡張するクライアントは、検証されていないクライアントアドレスへのサーバ応答によって引き起こされる増幅攻撃の振幅を減らすのに役立つ。第 8 章を参照。

Initial パケットを含むデータグラムは、送信側がネットワークパスとピアの両方が選択したサイズをサポートしていると確信している場合、1200 バイトを超える可能性がある。

サーバは、1200 バイトの最小許容最大データグラムサイズよりも小さいペイロードを持つ UDP データグラムで伝送される ack-eliciting Initial パケットを破棄しなければならない[MUST]。また、サーバは PROTOCOL_VIOLATION のエラーコードを含む CONNECTION_CLOSE フレームを送信することによって、直ちにコネクションを閉じてもよい[MAY]。第 10.2.3 項を参照。

サーバはまた、クライアントのアドレスを検証する前に送信するバイト数を制限しなければならない[MUST]。第 8 章を参照。

14.2 パス最大送信単位

PMTU は、IP ヘッダ、UDP ヘッダ、および UDP ペイロードを含む IP パケット全体の最大サイズである。UDP ペイロードには、1 つ以上の QUIC パケットヘッダと保護されたペイロードが含まれる。PMTU はパスの特性に依存する可能性があるため、時間の経過と共に変化する可能性がある。エンドポイントが任意の時点で送信する最大の UDP ペイロードは、エンドポイントの最大データグラムサイズと呼ばれる。

エンドポイントは、DPLPMTUD (第 14.3 節)または PMTUD (第 14.2.1 項) を使用して、宛先へのパスが断片化なしで必要な最大データグラムサイズをサポートするかどうかを判断する必要がある[SHOULD]。これらのメカニズムがない場合、QUIC エンドポイントは、許可されている最小の許容最大データグラムサイズよりも大きいデータグラムを送信するべきではない[SHOULD NOT]。

DPLPMTUD と PMTUD はどちらも、PMTU プローブと呼ばれる、現在の最大データグラムサイズよりも大きいデータグラムを送信する。PMTU プローブで送信されないすべての QUIC パケットは、データグラムがフラグメント化または破棄されないように、最大データグラムサイズ内に収まるようにサイズ設定する必

要がある[SHOULD] [RFC8085] 。

QUIC エンドポイントが、ローカル IP アドレスとリモート IP アドレスのペア間の PMTU が最小の最大許容データグラムサイズである 1200 バイトをサポートできないと判断した場合、影響を受けるパス上で、PMTU プローブ内のパケットまたは CONNECTION_CLOSE フレームを含むパケットを除き、QUIC パケットの送信を直ちに停止しなければならない[MUST]。代替パスが見つからない場合、エンドポイントはコネクションを終了してもかまわない[MAY]。

ローカルアドレスとリモートアドレスの各ペアは、異なる PMTU を持つことができる。したがって、任意の種類の PMTU 検出を実装する QUIC 実装では、ローカル IP アドレスとリモート IP アドレスの組み合わせごとに最大データグラムサイズを維持する必要がある[SHOULD]。

QUIC 実装では、不明なトンネルオーバーヘッドまたは IP ヘッダオプション/拡張を可能にするために、最大データグラムサイズの計算をより慎重に行うことができる[MAY]。

14.2.1 PMTUD による ICMP メッセージの処理

PMTUD [RFC1191] [RFC8201] は、IP パケットがローカルルーターの MTU より大きいために破棄されたことを示す ICMP メッセージ(IPv6 Packet Too Big (PTB) メッセージ)の受信に依存している。DPLPMTUD は、オプションでこれらのメッセージを使用することもできる。ICMP メッセージのこの使用は、パケットを監視できないエンティティによる攻撃に対して潜在的に脆弱だが、パスで使用されているアドレスを正常に推測できる可能性がある。これらの攻撃は、PMTU を帯域幅効率の高い値に減らす可能性がある。

エンドポイントは、PMTU が QUIC の最小許容最大データグラムサイズを下回ったことを要求する ICMP メッセージを無視する必要がある[MUST]。

ICMP [RFC1812] [RFC4443] を生成するための要件は、引用符で囲まれたパケットが IP バージョンの最小 MTU を超えずに可能な限り多くの元のパケットを含むべきであると述べている。引用符で囲まれたパケットのサイズは、[DPLPMTUD] の第 1.1 節で説明されているように、実際には小さくなるか、情報が理解できなくなる可能性がある。

PMTUD を使用する QUIC エンドポイントは、[RFC8201] および [RFC8085] の第 5.2 節で指定されているように、パケットインジェクションから保護するために ICMP メッセージを検証すべきである[SHOULD]。この検証は、ICMP メッセージのペイロードで提供される引用符で囲まれたパケットを使用して、メッセージを対応するトランスポートコネクションに関連付けるべきである[SHOULD] ([DPLPMTUD] の第 4.6.1 項を参照) 。ICMP メッセージの検証には、一致する IP アドレスと UDP ポート [RFC8085]、および可能な場合はアクティブな QUIC セッションへのコネクション ID を含めなければならない[MUST]。エンドポイントは、検証に失敗したすべての ICMP メッセージを無視すべきである[SHOULD]。

エンドポイントは、ICMP メッセージに基づいて PMTU を増やしてはならない[MUST NOT]。[DPLPMTUD] の第 3 章の項目 6 を参照。ICMP メッセージに応答した QUIC の最大データグラムサイズの削減は、QUIC の損失検出アルゴリズムが引用符で囲まれたパケットが実際に失われたことを判断するまで暫定的なものである可能性がある[MAY]。

14.3 データグラムパケット化層 PMTU 検出

DPLPMTUD [DPLPMTUD] は、PMTU プローブで伝送される QUIC パケットの損失または確認応答の追跡に依存している。PADDING フレームを使用する DPLPMTUD の PMTU プローブは、[DPLPMTUD] の第 4.1 節で定義されているように、"埋め込みデータを使用したプローブ"を実装する。

エンドポイントは、QUIC で許可されている最小の最大データグラムサイズと一致するように、BASE_PLPMTU の初期値 ([DPLPMTUD] の第 5.1 節) を設定する必要がある[SHOULD]。MIN_PLPMTU は、BASE_PLPMTU と同じである。

DPLPMTUD を実装する QUIC エンドポイントは、ローカル IP アドレスとリモート IP アドレスの組み合わ

せごとに、DPLPMTUD の最大パケットサイズ (MPS) ([DPLPMTUD] の第 4.4 節) を維持する。これは、最大データグラムサイズに対応する。

14.3.1 DPLPMTUD と初期コネクション

DPLPMTUD の観点からは、QUIC は確認済みパケット化層 (PL) である。そのため、QUIC 送信側は、QUIC コネクションハンドシェイクが完了したときに DPLPMTUD BASE 状態 ([DPLPMTUD] の第 5.2 節) に入ることができる。

14.3.2 DPLPMTUD によるネットワークパスの検証

QUIC は確認済み PL である。そのため、QUIC 送信側は、SEARCH_COMPLETE 状態の間は DPLPMTUD CONFIRMATION_TIMER を実装しない。[DPLPMTUD] の第 5.2 節を参照。

14.3.3 DPLPMTUD による ICMP メッセージの処理

DPLPMTUD を使用するエンドポイントでは、[DPLPMTUD] の第 4.6 節で定義されているように、PTB 情報を使用する前に、受信した ICMP PTB メッセージの検証が必要である。UDP ポートの検証に加えて、QUIC は他の PL 情報(例えば、受信した ICMP メッセージの引用パケット中のコネクション ID の検証)を使用して ICMP メッセージを検証する。

第 14.2.1 項で説明されている ICMP メッセージの処理に関する考慮事項は、これらのメッセージが DPLPMTUD によって使用される場合にも適用される。

14.4 QUIC PMTU プローブの送信

PMTU プローブは ack-eliciting パケットである。

現在の最大データグラムサイズよりも大きいパケットはネットワークによって破棄される可能性が高いため、エンドポイントは PMTU プローブの内容を PING および PADDING フレームに制限できる。したがって、PMTU プローブで伝送される QUIC パケットの損失は、輻輳の信頼できる指標ではなく、輻輳制御反応をトリガにするべきではない[SHOULD NOT]。[DPLPMTUD] の第 3 章の項目 7 を参照。ただし、PMTU プローブは輻輳ウィンドウを消費するため、アプリケーションによる後続の送信が遅れる可能性がある。

14.4.1 Source Connection ID を含む PMTU プローブ

着信 QUIC パケットをルーティングするために Destination Connection ID フィールドに依存するエンドポイントは、生成される ICMP メッセージ (第 14.2.1 項) を正しいエンドポイントにルーティングするために、コネクション ID が PMTU プローブに含まれていることを要求する可能性がある。ただし、Source Connection ID フィールドが含まれるのはロングヘッダパケット(第 17.2 節)だけであり、ロングヘッダパケットは、ハンドシェイクが完了してもピアによって復号化または確認応答されない。

PMTU プローブを作成する 1 つの方法は、Handshake パケットや 0-RTT パケット(第 17.2 節)などのロングヘッダを持つパケットを、単一の UDP データグラム内のショートヘッダパケットと結合(第 12.2 節)することである。結果の PMTU プローブがエンドポイントに到達すると、ロングヘッダを持つパケットは無視されるが、ショートヘッダのパケットは確認応答される。PMTU プローブによって ICMP メッセージが送信される場合、そのメッセージではプローブの最初の部分が引用符で囲まれる。Source Connection ID フィールドがプローブの引用符で囲まれた部分内にある場合は、ICMP メッセージのルーティングまたは検証に使用できる。

注意:ロングヘッダを持つパケットを使用する目的は、ICMP メッセージに含まれる引用符で囲まれたパケットに Source Connection ID フィールドが含まれていることを確認することだけである。このパケットは有効なパケットである必要はなく、その種類のパケットが現在使用されていない場合でも送信でき

る。

１５．　バージョン

QUIC バージョンは、32 ビットの符号なし番号を使用して識別される。

バージョン 0x00000000 は、バージョンネゴシエーションを表すために予約されている。この仕様書のバージョンは、番号 0x00000001 によって識別される。

QUIC の他のバージョンは、このバージョンとは異なるプロパティを持つ場合がある。プロトコルのすべてのバージョンで一貫性が保証される QUIC のプロパティについては、[QUIC-INVARIANTS] を参照。

QUIC のバージョン 0x00000001 は、[QUIC-TLS] で説明されているように、暗号ハンドシェイクプロトコルとして TLS を使用する。

バージョン番号の最上位 16 ビットがクリアされているバージョンは、将来の IETF コンセンサスドキュメントで使用するために予約されている。

パターン 0x?a?a?a?a は、バージョンネゴシエーションを強制的に実行するために予約されている。つまり、すべてのバイトの下位 4 ビットが 1010 (二進数) である任意のバージョン番号は、これに従う。クライアントまたはサーバは、これらの予約されたバージョンに対するサポートをアドバタイズしてもかまわない [MAY]。

予約されたバージョン番号は実際のプロトコルを表すことはない。クライアントは、サーバがバージョンネゴシエーションを開始することを期待して、これらのバージョン番号のいずれかを使用してもかまわない [MAY]。サーバは、これらのバージョンのいずれかのサポートをアドバタイズしてもよく [MAY]、クライアントが値を無視することを期待できる。

１６．　可変長整数エンコーディング

QUIC パケットおよびフレームは、通常、負でない整数値に可変長エンコードを使用する。このエンコードにより、小さい整数値をエンコードするために必要なバイト数が少なくなる。

QUIC 可変長整数エンコードは、整数エンコード長の 2 を底とする対数をバイト単位でエンコードするために、最初のバイトの最上位 2 ビットを予約する。整数値は、残りのビットでネットワークバイトオーダーにエンコードされる。

つまり、整数は 1、2、4、または 8 バイトでエンコードされ、それぞれ 6、14、30、または 62 ビット値をエンコードできる。表 4 は、エンコードのプロパティをまとめたものである。

表 4　整数エンコーディングの要約

2MSB	長さ	使用可能ビット	範囲
00	1	6	0-63
01	2	14	0-16383
10	4	30	0-1073741823
11	8	62	0-4611686018427387903

デコードアルゴリズムの例とサンプルエンコーディングを付録 A.1 に示す。

Frame Type フィールドを唯一の例外として、値は必要最小限のバイト数でエンコードする必要はない。第 12.4 節を参照。

バージョン (第 15 章)、ヘッダで送信されるパケット番号(第 17.1 節)、およびロングヘッダパケット内のコネクション ID の長さ(第 17.2 節)は整数を使用して記述されるが、このエンコーディングは使用しない。

フィールドとパケットで検索

１７．　パケットフォーマット

すべての数値はネットワークバイトオーダー(つまり、ビッグエンディアン)でエンコードされ、すべてのフィールドサイズはビット単位である。フィールドの値の記述には 16 進表記が使用される。

17.1　パケット番号のエンコードとデコード

パケット番号は 0~2^{62}-1 の範囲の整数である(第 12.3 節参照)。ロングパケットヘッダまたはショートパケットヘッダに存在する場合は、1~4 バイトでエンコードされる。パケット番号を表すために必要なビット数は、パケット番号の最下位ビットのみを含めることによって削減される。

エンコードされたパケット番号は、[QUIC-TLS] セクション 5.4 で説明されているように保護される。

パケット番号空間の確認応答を受信する前に、完全なパケット番号が含まれていなければならない[MUST]。以下に説明するように、切り捨てられてはならない。

パケット番号空間の確認応答を受信した後、送信側は、最大の確認済みパケット番号と送信されるパケット番号の差の倍以上の範囲を表すことができるパケット番号サイズを使用しなければならない[MUST]。その後、パケットを受信したピアは、パケットが転送中に遅延して、より高い番号のパケットが多数受信された後に到着しない限り、パケット番号を正しくデコードする。エンドポイントは、その後に送信されたパケットの後にパケットが到着した場合でも、パケット番号を回復できるように、十分に大きなパケット番号エンコーディングを使用する必要がある[SHOULD]。

その結果、パケット番号エンコーディングのサイズは、新しいパケットを含む連続する未確認のパケット番号の 2 を底とする対数よりも少なくとも 1 ビット多くなる。パケット番号エンコーディングの擬似コードと例については、付録 A.2 参照。

受信側では、完全なパケット番号を回復する前に、パケット番号の保護が解除される。その後、完全なパケット番号は、存在する有効ビットの数、それらのビットの値、および正常に認証されたパケットで受信された最大のパケット番号に基づいて再構築される。パケット保護の解除を正常に完了するには、完全なパケット番号を回復する必要がある。

ヘッダ保護が解除されると、次に予想されるパケットに最も近いパケット番号値を見つけることによって、パケット番号がデコードされる。次に予想されるパケットは、受信した最大のパケット番号に 1 を加えたものである。擬似コードとパケット番号のデコードの例については、付録 A.3 参照。

17.2　ロングヘッダパケット

```
Long Header Packet {
  Header Form (1) = 1,
  Fixed Bit (1) = 1,
  Long Packet Type (2),
  Type-Specific Bits (4),
  Version (32),
  Destination Connection ID Length (8),
  Destination Connection ID (0..160),
  Source Connection ID Length (8),
  Source Connection ID (0..160),
  Type-Specific Payload (..),
}
```

図 13　ロングヘッダパケット形式

ロングヘッダは、1-RTT キーが確立される前に送信されるパケットに使用される。1-RTT キーが使用可能になると、送信側はショートヘッダ(第 17.3 節参照)を使用したパケットの送信に切り替える。ロング形式を使用すると、Version Negotiation パケットなどの特殊なパケットを、この統一された固定長のパケット形式で表すことができる。ロングヘッダを使用するパケットには、次のフィールドが含まれる。

Header Form: バイト 0 (最初のバイト) の最上位ビット (0x80) は、ロングヘッダの場合は 1 に設定される。

Fixed Bit: パケットが Version Negotiation パケットでない限り、バイト 0 の次のビット (0x40) は 1 に設定される。このビットに 0 の値を含むパケットは、このバージョンでは有効なパケットではないため、破棄する必要がある[MUST]。このビットの値が 1 の場合、QUIC は他のプロトコルと共存できる ([RFC7983] 参照）。

Long Packet Type: バイト 0 の次の 2 ビット (マスクが 0x30 のビット) には、パケットタイプが含まれる。パケットタイプを表 5 に示す。

Type-Specific Bits: バイト 0 の下位 4 ビット (マスクが 0x0f のビット) のセマンティクスは、パケットタイプによって決まる。

Version: QUIC バージョンは、最初のバイトに続く 32 ビットのフィールドである。このフィールドは、使用されている QUIC のバージョンを示し、残りのプロトコルフィールドがどのように解釈されるかを決定する。

Destination Connection ID Length: バージョンに続くバイトには、それに続く Destination Connection ID フィールドのバイト単位の長さが含まれる。この長さは、8 ビット符号なし整数としてエンコードされる。QUIC バージョン 1 では、この値は 20 バイトを超えてはならない[MUST NOT]。20 より大きい値を持つバージョン 1 のロングヘッダを受信するエンドポイントは、パケットを破棄しなければならない[MUST]。Version Negotiation パケットを適切に形成するために、サーバは他の QUIC バージョンからより長いコネクション ID を読み取ることができる必要がある[SHOULD]。

Destination Connection ID: Destination Connection ID フィールドは、このフィールドの長さを示す Destination Connection ID Length フィールドの後に続く。第 7.2 節は、このフィールドの使用をより詳細に説明する。

Source Connection ID Length: Destination Connection ID に続くバイトには、それに続く Source Connection ID フィールドのバイト単位の長さが含まれる。この長さは、8 ビットの符号なし整数としてエンコードされる。QUIC バージョン 1 では、この値は 20 バイトを超えてはならない[MUST NOT]。20 より大きい値を持つバージョン 1 のロングヘッダを受信するエンドポイントは、パケットを破棄しなければならない[MUST]。Version Negotiation パケットを適切に形成するために、サーバは他の QUIC バージョンからより長いコネクション ID を読み取ることができる必要がある[SHOULD]。

Source Connection ID: Source Connection ID フィールドは、このフィールドの長さを示す Source Connection ID Length フィールドの後に続く。第 7.2 節では、このフィールドの使用についてより詳細に説明する。

Type-Specific Payload: パケットの残りの部分 (存在する場合) は、タイプ固有である。

このバージョンの QUIC では、ロングヘッダを持つ次のパケットタイプが定義されている。

表 5 ロングヘッダパケットタイプ

タイプ	名前	セクション
0x00	Initial	第 17.2.2 項
0x01	0-RTT	第 17.2.3 項
0x02	Handshake	第 17.2.4 項
0x03	Retry	第 17.2.5 項

ロングヘッダパケットのヘッダフォームビット、Destination Connection ID Length フィールドと Source Connection ID Length フィールド、Destination Connection ID フィールドと Source Connection ID フィールド、および Version フィールドは、バージョンに依存しない。最初のバイトの他のフィールドはバージョン固有

である。異なるバージョンの QUIC からのパケットがどのように解釈されるかの詳細については、[QUIC-INVARIANTS] 参照。

フィールドとペイロードの解釈は、バージョンとパケットタイプに固有である。このバージョンのタイプ固有のセマンティクスについては、次の節で説明するが、このバージョンの QUIC のいくつかのロングヘッダパケットには、次の追加フィールドが含まれている。

Reserved Bits: バイト 0 の 2 ビット (マスクが 0x0c のもの) は、複数のパケットタイプにわたって予約されている。これらのビットは、ヘッダ保護を使用して保護される([QUIC-TLS] セクション 5.4 参照)。保護の前に含まれる値は、0 に設定しなければならない[MUST]。エンドポイントは、パケットとヘッダの両方の保護を削除した後に、これらのビットの値が 0 以外のパケットの受信を PROTOCOL_VIOLATION タイプのコネクションエラーとして処理しなければならない[MUST]。ヘッダ保護のみを削除した後にこのようなパケットを破棄すると、エンドポイントが攻撃にさらされる可能性がある([QUIC-TLS] セクション 9.5 参照)。

Packet Number Length: Packet Number フィールドを含むパケットタイプでは、バイト 0 の最下位の 2 ビット (マスクが 0x03 のもの) に Packet Number フィールドの長さが含まれる。これは、Packet Number フィールドの長さ (バイト単位) よりも 1 小さい符号なしの 2 ビット整数としてエンコードされる。つまり、Packet Number フィールドの長さは、このフィールドの値に 1 を加えた値である。これらのビットはヘッダ保護を使用して保護される([QUIC-TLS] セクション 5.4 参照)。

Length: パケットの残りの長さ(つまり、Packet Number フィールドと Payload フィールド)で、可変長整数としてエンコードされる (第 16 章参照) 。

Packet Number: このフィールドの長さは 1 から 4 バイトである。パケット番号はヘッダ保護を使用して保護される([QUIC-TLS] セクション 5.4 参照)。Packet Number フィールドの長さは、バイト 0 のパケット番号長ビットでエンコードされる(上記を参照)。

Packet Payload: これは、パケット保護を使用して保護されている、一連のフレームを含むパケットのペイロードである。

17.2.1 Version Negotiation パケット

Version Negotiation パケットは、本質的にバージョン固有ではない。クライアントが受信すると、値が 0 の Version フィールドに基づいて、Version Negotiation パケットとして識別される。

Version Negotiation パケットは、サーバでサポートされていないバージョンを含むクライアントパケットへの応答である。サーバによってのみ送信される。

Version Negotiation パケットのレイアウトは次のとおりである。

```
Version Negotiation Packet {
  Header Form (1) = 1,
  Unused (7),
  Version (32) = 0,
  Destination Connection ID Length (8),
  Destination Connection ID (0..2040),
  Source Connection ID Length (8),
  Source Connection ID (0..2040),
  Supported Version (32) ...,
}
```

図 14 Version Negotiation パケット

Unused フィールドの値は、サーバによって任意の値に設定される。クライアントは、このフィールドの値を無視しなければならない[MUST]。QUIC が他のプロトコルと多重化される可能性がある場合 ([RFC7983] 参照)、サーバは、Version Negotiation パケットが Fixed Bit フィールドを持つように見えるように、このフィー

ルドの最上位ビット (0x40) を 1 に設定する必要がある[SHOULD]。QUIC の他のバージョンが同様の推奨を行わない可能性があることに注意する。

Version Negotiation パケットの Version フィールドは、0x00000000 に設定しなければならない[MUST]。

サーバは、受信したパケットの Source Connection ID フィールドの値を Destination Connection ID フィールドに含めなければならない[MUST]。Source Connection ID フィールドの値は、最初にクライアントによってランダムに選択される、受信したパケットの Destination Connection ID フィールドの値からコピーしなければならない[MUST]。両方のコネクション ID をエコーすると、サーバがパケットを受信したこと、および Version Negotiation パケットが Initial パケットを監視していないエンティティによって生成されなかったことがクライアントに保証される。

QUIC の将来のバージョンでは、コネクション ID の長さに関する要件が異なる可能性がある。特に、コネクション ID は最小長が小さいか、最大長が大きいかもしれない。したがって、コネクション ID のバージョン固有の規則は、Version Negotiation パケットを送信するかどうかの決定に影響を与えてはならない[MUST NOT]。

Version Negotiation パケットの残りの部分は、サーバがサポートする 32 ビットバージョンのリストである。

Version Negotiation パケットは確認応答されない。サポートされていないバージョンを示すパケットへの応答としてのみ送信される(第 5.2.2 項参照)。

Version Negotiation パケットには、ロングヘッダ形式を使用する他のパケットに存在する Packet Number フィールドおよび Length フィールドは含まれない。その結果、Version Negotiation パケットは UDP データグラム全体を消費する。

サーバは、単一の UDP データグラムに応答して複数の Version Negotiation パケットを送信してはならない[MUST NOT]。

バージョンネゴシエーションプロセスの説明については、第 6 章を参照。

17.2.2 Initial パケット

Initial パケットは、タイプ値が 0x00 のロングヘッダを使用する。このパケットは、キー交換を実行するためにクライアントとサーバによって送信される最初の CRYPTO フレームを伝送し、いずれかの方向に ACK フレームを伝送する。

```
Initial Packet {
  Header Form (1) = 1,
  Fixed Bit (1) = 1,
  Long Packet Type (2) = 0,
  Reserved Bits (2),
  Packet Number Length (2),
  Version (32),
  Destination Connection ID Length (8),
  Destination Connection ID (0..160),
  Source Connection ID Length (8),
  Source Connection ID (0..160),
  Token Length (i),
  Token (..),
  Length (i),
  Packet Number (8..32),
  Packet Payload (8..),
}
```

図 15　初期パケット

Initial パケットはロングヘッダと Length と Packet Number フィールドを含む(第 17.2 節参照)。最初のバイトは Reserved ビットと Packet Number Length ビットを含んでいる(第 17.2 節参照)。Source Connection ID フィー

ルドと Source Connection ID Length フィールドの間に、Initial パケットに固有の二つの追加フィールドがある。

Token Length:　Token フィールドの長さをバイト単位で指定する可変長整数。トークンが存在しない場合、この値は 0 である。サーバによって送信された Initial パケットは、Token Length フィールドを 0 に設定しなければならない[MUST]。Token Length フィールドが 0 以外の Initial パケットを受信したクライアントは、パケットを破棄するか、PROTOCOL_VIOLATION タイプのコネクションエラーを生成しなければならない[MUST]。

Token:　Retry パケットまたは NEW_TOKEN フレームで以前に提供されたトークンの値(第 8.1 節参照)。

バージョンを認識しないミドルボックスによる改ざんを防ぐために、[QUIC-TLS] で説明されているように、Initial パケットはコネクションおよびバージョン固有のキー (Initial キー) で保護される。この保護は、パケットを監視できる攻撃者に対して機密性や完全性を提供しないが、パケットを監視できない攻撃者が Initial パケットを偽装することを防ぐ。

クライアントとサーバは、初期暗号化ハンドシェイクメッセージを含むすべてのパケットに Initial パケットタイプを使用する。これには、Retry パケットを受信した後に送信されるパケットなど、初期暗号化メッセージを含む新しいパケットを作成する必要があるすべてのケースが含まれる(第 17.2.5 項参照)。

サーバは、クライアントの Initial パケットに応答して最初の Initial パケットを送信する。サーバは、複数の Initial パケットを送信することができる[MAY]。暗号キー交換では、このデータの複数のラウンドトリップまたは再送信が必要になる場合がある。

Initial パケットのペイロードには、暗号化ハンドシェイクメッセージ、ACK フレーム、またはその両方を含む CRYPTO フレーム (またはフレーム) が含まれる。PING フレーム、PADDING フレーム、および 0x1c タイプの CONNECTION_CLOSE フレームも許可される。他のフレームを含む Initial パケットを受信するエンドポイントは、パケットを誤って破棄するか、コネクションエラーとして扱うことができる。

クライアントによって送信される最初のパケットには、常に、最初の暗号化ハンドシェイクメッセージの開始またはすべてを含む CRYPTO フレームが含まれる。送信される最初の CRYPTO フレームは、常にオフセット 0 から始まる(第 7 章参照)。

サーバが TLS HelloRetryRequest ([QUIC-TLS] セクション 4.7 参照) を送信する場合、クライアントは別の一連の Initial パケットを送信することに注意する。これらの Initial パケットは暗号化ハンドシェイクを継続し、Initial パケットの最初のフライトで送信された CRYPTO フレームのサイズと一致するオフセットから始まる CRYPTO フレームを含む。

17.2.2.1　Initial パケットの破棄

クライアントは、最初の Handshake パケットを送信すると、Initial パケットの送信と処理の両方を停止する。サーバは、最初の Handshake パケットを受信すると、Initial パケットの送信と処理を停止する。パケットはまだ送信中または確認応答待ちの場合があるが、この時点を超えて Initial パケットを交換する必要はない。Initial パケット保護キーは、損失回復および輻輳制御状態とともに破棄される ([QUIC-TLS] セクション 4.9.1、[QUIC-RECOVERY] セクション 6.4 参照)。

Initial キーが破棄されると、CRYPTO フレーム内のデータはすべて破棄され、再送信されなくなる。

17.2.3　0-RTT パケット

0-RTT パケットは、タイプ値が 0x01 のロングヘッダを使用し、その後に Length フィールドと Packet Number フィールドが続く(第 17.2 節を参照)。最初のバイトには予約ビットとパケット番号長ビットが含まれる(第 17.2 節を参照)。0-RTT パケットは、ハンドシェイクが完了する前に、最初のフライトの一部としてクライアントからサーバに"初期"データを運ぶために使用される。TLS ハンドシェイクの一部として、サーバはこの

初期データを受け入れたり拒否したりできる。

0-RTT データとその制限については、[TLS13] セクション 2.3 参照。

```
0-RTT Packet {
  Header Form (1) = 1,
  Fixed Bit (1) = 1,
  Long Packet Type (2) = 1,
  Reserved Bits (2),
  Packet Number Length (2),
  Version (32),
  Destination Connection ID Length (8),
  Destination Connection ID (0..160),
  Source Connection ID Length (8),
  Source Connection ID (0..160),
  Length (i),
  Packet Number (8..32),
  Packet Payload (8..),
}
```

図 16　0-RTT パケット

0-RTT で保護されたパケットのパケット番号は、1-RTT で保護されたパケットと同じ空間を使用する。

クライアントが Retry パケットを受信した後、0-RTT パケットはサーバによって失われたか破棄された可能性がある。クライアントは、新しい Initial パケットを送信した後、0-RTT パケットでデータの再送信を試みる必要がある[SHOULD]。新しいパケット番号は、送信される新しいパケットに使用されなければならない[MUST]。第 17.2.5.3 項で説明しているように、パケット番号を再利用するとパケット保護が損なわれる可能性がある。

クライアントは、[QUIC-TLS] セクション 4.1.1 で定義されているように、ハンドシェイクが完了した後にのみ 0-RTT パケットの確認応答を受信する。

クライアントは、サーバからの 1-RTT パケットの処理を開始したら、0-RTT パケットを送信してはならない[MUST NOT]。これは、0-RTT パケットに 1-RTT パケットからのフレームに対する応答を含めることができないことを意味する。たとえば、クライアントは 1-RTT パケットのみを確認できるため、0-RTT パケットで ACK フレームを送信することはできない。1-RTT パケットの確認は、1-RTT パケットで伝送する必要がある[MUST]。

サーバは、記憶された制限 (第 7.4.1 項参照) の違反を適切な種類のコネクションエラーとして扱う必要がある(たとえば、ストリームデータ制限を超えた場合の FLOW_CONTROL_ERROR)。

17.2.4　Handshake パケット

Handshake パケットは、タイプ値が 0x02 のロングヘッダを使用し、その後に Length フィールドと Packet Number フィールドが続く(第 17.2 節参照)。最初のバイトには、Reserved ビットと Packet Number Length ビットが含まれる(第 17.2 節参照)。これは、サーバとクライアントからの暗号化ハンドシェイクメッセージと確認応答を伝送するために使用される。

```
Handshake Packet {
  Header Form (1) = 1,
  Fixed Bit (1) = 1,
  Long Packet Type (2) = 2,
  Reserved Bits (2),
  Packet Number Length (2),
  Version (32),
  Destination Connection ID Length (8),
  Destination Connection ID (0..160),
  Source Connection ID Length (8),
  Source Connection ID (0..160),
  Length (i),
  Packet Number (8..32),
  Packet Payload (8..),
}
```

図 17　ハンドシェイクで保護されたパケット

クライアントは、サーバから Handshake パケットを受信すると、Handshake パケットを使用して、後続の暗号化ハンドシェイクメッセージと確認応答をサーバに送信する。

Handshake パケットの Destination Connection ID フィールドには、パケットの受信側によって選択されたコネクション ID が含まれる。Source Connection ID フィールドには、パケットの送信側が使用するコネクション ID が含まれる(第 7.2 節参照)。

Handshake パケットには独自のパケット番号空間があるため、サーバによって送信される最初の Handshake パケットにはパケット番号 0 が含まれる。

このパケットのペイロードには CRYPTO フレームが含まれ、PING フレーム、PADDING フレーム、または ACK フレームを含めることができる。Handshake パケットには、種類が 0x1c タイプの CONNECTION_CLOSE フレームを含めることができる[MAY]。エンドポイントは、他のフレームを含む Handshake パケットの受信を、種類が PROTOCOL_VIOLATION タイプのコネクションエラーとして扱う必要がある[MUST]。

Initial パケット (第 17.2.2.1 項参照) と同様に、Handshake 保護キーが破棄されると、Handshake パケットの CRYPTO フレーム内のデータは破棄され、再送信されなくなる。

17.2.5　Retry パケット

図 18 に示すように、Retry パケットは、タイプ値が 0x03 のロングパケットヘッダを使用する。サーバによって作成されたアドレス検証トークンを伝送する。再試行を実行するサーバによって使用される(第 8.1 節参照)。

```
Retry Packet {
  Header Form (1) = 1,
  Fixed Bit (1) = 1,
  Long Packet Type (2) = 3,
  Unused (4),
  Version (32),
  Destination Connection ID Length (8),
  Destination Connection ID (0..160),
  Source Connection ID Length (8),
  Source Connection ID (0..160),
  Retry Token (..),
  Retry Integrity Tag (128),
}
```

図 18　再試行パケット

Retry パケットには、保護されたフィールドは含まれない。Unused フィールドの値は、サーバによって任

意の値に設定される。クライアントはこれらのビットを無視しなければならない[MUST]。ロングヘッダのフィールドに加えて、次の追加フィールドが含まれる。

Retry Token:　　サーバがクライアントのアドレスを検証するために使用できる不透明なトークン。

Retry Integrity Tag:[QUIC-TLS] セクション 5.8 「再試行パケットの整合性」で定義されている。

17.2.5.1　再試行パケットの送信

サーバは、クライアントが Initial パケットの Source Connection ID フィールドに含めたコネクション ID を Destination Connection ID フィールドに入力する。

サーバは、選択したコネクション ID を Source Connection ID フィールドに含める。この値は、クライアントによって送信されたパケットの Destination Connection ID フィールドと同じであってはならない[MUST NOT]。クライアントは、Initial パケットの Destination Connection ID フィールドと同じ Source Connection ID フィールドを含む Retry パケットを破棄しなければならない[MUST]。クライアントは、送信する後続のパケットの Destination Connection ID フィールドで、Retry パケットの Source Connection ID フィールドの値を使用しなければならない[MUST]。

サーバは、Initial パケットおよび 0-RTT パケットに応答して Retry パケットを送信してもよい[MAY]。サーバは、受信した 0-RTT パケットを破棄またはバッファすることができる。サーバは、Initial パケットまたは 0-RTT パケットを受信するときに複数の Retry パケットを送信できる。サーバは、1 つの UDP データグラムに応答して複数の Retry パケットを送信してはならない[MUST NOT]。

17.2.5.2　再試行パケットの処理

クライアントは、コネクション試行ごとに最大で 1 つの Retry パケットを受け入れて処理しなければならない[MUST]。クライアントは、サーバから Initial パケットまたは Retry パケットを受信して処理した後、受信した後続の Retry パケットをすべて破棄しなければならない[MUST]。

クライアントは、検証できない Retry Integrity Tag を持つ Retry パケットを破棄しなければならない[MUST]([QUIC-TLS] セクション 5.8 参照)。これにより、攻撃者が Retry パケットを挿入する能力が低下し、Retry パケットの偶発的な破損から保護される。クライアントは、長さゼロの Retry Token フィールドを持つ Retry パケットを破棄しなければならない[MUST]。

クライアントは、コネクション確立を続行するために、提供された Retry Token を含む Initial パケットで Retry パケットに応答する。

クライアントは、この Initial パケットの Destination Connection ID フィールドを Retry パケットの Source Connection ID フィールドの値に設定する。Destination Connection ID フィールドを変更すると、Initial パケットを保護するために使用されるキーも変更される。また、Token フィールドを Retry パケットで提供されたトークンに設定する。サーバはトークン検証ロジックの一部としてコネクション ID を含めることができるため、クライアントは Source Connection ID を変更してはならない[MUST NOT](第 8.1.4 項参照)。

Retry パケットにはパケット番号が含まれておらず、クライアントが明示的に確認応答することはできない。

17.2.5.3　再試行後のハンドシェイクの継続

クライアントからの後続の Initial パケットには、Retry パケットのコネクション ID とトークン値が含まれる。クライアントは、Retry パケットの Source Connection ID フィールドを Destination Connection ID フィールドにコピーし、更新された値を持つ Initial パケットが受信されるまでこの値を使用する(第 7.2 節参照)。Token フィールドの値は、後続のすべての Initial パケットにコピーされる(第 8.1.2 項参照)。

Destination Connection ID フィールドおよび Token フィールドの更新を除き、クライアントによって送信さ

れる Initial パケットは、最初の Initial パケットと同じ制限を受ける。クライアントは、このパケットに含まれているのと同じ暗号化ハンドシェイクメッセージを使用する必要がある[MUST]。サーバは、異なる暗号化ハンドシェイクメッセージを含むパケットをコネクションエラーとして扱うか、破棄することができる[MAY]。Token フィールドを含めると、暗号化ハンドシェイクメッセージに使用できる領域が減少するため、クライアントが複数の Initial パケットを送信する必要が生じる可能性があることに注意する。

クライアントは、サーバによって提供されたコネクション ID に 0-RTT パケットを送信することによって、Retry パケットを受信した後に 0-RTT を試行することができる[MAY]。

クライアントは、Retry パケットを処理した後に、パケット番号空間のパケット番号をリセットしてはならない[MUST NOT]。特に、0-RTT パケットには、Retry パケットの受信時に再送信される可能性が高い機密情報が含まれている。これらの新しい 0-RTT パケットを保護するために使用されるキーは、Retry パケットに応答した結果として変更されない。ただし、これらのパケットで送信されるデータは、以前に送信されたものとは異なる可能性がある。同じパケット番号を持つこれらの新しいパケットを送信すると、異なるコンテンツを保護するために同じキーと nonce が使用される可能性があるため、これらのパケットのパケット保護が侵害される可能性がある。クライアントがパケット番号をリセットしたことを検出した場合、サーバはコネクションを中止してもよい[MAY]。

クライアントとサーバの間で交換される Initial パケットと Retry パケットで使用されるコネクション ID は、トランスポートパラメータにコピーされ、第 7.3 節で説明されているように検証される。

17.3 ショートヘッダパケット

このバージョンの QUIC は、ショートパケットヘッダを使用する単一のパケットタイプを定義する。

17.3.1 1-RTT パケット

1-RTT パケットは、ショートパケットヘッダを使用する。これは、バージョンと 1-RTT キーがネゴシエートされた後に使用される。

```
1-RTT Packet {
  Header Form (1) = 0,
  Fixed Bit (1) = 1,
  Spin Bit (1),
  Reserved Bits (2),
  Key Phase (1),
  Packet Number Length (2),
  Destination Connection ID (0..160),
  Packet Number (8..32),
  Packet Payload (8..),
}
```

図 19　1-RTT パケット

1-RTT パケットには、次のフィールドが含まれる。

Header Form:　バイト 0 の最上位ビット (0x80) は、ショートヘッダ用に 0 に設定される。

Fixed Bit:　バイト 0 の次のビット (0x40) は 1 に設定される。このビットに 0 の値を含むパケットは、このバージョンでは有効なパケットではないため、破棄しなければならない[MUST]。このビットの値が 1 の場合、QUIC は他のプロトコルと共存できる([RFC7983] 参照)。

Spin Bit:　バイト 0 の 3 番目の最上位ビット (0x20) はレイテンシスピンビットで、第 17.4 節で説明しているように設定する。

Reserved Bits:　バイト 0 の次の 2 ビット (マスクが 0x18 のビット) は予約されている。これらのビットはヘッダ保護を使用して保護されている([QUIC-TLS] セクション 5.4 参照)。保護の前に含まれる値は、0 に設定する必要がある[MUST]。エンドポイントは、パケットとヘッダの両方の保護を削除した後に、

これらのビットにゼロ以外の値を持つパケットの受信を PROTOCOL_VIOLATION タイプのコネクションエラーとして処理する必要がある[MUST]。ヘッダ保護のみを削除した後にこのようなパケットを破棄すると、エンドポイントが攻撃にさらされる可能性がある([QUIC-TLS] セクション 9.5 参照)。

Key Phase: バイト 0 の次のビット (0x04) はキーフェーズを示す。これにより、パケットの受信側は、パケットを保護するために使用されるパケット保護キーを識別できる。詳細については、[QUICTLS] を参照。このビットは、ヘッダ保護を使用して保護されている([QUIC-TLS] セクション 5.4 参照)。

Packet Number Length: バイト 0 の最下位の 2 ビット (マスクが 0x03 のビット) には、Packet Number フィールドの長さが含まれる。これは、Packet Number フィールドの長さ (バイト単位) よりも 1 小さい符号なしの 2 ビット整数としてエンコードされる。つまり、Packet Number フィールドの長さは、このフィールドの値に 1 を加えた値である。これらのビットは、ヘッダ保護を使用して保護される([QUIC-TLS] セクション 5.4 参照)。

Destination Connection ID: Destination Connection ID は、パケットの目的の受信側によって選択されるコネクション ID である。詳細については、第 5.1 節参照。

Packet Number: Packet Number フィールドの長さは 1 から 4 バイトである。パケット番号は、ヘッダ保護を使用して保護される([QUIC-TLS] セクション 5.4 参照)。Packet Number フィールドの長さは、Packet Number Length フィールドにエンコードされる。詳細については、第 17.1 節参照。

Packet Payload: 1-RTT パケットには、常に 1-RTT 保護ペイロードが含まれる。

ショートヘッダパケットのヘッダフォームビットと Destination Connection ID フィールドはバージョンに依存しない。残りのフィールドは、選択した QUIC バージョンに固有である。異なるバージョンの QUIC からのパケットがどのように解釈されるかの詳細については、[QUIC-INVARIANTS] を参照。

17.4 レイテンシスピンビット

レイテンシスピンビットは、1-RTT パケット用に定義されており (第 17.3.1 項を参照)、コネクション中のネットワークパス上の観測ポイントからの受動的なレイテンシモニタリングを可能にする。サーバは受信したスピン値を反映し、クライアントは 1 回の RTT 後にスピンする。経路上の観測者は、2 つのスピンビットトグルイベント間の時間を測定して、コネクションのエンドツーエンド RTT を推定できる。

ハンドシェイクを観測することでコネクションの初期 RTT を測定できるため、スピンビットは 1-RTT パケットにのみ存在する。したがって、スピンビットは、バージョンネゴシエーションとコネクション確立が完了した後に使用できる。オンパスの測定とレイテンシスピンビットの使用については、[QUIC-MANAGEABILITY] でさらに説明する。

スピンビットは、このバージョンの QUIC のオプション機能である[OPTIONAL]。この機能をサポートしていないエンドポイントは、以下に定義されているように、この機能を無効にしなければならない[MUST]。

各エンドポイントは、コネクションに対してスピンビットを有効にするか無効にするかを一方的に決定する。実装では、クライアントとサーバの管理者が、グローバルまたはコネクションごとにスピンビットを無効にすることを許可しなければならない[MUST]。管理者によってスピンビットが無効にされていない場合でも、エンドポイントは、スピンビットを無効にする QUIC コネクションがネットワーク上で一般的に観察されるようにするために、16 ネットワークパスごとに少なくとも 1 つ、または 16 コネクション ID ごとに 1 つのランダムな選択に対して、スピンビットの使用を無効にしなければならない[MUST]。各エンドポイントが個別にスピンビットを無効にすると、これにより、約 8 つのネットワークパスでスピンビット信号が無効になる。

スピンビットが無効になっている場合、エンドポイントはスピンビットを任意の値に設定してもよく[MAY]、受信値を無視しなければならない[MUST]。エンドポイントは、各パケットに対して個別に選択されたランダムな値、または各コネクション ID に対して個別に選択されたランダムな値にスピンビットを設定

することを推奨する[RECOMMENDED]。

コネクションに対してスピンビットが有効になっている場合、エンドポイントは各ネットワークパスのスピン値を保持し、1-RTT パケットがそのパスで送信されたときに、パケットヘッダのスピンビットを現在格納されている値に設定する。スピン値は、各ネットワークパスのエンドポイントで 0 に初期化される。各エンドポイントは、各パス上のピアから見た最大パケット番号も記憶する。

サーバは、特定のネットワークパス上のクライアントから見た最大パケット番号を増やす 1-RTT パケットを受信すると、そのパスのスピン値を、受信したパケットのスピンビットと同じに設定する。

クライアントは、特定のネットワークパス上のサーバからクライアントによって認識される最大パケット数を増加させる 1-RTT パケットを受信すると、そのパスのスピン値を受信したパケットのスピンビットの逆に設定する。

エンドポイントは、ネットワークパスで使用されているコネクション ID を変更するときに、ネットワークパスのスピン値を 0 にリセットする。

１８．　トランスポートパラメータのエンコード

[QUIC-TLS] で定義されている quic_transport_parameters 拡張の extension_data フィールドには、QUIC トランスポートパラメータが含まれている。これらは、図 20 に示すように、一連のトランスポートパラメータとしてエンコードされる。

```
Transport Parameters {
  Transport Parameter (..) ...,
}
```

図 20　トランスポートパラメータのシーケンス

各トランスポートパラメータは、図 21 に示すように、(識別子、長さ、値)タプルとしてエンコードされる。

```
Transport Parameter {
  Transport Parameter ID (i),
  Transport Parameter Length (i),
  Transport Parameter Value (..),
}
```

図 21　トランスポートパラメータのエンコード

Transport Parameter Length フィールドには、Transport Parameter Value フィールドの長さがバイト単位で格納される。

QUIC は、トランスポートパラメータをバイトシーケンスにエンコードし、暗号化ハンドシェイクに含める。

18.1　予約済みのトランスポートパラメータ

N の整数値に対して 31*N+27 という形式の識別子を持つトランスポートパラメータは、不明なトランスポートパラメータを無視するという要件を実行するために予約されている。これらのトランスポートパラメータにはセマンティクスがなく、任意の値を渡すことができる。

18.2　トランスポートパラメータの定義

この節では、このドキュメントで定義されているトランスポートパラメータについて詳しく説明する。

ここに列挙されている多くのトランスポートパラメータは整数値を持つ。整数として識別されるトランスポートパラメータは、可変長整数エンコーディングを使用する(第 16 章参照)。トランスポートパラメータが存在しない場合、特に指定がない限り、トランスポートパラメータのデフォルト値は 0 である。

以下のトランスポートパラメータが定義されている。

original_destination_connection_id (0x00) :　　このパラメータは、クライアントが最初に送信した Initial パケットの Destination Connection ID フィールドの値である(第 7.3 節参照)。このトランスポートパラメータはサーバによってのみ送信される。

max_idle_timeout (0x01) : 最大アイドルタイムアウトは、整数としてエンコードされたミリ秒単位の値である(第 10.1 節参照)。両方のエンドポイントがこのトランスポートパラメータを省略した場合、または値 0 を指定した場合、アイドルタイムアウトは無効になる。

stateless_reset_token (0x02) :　　ステートレスリセットトークンは、ステートレスリセットの検証に使用される(第 10.3 節参照)。このパラメータは 16 バイトのシーケンスである。このトランスポートパラメータはクライアントから送信してはならない[MUST NOT]、しかしサーバからは送信してもよい[MAY]。このトランスポートパラメータを送信しないサーバは、ハンドシェイク中にネゴシエートされたコネクション ID に対してステートレスリセット(第 10.3 節参照)を使用できない。

max_udp_payload_size (0x03) :　最大 UDP ペイロードサイズパラメータは、エンドポイントが受信できる UDP ペイロードのサイズを制限する整数値である。この制限より大きなペイロードを持つ UDP データグラムは、受信側によって処理されない可能性がある。

　このパラメータのデフォルトは、最大許容 UDP ペイロードの 65527 である。1200 未満の値は無効である。

　この制限は、パス MTU と同じようにデータグラムサイズに対する追加の制約として動作するが、これはエンドポイントのプロパティであり、パスのプロパティではない(第 14 章参照)。これは、エンドポイントが受信パケットを保持するために専用の領域であることが想定されている。

initial_max_data (0x04) :　initial maximum data パラメータは、コネクションで送信できるデータの最大量の初期値を含む整数値である。これは、ハンドシェイクの完了直後にコネクションの MAX_DATA フレーム(第 19.9 節参照)を送信することと同じである。

initial_max_stream_data_bidi_local (0x05) :　　このパラメータは、ローカルで開始された双方向ストリームの初期フロー制御制限を指定する整数値である。この制限は、トランスポートパラメータを送信するエンドポイントによって開かれた、新しく作成された双方向ストリームに適用される。クライアントトランスポートパラメータでは、これは、最下位の 2 ビットが 0x00 に設定された識別子を持つストリームに適用される。サーバトランスポートパラメータでは、これは、最下位の 2 ビットが 0x01 に設定されたストリームに適用される。

initial_max_stream_data_bidi_remote (0x06) :　このパラメータは、ピアによって開始される双方向ストリームの初期フロー制御制限を指定する整数値である。この制限は、トランスポートパラメータを受け取るエンドポイントによって開かれた、新しく作成された双方向ストリームに適用される。クライアントトランスポートパラメータでは、これは、最下位の 2 ビットが 0x01 に設定された識別子を持つストリームに適用される。サーバトランスポートパラメータでは、これは、最下位の 2 ビットが 0x00 に設定されたストリームに適用される。

initial_max_stream_data_uni (0x07) :　　このパラメータは、単方向ストリームの初期フロー制御制限を指定する整数値である。この制限は、トランスポートパラメータを受け取るエンドポイントによって開かれた、新しく作成された単方向ストリームに適用される。クライアントトランスポートパラメータでは、これは、最下位 2 ビットが 0x03 に設定された識別子を持つストリームに適用される。サーバトランスポートパラメータでは、これは、最下位 2 ビットが 0x02 に設定されたストリームに適用される。

initial_max_streams_bidi (0x08) : 初期最大双方向ストリームパラメータは、このトランスポートパラメータを受け取るエンドポイントが開始を許可される双方向ストリームの初期最大数を含む整数値である。このパラメータが存在しないか 0 の場合、MAX_STREAMS フレームが送信されるまで、ピアは双方向ストリームを開くことができない。このパラメータを設定することは、対応するタイプの

MAX_STREAMS フレーム (第 19.11 節参照) を同じ値で送信することと同じである。

initial_max_streams_uni (0x09) : 初期最大単一方向ストリームパラメータは、このトランスポートパラメータを受け取るエンドポイントが開始を許可される単一方向ストリームの初期最大数を含む整数値である。このパラメータが存在しないか 0 の場合、MAX_STREAMS フレームが送信されるまで、ピアは単一方向ストリームを開くことができない。このパラメータを設定することは、対応するタイプの MAX_STREAMS フレーム (第 19.11 節参照) を同じ値で送信することと同じである。

ack_delay_exponent (0x0a) : 受信確認遅延指数は、ACK フレームの ACK Delay フィールドをデコードするために使用される指数を示す整数値である(第 19.3 節参照)。この値が指定されていない場合は、デフォルト値の 3 が使用される (乗数が 8 であることを示す) 。20 を超える値は無効である。

max_ack_delay (0x0b) : 最大確認応答遅延は、エンドポイントが確認応答の送信を遅延させるミリ秒単位の最大時間を示す整数値である。この値には、アラームの発生における受信側の予想遅延を含める必要がある[SHOULD]。たとえば、受信側がタイマを 5 ミリ秒に設定し、アラームが通常最大 1 ミリ秒遅れて発生する場合は、6 ミリ秒の max_ack_delay を送信する必要がある。この値が指定されていない場合は、既定値の 25 ミリ秒が想定される。214 以上の値は無効である。

disable_active_migration (0x0c) : エンドポイントがハンドシェイク中に使用されているアドレスでアクティブなコネクション移行 (第 9 章参照) をサポートしていない場合は、disable active migration トランスポートパラメータが含まれる。このトランスポートパラメータを受け取るエンドポイントは、ハンドシェイク中にピアが使用したアドレスに送信するときに、新しいローカルアドレスを使用してはならない[MUST NOT]。このトランスポートパラメータは、クライアントが preferred_address トランスポートパラメータを操作した後のコネクション移行を禁止しない。このパラメータは長さゼロの値である。

preferred_address (0x0d) : サーバの優先アドレスは、第 9.6 節で説明しているように、ハンドシェイクの終了時にサーバアドレスを変更するために使用される。このトランスポートパラメータはサーバによってのみ送信される。サーバは、他のファミリに対してすべてゼロのアドレスとポート(0.0.0.0:0 または [::] :0)を送信することによって、一方のアドレスファミリの優先アドレスのみを送信することを選択してもよい[MAY]。IP アドレスはネットワークバイトオーダーでエンコードされる。

preferred_address トランスポートパラメータには、IPv4 と IPv6 の両方のアドレスとポートが含まれる。4 バイトの IPv4 Address フィールドの後に、関連する 2 バイトの IPv4 Port フィールドが続く。これには、16 バイトの IPv6 Address フィールドと 2 バイトの IPv6 Port フィールドが続く。アドレスとポートのペアの後、Connection ID Length フィールドには、次の Connection ID フィールドの長さが記述される。最後に、16 バイトの Stateless Reset Token フィールドには、コネクション ID に関連付けられたステートレスリセットトークンが含まれる。このトランスポートパラメータの形式を次の図 22 に示す。

Connection ID フィールドと Stateless Reset Token フィールドには、シーケンス番号 1 の代替コネクション ID が含まれる(第 5.1.1 項を参照)。これらの値を優先アドレスと共に送信すると、クライアントが優先アドレスへの移行を開始したときに、少なくとも 1 つの未使用のアクティブなコネクション ID があることが保証される。

優先アドレスの Connection ID フィールドと Stateless Reset Token フィールドは、構文とセマンティクスが NEW_CONNECTION_ID フレーム (第 19.15 節参照) の対応するフィールドと同じである。長さがゼロのコネクション ID を選択するサーバは、優先アドレスを提供してはならない[MUST NOT]。同様に、サーバは、このトランスポートパラメータに長さがゼロのコネクション ID を含めてはならない[MUST NOT]。クライアントは、これらの要件の違反を TRANSPORT_PARAMETER_ERROR タイプのコネクションエラーとして処理しなければならない[MUST]。

```
Preferred Address {
  IPv4 Address (32),
  IPv4 Port (16),
  IPv6 Address (128),
  IPv6 Port (16),
  Connection ID Length (8),
  Connection ID (..),
  Stateless Reset Token (128),
}
```

図 22　推奨されるアドレス形式

active_connection_id_limit (0x0e) :　これは、エンドポイントが格納できるピアからのコネクション ID の最大数を指定する整数値である。この値には、ハンドシェイク中に受信したコネクション ID、preferred_address トランスポートパラメータで受信したコネクション ID、および NEW_CONNECTION_ID フレームで受信したコネクション ID が含まれる。active_connection_id_limit パラメータの値は 2 以上である必要がある[MUST]。2 未満の値を受信したエンドポイントは、TRANSPORT_PARAMETER_ERROR タイプのエラーでコネクションを閉じなければならない[MUST]。このトランスポートパラメータが存在しない場合は、既定値の 2 が想定される。エンドポイントがゼロ長のコネクション ID を発行した場合、NEW_CONNECTION_ID フレームは送信されないため、ピアから受信した active_connection_id_limit 値は無視される。

initial_source_connection_id (0x0f) :　これは、エンドポイントがコネクションのために送信する最初の Initial パケットの Source Connection ID フィールドに含める値である(第 7.3 節参照)。

retry_source_connection_id (0x10) :これは、サーバが Retry パケットの Source Connection ID フィールドに含める値である(第 7.3 節参照)。このトランスポートパラメータは、サーバによってのみ送信される。

存在する場合、初期のストリームごとのフロー制御制限(initial_max_stream_data_bidi_local、initial_max_stream_data_bidi_remote、および initial_max_stream_data_uni)を設定するトランスポートパラメータは、オープン直後に対応する種類のすべてのストリームで MAX_STREAM_DATA フレーム (第 19.10 節参照) を送信することと同じである。トランスポートパラメータが指定されていない場合、そのタイプのストリームはフロー制御制限 0 で開始される。

クライアントには、サーバ専用のトランスポートパラメータ(original_destination_connection_id、preferred_address、retry_source_connection_id、または stateless_reset_token) を含めてはならない[MUST NOT]。サーバは、これらのトランスポートパラメータの受信を TRANSPORT_PARAMETER_ERROR タイプのコネクションエラーとして扱わなければならない[MUST]。

１９．　フレームのタイプとフォーマット

第 12.4 節で説明されているように、パケットには 1 つ以上のフレームが含まれる。本章では、コア QUIC フレームのタイプのフォーマットとセマンティクスについて説明する。

19.1　PADDING フレーム

PADDING フレーム (type=0x00) にはセマンティックな値は存在しない。PADDING フレームは、パケットのサイズを増加させるために使用できる。パディングは、Initial パケットを必要最小限のサイズに増加させる場合や、保護されたパケットに対するトラフィック分析からの保護を提供する場合に使用される。

PADDING フレームは図 23 に示されるように書式設定されており、PADDING フレームにはコンテンツが存在しないことを示している。つまり、PADDING フレームは、そのフレームを PADDING フレームとして識別する単一バイトで構成される。

```
PADDING Frame {
  Type (i) = 0x00,
}
```

図 23　PADDING フレームのフォーマット

19.2　PING フレーム

エンドポイントは、PING フレーム (type=0x01) を使用して、ピアがまだ生存していることを確認したり、ピアへの到達可能性を確認したりすることができる。

PING フレームは、図 24 に示されるように書式設定されており、PING フレームにはコンテンツが存在しないことを示している。

```
PING Frame {
  Type (i) = 0x01,
}
```

図 24　PING フレームのフォーマット

PING フレームの受信側は、このフレームを含むパケットを受信確認するだけでよい。

PING フレームは、アプリケーションまたはアプリケーションプロトコルがコネクションのタイムアウトを防ぎたい場合に、コネクションを維持するために使用できる (第 10.1.2 項)。

19.3　ACK フレーム

受信側は ACK フレーム (type 0x02 とタイプ 0x03) を送信して、受信して処理したパケットを送信側に通知する。ACK フレームには 1 つ以上の ACK Range が含まれる。ACK Range は、確認されたパケットを識別する。フレームタイプが 0x03 の場合、ACK フレームには、この時点までにコネクションで受信した関連する ECN マーク付きの QUIC パケットの累積数も含まれる。QUIC の実装は両方のタイプを適切に処理しなければならず[MUST]、送信するパケットに対して ECN を有効にしている場合は、ECN セクションの情報を使用して輻輳状態を管理するべきである [SHOULD]。

QUIC の確認応答は取り消し不可能である。いったん識別されたパケットは、将来の ACK フレームに表示されない場合でも、確認されたままとなる。これは、TCP 選択的確認応答 (SACK) [RFC2018] における取り消しとは異なる。

異なるパケット番号空間からのパケットは、同じ数値を使用して識別することができる。パケットの確認応答では、パケット番号とパケット番号空間の両方を示す必要がある。これを実現するために、各 ACK フレームは、その ACK フレームが含まれているパケットと同じ空間内のパケット番号のみを識別する。

Version Negotiation パケットと Retry パケットは、パケット番号を含まないため、識別することはできない。これらのパケットは、ACK フレームに依存するのではなく、クライアントによって送信される次の Initial パケットによって暗黙的に識別される。

ACK フレームは、図 25 に示されるように書式設定される。

```
ACK Frame {
  Type (i) = 0x02..0x03,
  Largest Acknowledged (i),
  ACK Delay (i),
  ACK Range Count (i),
  First ACK Range (i),
  ACK Range (..) ...,
  [ECN Counts (..)],
}
```

図 25　ACK フレームのフォーマット

ACK フレームには以下フィールドが含まれる：

Largest Acknowledged: ピアが確認している最大のパケット番号を表す可変長整数。この値は通常、ACK フレームを生成する前にピアが受信した最大のパケット番号である。QUIC ロングヘッダまたはショートヘッダのパケット番号とは異なり、ACK フレーム内の値は切り捨てられない。

ACK Delay: 確認応答遅延をマイクロ秒単位でエンコードする可変長整数 (第 13.2.5 項)。このフィールドの値は、ACK フレームの送信側が送信する ack_delay_exponent トランスポートパラメータの累乗を 2 倍してデコードされる (第 18.2 節)。単純に遅延を整数で表現する場合と比較して、このエンコーディングでは解像度が低くなるが、同じバイト数内でより広い範囲の値を表現できる。

ACK Range Count: フレーム内の ACK Range フィールドの数を指定する可変長整数。

First ACK Range: 確認されている Largest Acknowledged パケットの前にある連続パケットの数を示す可変長整数。つまり、範囲内で識別された最小のパケットは、Largest Acknowledged フィールドから最初の ACK Range 値を減算することによって決定される。

ACK Ranges: 識別されない (Gap) または識別される (ACK Range) パケットの追加範囲を含む (第 19.3.1 項)。

ECN Counts: 3 つの ECN カウント(第 19.3.2 項)。

19.3.1 ACK Ranges

各 ACK Range は、パケット番号の降順で、Gap 値と ACK Range Length 値が交互に並べられて構成される。ACK Range は繰り返し可能である。Gap 値と ACK Range Length 値の個数は、ACK Range Count フィールドによって決定される。ACK Range Count フィールドの値ごとに、それぞれの値が 1 つ存在する。

ACK Range の構造は、図 26 に示すとおりである。

```
ACK Range {
  Gap (i),
  ACK Range Length (i),
}
```

図 26　ACK Range のフォーマット

各 ACK Range を構成するフィールドは以下のとおりである。

Gap: 可変長整数で表され、前の ACK Range における最小のパケット番号よりも 1 つ小さいパケット番号の前に存在する、連続した識別されていないパケットの数を示す。

ACK Range Length: 可変長整数で表され、前のギャップによって決定される最大のパケット番号の前に存在する、連続した確認済みパケットの数を示す。

Gap 値と ACK Range Length 値は、効率のために相対整数エンコードを使用する。エンコードされた各値は正であるが、値が減算されるため、各 ACK Range は徐々に小さい番号のパケットを記述する。

各 ACK Range は、その範囲内の最大のパケット番号の前に存在する識別済みパケットの数を示すことによって、連続したパケットの範囲を確認する。値が 0 の場合、最大のパケット番号のみが確認されることを示す。ACK Range の値が大きいほど、範囲が広くなり、その範囲内の最小のパケット番号に対応する値は小さくなる。したがって、範囲の最大のパケット番号が与えられた場合、最小値は次の式によって決定される。

```
smallest = largest - ack_range
```

ACK Range は、最小のパケット番号から最大のパケット番号までのすべてのパケットを確認する。

ACK Range の最大値は、先行するすべての ACK Range Length と Gap のサイズを累積的に減算することによって決定される。

各 Gap は、受信確認されていないパケットの範囲を示す。Gap 内のパケットの数は、Gap フィールドのエンコードされた値よりも 1 つ多くなる。

Gap フィールドの値は、次の式を使用して、後続の ACK Range の最大パケット番号値を確立する。

```
largest = previous_smallest - gap - 2
```

計算されたパケット番号が負の場合、エンドポイントはFRAME_ENCODING_ERRORタイプのコネクションエラーを生成しなければならない [MUST]。

19.3.2 ECN Counts

ACKフレームは、タイプ値の最下位ビット(すなわち、type 0x03)を使用してECNフィードバックを示し、パケットのIPヘッダに関連付けられたECNコードポイントが、ECT (0)、ECT (1)、またはECN-CEであるQUICパケットの受信を報告する。ECNカウントは、ACKフレームのタイプが0x03の場合にのみ存在する。

ECNカウントが存在する場合、図27に示すように、3つのECNカウントがある。

```
ECN Counts {
  ECT0 Count (i),
  ECT1 Count (i),
  ECN-CE Count (i),
}
```

図 27　ECN Count のフォーマット

ECN countフィールドは次のとおりである。

ECT0 Count:　ACKフレームのパケット番号空間内でECT (0)コードポイントを持つパケットとして受信された合計数を表す可変長整数。

ECT1 Count:　ACKフレームのパケット番号空間内でECT (1)コードポイントを持つパケットとして受信された合計数を表す可変長整数。

ECN-CE Count:　ACKフレームのパケット番号空間内でECN-CEコードポイントを持つパケットとして受信された合計数を表す可変長整数。

ECNカウントは、パケット番号空間ごとに個別に保持される。

19.4 RESET_STREAM フレーム

エンドポイントは、RESET_STREAMフレーム (type=0x04) を使用して、ストリームの送信部分を唐突に終了する。

RESET_STREAMを送信した後、エンドポイントは、識別されたストリームにおけるSTREAMフレームの送信および再送信を停止する。RESET_STREAMフレームの受信側は、そのストリームで既に受信したデータを破棄することができる。

送信専用ストリームに対してRESET_STREAMフレームを受信したエンドポイントは、STREAM_STATE_ERRORタイプのエラーでコネクションを終了しなければならない [MUST]。

RESET_STREAMフレームのフォーマットは、図28に示されているとおりである。

```
RESET_STREAM Frame {
  Type (i) = 0x04,
  Stream ID (i),
  Application Protocol Error Code (i),
  Final Size (i),
}
```

図 28　RESET_STREAM フレームのフォーマット

RESET_STREAMフレームには、次のフィールドが含まれている:

Stream ID:　終了対象のストリームのストリームIDを示す可変長整数エンコード。

Application Protocol Error Code:　ストリームが閉じられる理由を示すアプリケーションプロトコルエラーコード(第20.2節)を含む可変長整数。

Final Size: RESET_STREAM フレームの送信側が示すストリームの最終サイズをバイト単位で示す可変長整数 (第 4.5 節)。

19.5 STOP_SENDING フレーム

エンドポイントは STOP_SENDING フレーム (type=0x05) を使用して、アプリケーションの要求に基づき受信データが破棄されていることを通知する。STOP_SENDING フレームは、ピアに対してストリームでの転送を停止するよう要求する。

STOP_SENDING フレームは、"Recv"または"Size Known"状態のストリームに対して送信することができる (第 3.2 節)。まだ作成されていないローカルで開始されたストリームに対する STOP_SENDING フレームの受信は、STREAM_STATE_ERROR タイプのコネクションエラーとして処理しなければならない [MUST]。受信専用ストリームに対する STOP_SENDING フレームを受信するエンドポイントは、STREAM_STATE_ERROR タイプのコネクションエラーを終了しなければならない [MUST]。

STOP_SENDING フレームのフォーマットは、図 29 に示されているとおりである。

```
STOP_SENDING Frame {
  Type (i) = 0x05,
  Stream ID (i),
  Application Protocol Error Code (i),
}
```

図 29　STOP_SENDING フレームのフォーマット

STOP_SENDING フレームには、次のフィールドが含まれる。

Stream ID: 無視されるストリームのストリーム ID を保持する可変長整数。

Application Protocol Error Code: 送信側がストリームを無視する理由を示す、アプリケーション指定の可変長整数(第 20.2 節)。

19.6 CRYPTO フレーム

CRYPTO フレーム (type=0x06) は、暗号ハンドシェイクメッセージの送信に使用される。これは、0-RTT を除くすべてのパケットタイプで送信可能である。CRYPTO フレームは、暗号プロトコルに対してバイトの in-order ストリームを提供する。CRYPTO フレームは、ストリーム識別子を持たない点を除き、機能的には STREAM フレームと同一である。また、フロー制御が適用されず、オプションのオフセット、オプションの長さ、およびストリームの末尾のマーカーを保持しない。

CRYPTO フレームのフォーマットは、図 30 に示されているとおりである。

```
CRYPTO Frame {
  Type (i) = 0x06,
  Offset (i),
  Length (i),
  Crypto Data (..),
}
```

図 30　CRYPTO フレームのフォーマット

CRYPTO フレームには、次のフィールドが含まれる。

Offset: この CRYPTO フレーム内のデータのストリーム内のバイトオフセットを指定する可変長整数。

Length: この CRYPTO フレーム内の暗号データフィールドの長さを指定する可変長整数。

Crypto Data: 暗号メッセージデータ。

暗号化レベルごとに暗号ハンドシェイクデータの個別のフローが存在し、それぞれがオフセット 0 から開

始する。これは、各暗号化レベルがデータの個別の CRYPTO ストリームとして扱われることを意味する。

ストリームで配信される最大オフセット (オフセットとデータ長の合計) は、2^{62}-1 を超えることはできない。この制限を超えるフレームを受信した場合、FRAME_ENCODING_ERROR タイプまたは CRYPTO_BUFFER_EXCEEDED タイプのコネクションエラーとして処理しなければならない [MUST]。

STREAM フレームとは異なり、STREAM フレームはデータが属するストリームを示すストリーム ID を含むが、CRYPTO フレームは暗号化レベルごとに単一のストリームのデータを伝送する。ストリームには明示的な終了がないため、CRYPTO フレームには FIN ビットが存在しない。

19.7 NEW_TOKEN フレーム

サーバは、NEW_TOKEN フレーム (type=0x07) を送信して、将来のコネクションにおいてクライアントが Initial パケットのヘッダで送信するためのトークンを提供する。

NEW_TOKEN フレームのフォーマットは、図 31 に示されているとおりである。

```
NEW_TOKEN Frame {
  Type (i) = 0x07,
  Token Length (i),
  Token (..),
}
```

図 31　NEW_TOKEN フレームのフォーマット

NEW_TOKEN フレームには、次のフィールドが含まれる。

Token Length:　トークンの長さをバイト単位で指定する可変長整数。

Token:　クライアントが将来の Initial パケットで使用できる不透明なデータ（BLOB）。トークンは空であってはならない [MUST NOT]。クライアントは、Token フィールドが空の NEW_TOKEN フレームを受信した場合、それを FRAME_ENCODING_ERROR タイプのコネクションエラーとして処理しなければならない[MUST]。

フレームを含むパケットが誤って送信失敗と判断された場合、クライアントは同じトークン値を含む複数の NEW_TOKEN フレームを受信する可能性がある。クライアントは、接続試行をリンクするために使用される可能性がある重複した値を破棄する責任がある(第 8.1.3 項)。

クライアントは NEW_TOKEN フレームを送信してはならない[MUST NOT]。サーバは、NEW_TOKEN フレームを受信した場合、それを PROTOCOL_VIOLATION タイプのコネクションエラーとして処理しなければならない[MUST]。

19.8 STREAM フレーム

STREAM フレームは暗黙的にストリームを作成し、ストリームデータを伝送する。STREAM フレームの Type フィールドは、0b00001XXX (または 0x08 から 0x0f までの値の集合) の形式を取る。フレームタイプの下位 3 ビットは、フレーム内に存在するフィールドを決定する:

- フレームタイプ内の OFF ビット (0x04) は、Offset フィールドが存在することを示すために設定される。これが 1 に設定されている場合、Offset フィールドが存在する。0 に設定されている場合、Offset フィールドは存在せず、Stream Data はオフセット 0 から開始される (つまり、このフレームはストリームの最初のバイトを含むか、またはデータを持たないストリームの末尾を含む。)。
- フレームタイプ内の LEN ビット (0x02) は、Length フィールドが存在することを示すために設定される。このビットが 0 に設定されている場合、Length フィールドは存在せず、Stream Data フィールドはパケットの末尾まで拡張される。このビットが 1 に設定されている場合、Length フィールドが存在する。

- FIN ビット (0x01) は、このフレームがストリームの末尾であることを示す。ストリームの最終サイズは、オフセットとこのフレームの長さの合計値である。

エンドポイントは、まだ作成されていないローカルで開始されたストリーム、または送信専用ストリームに対する STREAM フレームを受信した場合、STREAM_STATE_ERROR エラーでコネクションを終了しなければならない [MUST]。

ストリームフレームのフォーマットは、図 32 に示されているとおりである。

```
STREAM Frame {
  Type (i) = 0x08..0x0f,
  Stream ID (i),
  [Offset (i)],
  [Length (i)],
  Stream Data (..),
}
```

図 32　STREAM フレームのフォーマット

STREAM フレームには、次のフィールドが含まれる。

Stream ID:　ストリームのストリーム ID を示す可変長整数(第 2.1 節)。

Offset:　この STREAM フレーム内のデータに対応するストリームのバイトオフセットを指定する可変長整数。OFF ビットが 1 に設定されている場合にこのフィールドが存在する。Offset フィールドが存在しない場合、オフセットは 0 である。

Length:　この STREAM フレーム内の Stream Data フィールドの長さを指定する可変長整数。LEN ビットが 1 に設定されている場合にこのフィールドが存在する。LEN ビットが 0 に設定されている場合、Stream Data フィールドはパケット内の残りのすべてのバイトを消費する。

Stream Data: 指定されたストリームから配送されるバイト列。

Stream Data フィールドの長さが 0 の場合、STREAM フレーム内のオフセットは、次に送信されるバイトのオフセットとなる。

ストリーム内の最初のバイトのオフセットは 0 である。ストリームで配送される最大のオフセット (オフセットとデータ長の合計) は、2^{62}-1 を超えてはならない。これは、そのデータに対してフロー制御クレジットを提供できないためである。この制限を超えるフレームを受信した場合、FRAME_ENCODING_ERROR タイプまたは FLOW_CONTROL_ERROR タイプのコネクションエラーとして処理しなければならない [MUST]。

19.9　MAX_DATA フレーム

MAX_DATA フレーム (type=0x10) は、フロー制御において、コネクション全体で送信できるデータの最大量をピアに通知するために使用される。

MAX_DATA フレームのフォーマットは、図 33 に示されているとおりである。

```
MAX_DATA Frame {
  Type (i) = 0x10,
  Maximum Data (i),
}
```

図 33　MAX_DATA フレームのフォーマット

MAX_DATA フレームには、次のフィールドが含まれる。

Maximum Data:　コネクション全体で送信できるデータの最大量をバイト単位で示す可変長整数。

STREAM フレームで送信されるすべてのデータは、この制限にカウントされる。すべてのストリームの最終サイズの合計 (終端状態のストリームを含む) は、受信側が通知した値を超えてはならない[MUST NOT]。

エンドポイントは、送信した最大データ値よりも多くのデータを受信した場合、FLOW_CONTROL_ERROR タイプのエラーでコネクションを終了しなければならない[MUST]。これには、Early Data の記憶されている制限の違反も含まれる(第 7.4.1 項)。

19.10 MAX_STREAM_DATA フレーム

MAX_STREAM_DATA フレーム (type=0x11) は、フロー制御において、ストリームで送信できるデータの最大量をピアに通知するために使用される。

MAX_STREAM_DATA フレームは、"Recv"状態のストリームに対して送信できる(第 3.2 節)。まだ作成されていないローカルで開始したストリームに対して MAX_STREAM_DATA フレームを受信した場合は、STREAM_STATE_ERROR タイプのコネクションエラーとして扱わなければならない [MUST]。受信専用ストリームに対して MAX_STREAM_DATA フレームを受信したエンドポイントは、STREAM_STATE_ERROR タイプのコネクションエラーで終了しなければならない [MUST]。

MAX_STREAM_DATA フレームのフォーマットは、図 34 に示されているとおりである。

```
MAX_STREAM_DATA Frame {
  Type (i) = 0x11,
  Stream ID (i),
  Maximum Stream Data (i),
}
```

図 34　MAX_STREAM_DATA フレームのフォーマット

MAX_STREAM_DATA フレームには、次のフィールドが含まれる。

Stream ID: 影響を受けるストリームのストリーム ID を、可変長整数としてエンコードしたもの。

Maximum Stream Data:　識別されたストリームで送信できるデータの最大量をバイト単位で示す可変長整数。

この制限にデータをカウントする場合、エンドポイントは、ストリームで送信または受信されたデータの最大受信オフセットを考慮する。損失や並べ替えにより、ストリームの最大受信オフセットが、そのストリームで実際に受信したデータの合計サイズよりも大きくなる場合がある。STREAM フレームを受信しても、必ずしも最大受信オフセットが増加するわけではない。

ストリームで送信されるデータは、受信側が通知した最大ストリームデータ値のうち最大の値を超えてはならない[MUST NOT]。エンドポイントは、影響を受けるストリームに対して送信した最大ストリームデータよりも多くのデータを受信した場合、FLOW_CONTROL_ERROR タイプのエラーでコネクションを終了しなければならない [MUST]。これには、Early Data で記憶されている制限の違反も含まれる(第 7.4.1 項)。

19.11 MAX_STREAMS フレーム

MAX_STREAMS フレーム (type=0x12 または 0x13) は、ピアが開くことを許可されている特定のタイプのストリームの累積数を通知する。type が 0x12 の MAX_STREAMS フレームは双方向ストリームに適用され、type が 0x13 の MAX_STREAMS フレームは単方向ストリームに適用される。

MAX_STREAMS フレームのフォーマットは、図 35 に示されているとおりである。

```
MAX_STREAMS Frame {
  Type (i) = 0x12..0x13,
  Maximum Streams (i),
}
```

図 35　MAX_STREAMS フレームのフォーマット

MAX_STREAMS フレームには、次のフィールドが含まれる。

Maximum Streams:コネクションの有効期間中に開くことができる、対応する種類のストリームの累積数で

ある。この値は 2^{62}-1 より大きいストリーム ID をエンコードできないため、2^{60} を超えることはできない。この制限を超えるストリームを開くことを許可するフレームを受信した場合は、FRAME_ENCODING_ERROR タイプのコネクションエラーとして扱わなければならない [MUST]。

損失や並べ替えによって、エンドポイントは以前に受信したものよりも低いストリーム制限値の MAX_STREAMS フレームを受信する可能性がある。ストリーム制限を増やさない MAX_STREAMS フレームは無視しなければならない [MUST]。

エンドポイントは、ピアによって設定された現在のストリーム制限で許可されている数を超えてストリームを開いてはならない [MUST NOT]。例えば、単方向ストリームの制限値 3 を受信したサーバは、ストリーム 3、7、および 11 を開くことは許可されるが、ストリーム 15 を開くことは許可されない。ピアが許可された数よりも多くのストリームを開いた場合、エンドポイントは STREAM_LIMIT_ERROR タイプのエラーでコネクションを終了しなければならない[MUST]。これには、Early Data で記憶された制限違反も含まれる(第 7.4.1 項)。

これらのフレーム (および対応するトランスポートパラメータ) は、同時に開くことができるストリームの数を表しているわけではないことに注意すること。制限には、閉じられたストリームと開いているストリームの両方を含む。

19.12 DATA_BLOCKED フレーム

送信側は、データを送信したいがコネクションレベルのフロー制御のために送信できない場合、DATA_BLOCKED フレーム (type= 0x14) を送信するべきである[SHOULD](第 4 章)。DATA_BLOCKED フレームは、フロー制御アルゴリズムのチューニングのための入力として使用できる(第 4.2 節)。

DATA_BLOCKED フレームのフォーマットは、図 36 に示されているとおりである。

```
DATA_BLOCKED Frame {
  Type (i) = 0x14,
  Maximum Data (i),
}
```

図 36　DATA_BLOCKED フレームのフォーマット

DATA_BLOCKED フレームには、次のフィールドが含まれる。

Maximum Data:　ブロックが発生したコネクションレベルの制限を示す可変長整数。

19.13 STREAM_DATA_BLOCKED フレーム

送信側は、データを送信したいがストリームレベルのフロー制御により送信できない場合に、STREAM_DATA_BLOCKED フレーム (type=0x15) を送信するべきである[SHOULD]。このフレームは、DATA_BLOCKED(第 19.12 節) と類似している。

送信専用ストリームに対する STREAM_DATA_BLOCKED フレームを受信したエンドポイントは、STREAM_STATE_ERROR タイプのコネクションエラーで終了しなければならない[MUST]。

STREAM_DATA_BLOCKED フレームのフォーマットは、図 37 に示されているとおりである。

```
STREAM_DATA_BLOCKED Frame {
  Type (i) = 0x15,
  Stream ID (i),
  Maximum Stream Data (i),
}
```

図 37　STREAM_DATA_BLOCKED フレームのフォーマット

STREAM_DATA_BLOCKED フレームには、次のフィールドが含まれる。

Stream ID: フロー制御によりブロックされているストリームを示す可変長整数。

Maximum Stream Data: ブロックが発生したストリームのオフセットを示す可変長整数。

19.14 STREAMS_BLOCKED フレーム

送信側は、ストリームを開きたいがピアによって設定された最大ストリーム制限により開くことができない場合、STREAMS_BLOCKED フレーム (type=0x16 または 0x17) を送信するべきである[SHOULD](第 19.11 節)。0x16 タイプの STREAMS_BLOCKED フレームは双方向ストリーム制限に達したことを示すために使用され、0x17 タイプの STREAMS_BLOCKED フレームは単方向ストリーム制限に達したことを示すために使用される。

STREAMS_BLOCKED フレームはストリームを開くものではなく、新しいストリームが必要であったものの、ストリーム制限によりストリームの作成が妨げられたことをピアに通知する。

STREAMS_BLOCKED フレームのフォーマットは、図 38 に示されているとおりである。

```
STREAMS_BLOCKED Frame {
  Type (i) = 0x16..0x17,
  Maximum Streams (i),
}
```

図 38 STREAMS_BLOCKED フレームのフォーマット

STREAMS_BLOCKED フレームには、次のフィールドが含まれる。

Maximum Streams: フレーム送信時点で許可されているストリームの最大数を示す可変長整数。この値は 2^{60} を超えることはできない。なぜなら、2^{62}-1 を超えるストリーム ID をエンコードすることはできないためである。より大きなストリーム ID をエンコードしたフレームを受信した場合、それは STREAM_LIMIT_ERROR タイプまたは FRAME_ENCODING_ERROR タイプのコネクションエラーとして扱われなければならない[MUST]。

19.15 NEW_CONNECTION_ID フレーム

エンドポイントは NEW_CONNECTION_ID フレーム (type=0x18) を送信して、コネクションの移行時にリンカビリティを解除するために使用できる代替コネクション ID をピアに提供する(第 9.5 節)。

NEW_CONNECTION_ID フレームのフォーマットは、図 39 に示されているとおりである。

```
NEW_CONNECTION_ID Frame {
  Type (i) = 0x18,
  Sequence Number (i),
  Retire Prior To (i),
  Length (8),
  Connection ID (8..160),
  Stateless Reset Token (128),
}
```

図 39 NEW_CONNECTION_ID フレームのフォーマット

NEW_CONNECTION_ID フレームには、次のフィールドが含まれる。

Sequence Number: 送信側によってコネクション ID に割り当てられたシーケンス番号。可変長整数としてエンコードされる(第 5.1.1 項)。

Retire Prior To: 無効化すべきコネクション ID を示す可変長整数(第 5.1.2 項)。

Length: コネクション ID の長さを含む 8 ビット符号なし整数。1 より小さく 20 より大きい値は無効であり、FRAME_ENCODING_ERROR タイプのコネクションエラーとして扱われなければならない[MUST]。

Connection ID : 指定された長さのコネクション ID。

Stateless Reset Token:　関連付けられたコネクション ID が使用されるときにステートレスリセットで使用される 128 ビット値 (第 10.3 節)。

エンドポイントは、現在ピアが長さゼロの Destination Connection ID を持つパケットを送信することを要求している場合、このフレームを送信してはならない[MUST NOT]。コネクション ID の長さをゼロに変更したり、ゼロの長さから変更したりすると、コネクション ID の値がいつ変更されたのかを識別することが困難になる。長さゼロの Destination Connection ID を持つパケットを送信しているエンドポイントは、NEW_CONNECTION_ID フレームの受信を PROTOCOL_VIOLATION タイプのコネクションエラーとして処理しなければならない[MUST]。

送信エラー、タイムアウト、および再送信によって、同じ NEW_CONNECTION_ID フレームが複数回受信される可能性がある。同じフレームを複数回受信しても、コネクションエラーとして処理してはならない[MUST NOT]。受信側は、NEW_CONNECTION_ID フレームで提供されたシーケンス番号を用いることで、同じ NEW_CONNECTION_ID フレームを複数回受信した場合の処理を行うことができる。

エンドポイントが、以前に発行されたコネクション ID と異なる Stateless Reset Token フィールド値または異なる Sequence Number フィールド値で繰り返す NEW_CONNECTION_ID フレームを受信した場合、あるいは異なるコネクション ID に同じシーケンス番号が使用された場合、エンドポイントはその受信を PROTOCOL_VIOLATION タイプのコネクションエラーとして処理することができる[MAY]。

Retire Prior To フィールドは、コネクションのセットアップ中に確立されたコネクション ID と preferred_address トランスポートパラメータに適用される(第 5.1.2 項)。Retire Prior To フィールドの値は、シーケンス番号フィールドの値以下でなければならない[MUST]。Retire Prior To フィールドの値が Sequence Number フィールドの値を超えている場合は、それは FRAME_ENCODING_ERROR タイプのコネクションエラーとして扱わなければならない[MUST]。

送信側が Retire Prior To 値を示した後、それより小さい値が後続の NEW_CONNECTION_ID フレームで送信された場合、それらの値は効果を持たなくなる。受信側は、受信した Retire Prior To 値の最大値を増加させない Retire Prior To フィールドを無視しなければならない[MUST]。

エンドポイントが、以前に受信した NEW_CONNECTION_ID フレームの Retire Prior To フィールドよりも小さいシーケンス番号を持つ NEW_CONNECTION_ID フレームを受信した場合、そのシーケンス番号に対して対応する RETIRE_CONNECTION_ID フレームを送信していない限り、新たに受信したコネクション ID を無効化する対応をする RETIRE_CONNECTION_ID フレームを送信しなければならない[MUST]。

19.16　RETIRE_CONNECTION_ID フレーム

エンドポイントは、ピアによって発行されたコネクション ID を今後使用しないことを示すために、RETIRE_CONNECTION_ID フレーム (type=0x19) を送信する。これには、ハンドシェイク中に提供されたコネクション ID が含まれる。RETIRE_CONNECTION_ID フレームの送信は、将来の利用のために追加のコネクション ID を送信するように相手に要求する役割も果たす(第 5.1 節)。新たなコネクション ID は、NEW_CONNECTION_ID フレーム(第 19.15 節) を使用してピアに配信できる。

コネクション ID を破棄すると、そのコネクション ID に関連付けられているステートレスリセットトークンが無効になる。

RETIRE_CONNECTION_ID フレームのフォーマットは、図 40 に示されているとおりである。

```
RETIRE_CONNECTION_ID Frame {
  Type (i) = 0x19,
  Sequence Number (i),
}
```

図 40　RETIRE_CONNECTION_ID フレームのフォーマット

RETIRE_CONNECTION_ID フレームには、次のフィールドが含まれる。

Sequence Number:　　されるコネクション ID のシーケンス番号 (第 5.1.2 項)。

ピアに以前送信されたシーケンス番号よりも大きいシーケンス番号を含む RETIRE_CONNECTION_ID フレームを受信した場合、これを PROTOCOL_VIOLATION タイプのコネクションエラーとして扱わなければならない [MUST]。

RETIRE_CONNECTION_ID フレームで指定するシーケンス番号は、そのフレームを含むパケットの Destination Connection ID フィールドを参照してはならない [MUST NOT]。ピアは、これを PROTOCOL_VIOLATION タイプのコネクションエラーとして扱ってもよい [MAY]。

ピアから長さゼロのコネクション ID が提供された場合、エンドポイントはこのフレームを送信することはできない。長さゼロのコネクション ID を提供するエンドポイントは、RETIRE_CONNECTION_ID フレームの受信を PROTOCOL_VIOLATION タイプのコネクションエラーとして扱わなければならない [MUST]。

19.17　PATH_CHALLENGE フレーム

エンドポイントは、PATH_CHALLENGE フレーム (type=0x1a) を使用して、ピアへの到達性を確認したり、コネクションの移行中にパスを検証したりできる。

PATH_CHALLENGE フレームのフォーマットは、図 41 に示されているとおりである。

```
PATH_CHALLENGE Frame {
  Type (i) = 0x1a,
  Data (64),
}
```

図 41　PATH_CHALLENGE フレームのフォーマット

PATH_CHALLENGE フレームには、次のフィールドが含まれる。

Data:　　この 8 バイトのフィールドには、任意のデータが含まれる。

PATH_CHALLENGE フレームに 64 ビットのエントロピを含めることで、値を正しく推測するよりもパケットを受信するほうが容易になる。

このフレームの受信側は、同じ Data 値を含む PATH_RESPONSE フレーム(第 19.18 節) を生成しなければならない [MUST]。

19.18　PATH_RESPONSE フレーム

PATH_RESPONSE フレーム (type=0x1b) は、PATH_CHALLENGE フレームへの応答として送信される。

PATH_RESPONSE フレームのフォーマットは、図 42 に示されているとおりである。PATH_RESPONSE フレームのフォーマットは、PATH_CHALLENGE フレームのフォーマットと同じである(第 19.17 節)。

```
PATH_RESPONSE Frame {
  Type (i) = 0x1b,
  Data (64),
}
```

図 42　PATH_RESPONSE フレームのフォーマット

PATH_RESPONSE フレームの内容が、エンドポイントが以前に送信された PATH_CHALLENGE フレームの内容と一致しない場合、エンドポイントは PROTOCOL_VIOLATION タイプのコネクションエラーを生成してもよい [MAY]。

19.19　CONNECTION_CLOSE フレーム

エンドポイントは、CONNECTION_CLOSE フレーム (type=0x1c または 0x1d) を送信して、コネクションが

閉じられることをピアに通知する。タイプ 0x1c の CONNECTION_CLOSE フレームは、QUIC レイヤのみで発生したエラー、またはエラーが存在しない場合 (NO_ERROR コードを使用) を示すために使用される。タイプ 0x1d の CONNECTION_CLOSE フレームは、QUIC を使用するアプリケーションで発生したエラーを示すために使用される。

明示的に閉じられていない開いているストリームが存在する場合は、それらはコネクションが閉じられるときに暗黙的に閉じられる。

CONNECTION_CLOSE フレームのフォーマットは、図 43 に示されているとおりである。

```
CONNECTION_CLOSE Frame {
  Type (i) = 0x1c..0x1d,
  Error Code (i),
  [Frame Type (i)],
  Reason Phrase Length (i),
  Reason Phrase (..),
}
```

図 43　CONNECTION_CLOSE フレームのフォーマット

CONNECTION_CLOSE フレームには、次のフィールドが含まれる。

Error Code:　このコネクションを閉じる理由を示す可変長整数。タイプ 0x1c の CONNECTION_CLOSE フレームは、第 20.1 節で定義される領域のコードを使用する。タイプ 0x1d の CONNECTION_CLOSE フレームは、アプリケーションプロトコルで定義されたコードを使用する(第 20.2 節)。

Frame Type:　エラーを引き起こしたフレームのタイプをエンコードする可変長整数。フレームタイプが不明な場合、値 0 (PADDING フレームの記述と同じ) が使用される。CONNECTION_CLOSE フレーム(タイプ 0x1d) のアプリケーション固有のバリアントには、このフィールドは含まれない。

Reason Phrase Length: 理由フレーズの長さをバイト単位で指定する可変長整数。CONNECTION_CLOSE フレームはパケット間で分割できないため、パケットサイズの制限によって、理由フレーズに使用できる領域も制限される。

Reason Phrase:　クロージャに関する追加の診断情報。送信側がエラーコード値以上の詳細を提供しないことを選択した場合、この長さは 0 にすることができる。このフィールドは UTF-8 でエンコードされた文字列 [RFC3629] とすべき[SHOULD]であるが、フレームはそのテキストを作成したエンティティ以外による理解を支援する言語タグなどの情報は持たない。

CONNECTION_CLOSE フレームのアプリケーション固有のバリアント (タイプ 0x1d) は、0-RTT または 1-RTT パケットを使用してのみ送信できる(第 12.5 節)。アプリケーションがハンドシェイク中にコネクションを破棄したい場合、エンドポイントは、Initial パケットまたは Handshake パケットで APPLICATION_ERROR のエラーコードを含む CONNECTION_CLOSE フレーム (タイプ 0x1c) を送信できる。

19.20　HANDSHAKE_DONE フレーム

サーバは HANDSHAKE_DONE フレーム (type=0x1e) を使用して、ハンドシェイクの確認をクライアントに通知する。

HANDSHAKE_DONE フレームのフォーマットは、図 44 に示されているとおりである。図 44 は、HANDSHAKE_DONE フレームにコンテンツがないことを示している。

```
HANDSHAKE_DONE Frame {
  Type (i) = 0x1e,
}
```

図 44　HANDSHAKE_DONE フレームの書式

HANDSHAKE_DONE フレームはサーバのみが送信できる。サーバは、ハンドシェイクを完了する前に HANDSHAKE_DONE フレームを送信してはならない [MUST NOT]。サーバは、HANDSHAKE_DONE フレームの受信を PROTOCOL_VIOLATION タイプのコネクションエラーとして扱わなければならない [MUST]。

19.21 Extension(拡張)フレーム

QUIC フレームは自己記述エンコーディングを使用しない。したがって、エンドポイントはパケットを正常に処理する前に、すべてのフレームの構文を理解している必要がある。この方式によりフレームの効率的なエンコーディングを可能にするが、エンドポイントは自分のピアが未対応であるフレームタイプのフレームを送信することができない。

新しい種類のフレームを使用したい QUIC の拡張は、まずピアがそのフレームを理解できることを確実にしなければならない[MUST]。エンドポイントは、トランスポートパラメータを利用して、Extension フレームタイプの受信に同意する旨を通知できる。1 つのトランスポートパラメータは、1 つ以上の Extension フレームタイプの種類のサポートを示すことができる。

コアプロトコル機能 (フレームタイプを含む) を変更または置換する拡張は、組み合わせの動作が明確に定義されていない限り、同じ機能を変更または置換する他の拡張と組み合わせることは困難である。このような拡張は、同じプロトコルコンポーネントを変更する既存の拡張との相互作用を定義するべきである [SHOULD]。

Extension フレームは輻輳制御される必要があり [MUST]、ACK フレームの送信を引き起こさなければならない[MUST]。例外は、ACK フレームを置換または補完する Extension フレームである。Extension フレームは、拡張でその旨が指定されていない限り、フロー制御には含まれない。

IANA レジストリはフレームタイプの割り当て管理に使用される(第 22.4 節)。

２０． エラーコード

QUIC トランスポートエラーコードおよびアプリケーションエラーコードは、62 ビット符号なし整数である。

20.1 トランスポートエラーコード

この節では、0x1c タイプの CONNECTION_CLOSE フレームで使用できる定義済みの QUIC トランスポートエラーコードを示す。これらのエラーは、コネクション全体に適用される。

NO_ERROR (0x00) : エンドポイントは、これを CONNECTION_CLOSE フレームと共に使用して、エラーがない場合にコネクションが突然閉じられることを通知。

INTERNAL_ERROR (0x01) : エンドポイントで内部エラーが発生したため、コネクションを続行不可。

CONNECTION_REFUSED (0x02) : サーバが新しいコネクションの受け入れを拒否。

FLOW_CONTROL_ERROR (0x03) : エンドポイントが、アドバタイズされたデータ制限で許可されているよりも多くのデータを受信(第 4 章を参照)。

STREAM_LIMIT_ERROR (0x04) : エンドポイントが受信したストリーム識別子のフレームが、対応するストリームタイプのアドバタイズされたストリーム制限を超過。

STREAM_STATE_ERROR (0x05) : エンドポイントが受信したストリームのフレームが、そのフレームを許可する状態ではない(第 3 章を参照)。

FINAL_SIZE_ERROR (0x06) : (1) エンドポイントが、以前に確立された最終サイズを超えるデータを含む STREAM フレームを受信。 (2) エンドポイントが、既に受信したストリームデータのサイズよりも小さい最終サイズを含む STREAM フレームまたは RESET_STREAM フレームを受信。(3) エンドポイントは、既に確立されているものとは異なる最終サイズを含む STREAM フレームまたは RESET_STREAM フレームを受信。

FRAME_ENCODING_ERROR (0x07) : エンドポイントが、正しく書式設定されていないフレームを受信。たとえば、不明な種類のフレームや、パケットの残りの部分よりも多くの確認範囲を持つ ACK フレームなど。

TRANSPORT_PARAMETER_ERROR (0x08) : エンドポイントが、正しく書式設定されていない、無効な値が含まれている、必須のトランスポートパラメータが省略されている、禁止されているトランスポートパラメータが含まれている、またはその他のエラーが発生しているトランスポートパラメータを受信。

CONNECTION_ID_LIMIT_ERROR (0x09) : ピアによって提供されたコネクション ID の数が、アドバタイズされた active_connection_id_limit を超過。

PROTOCOL_VIOLATION (0x0a) : エンドポイントが、より具体的なエラーコードでカバーされていないプロトコル準拠のエラーを検出。

INVALID_TOKEN (0x0b) : サーバが、無効な Token フィールドを含むクライアントの Initial パケットを受信。

APPLICATION_ERROR (0x0c) : アプリケーションまたはアプリケーションプロトコルが原因でコネクションが終了。

CRYPTO_BUFFER_EXCEEDED (0x0d) : エンドポイントが、バッファできる量を超えるデータを CRYPTO フレームで受信。

KEY_UPDATE_ERROR (0x0e) : エンドポイントが、キーの更新の実行中にエラーを検出 [QUIC-TLS] (第 6 章を参照)。

AEAD_LIMIT_REACHED (0x0f) : エンドポイントが、指定されたコネクションで使用される AEAD アルゴリズムの機密性または整合性の制限に到達。

NO_VIABLE_PATH (0x10) : エンドポイントが、ネットワークパスが QUIC をサポートできないと判断。エンドポイントは、パスが十分な大きさの MTU をサポートしていない場合を除き、このコードを含む CONNECTION_CLOSE フレームを受信する可能性がある。

CRYPTO_ERROR (0x0100-0x01ff) : 暗号ハンドシェイクが失敗。256 の値の範囲は、使用される暗号ハンドシェイクに固有のエラーコードを伝達するために予約されている。TLS が暗号ハンドシェイクに使用されるときに発生するエラーのコードについては、[QUIC-TLS] の第 4.8 節で説明されている。

新しいエラーコードの登録の詳細については、第 22.5 節を参照。

これらのエラーコードを定義する際には、いくつかの原則が適用される。受信側で特定のアクションを必要とする可能性のあるエラー条件には、一意のコードが与えられる。一般的な条件を表すエラーには、特定のコードが与えられる。これらの条件のいずれかがない場合、エラーコードは、フロー制御やトランスポートパラメータの処理など、スタックの一般的な機能を識別するために使用される。最後に、実装がより具体的なコードを使用できない、または使用したくない状況に対して、一般的なエラーが提供される。

20.2 アプリケーションプロトコルのエラーコード

アプリケーションエラーコードの管理は、アプリケーションプロトコルに任されている。アプリケーションプロトコルのエラーコードは、RESET_STREAM フレーム(第 19.4 節)、STOP_SENDING フレーム(第 19.5 節)、および 0x1d タイプの CONNECTION_CLOSE フレームに使用される (第 19.19 節) 。

２１． セキュリティに関する考慮事項

QUIC の目標は、セキュリティで保護されたトランスポートコネクションを提供することである。第 21.1 節では、これらのプロパティの概要を示す。以降の節では、既知の攻撃と対策の説明を含め、これらのプロパティに関する制約と警告について説明する。

21.1　セキュリティプロパティの概要

QUIC の完全なセキュリティ分析は、このドキュメントの範囲外である。この節では、実装者を支援し、プロトコル分析のガイドとして役立つように、必要なセキュリティプロパティの非形式的な説明を提供する。

QUIC は、[SEC-CONS] で説明されている脅威モデルを想定し、そのモデルから発生する多くの攻撃に対する保護を提供する。

この目的のために、攻撃はパッシブ攻撃とアクティブ攻撃に分けられる。パッシブ攻撃者はネットワークからパケットを読み取ることができるが、アクティブ攻撃者はネットワークにパケットを書き込むこともできる。ただし、パッシブ攻撃には、コネクションを構成するパケットが通過するパスにルーティング変更またはその他の変更を引き起こす能力を持つ攻撃者が関与する可能性がある。

攻撃者はさらに、オンパス攻撃者またはオフパス攻撃者のいずれかに分類される。オンパス攻撃者は、パケットが宛先に到達しないように監視するパケットを読み取り、変更、または削除できる。一方、オフパス攻撃者はパケットを監視するが、元のパケットが意図した宛先に到達するのを防ぐことはできない。両方の種類の攻撃者は、任意のパケットを送信することもできる。この定義は、オフパス攻撃者がパケットを監視できるという点で、[SEC-CONS] の第 3.5 節とは異なる。

ハンドシェイク、保護されたパケット、およびコネクション移行のプロパティは、個別に考慮される。

21.1.1　ハンドシェイク

QUIC ハンドシェイクには TLS 1.3 ハンドシェイクが組み込まれており、[TLS13] の付録 E.1 に記載されている暗号化プロパティを継承している。QUIC のセキュリティプロパティの多くは、これらのプロパティを提供する TLS ハンドシェイクに依存している。TLS ハンドシェイクに対する攻撃は、QUIC に影響を与える可能性がある。

セッションキーの秘密性や一意性、または参加しているピアの認証を侵害する TLS ハンドシェイクに対する攻撃は、それらのキーに依存する QUIC によって提供される他のセキュリティ保証に影響する。たとえば、移行 (第 9 章) は、ネットワークパス間のリンク性を回避するために、TLS ハンドシェイクを使用したキーのネゴシエーションと QUIC パケット保護の両方について、機密保護の有効性に依存する。

TLS ハンドシェイクの整合性に対する攻撃により、攻撃者はアプリケーションプロトコルまたは QUIC バージョンの選択に影響を与える可能性がある。

TLS によって提供されるプロパティに加えて、QUIC ハンドシェイクはハンドシェイクに対する DoS 攻撃に対する防御を提供する。

21.1.1.1　増幅防止

アドレス検証 (第 8 章) は、特定のアドレスを要求するエンティティがそのアドレスでパケットを受信できることを検証するために使用される。アドレス検証は、攻撃者がパケットを監視できるアドレスに増幅攻撃のターゲットを制限する。

アドレス検証の前に、エンドポイントは送信できる内容が制限される。エンドポイントは、未検証のアドレスに対して、そのアドレスから受信したデータの 3 倍を超えるデータを送信することはできない。

> 注意:増幅防止の制限は、エンドポイントが未検証のアドレスから受信したパケットに応答する場合にのみ適用される。増幅防止の制限は、新しいコネクションを確立するとき、またはコネクションの移行を開始するときに、クライアントには適用されない。

21.1.1.2　サーバ側 DoS

フルハンドシェイクのためのサーバの最初のフライトの計算は、署名とキー交換の両方の計算を必要とするため、コストがかかる可能性がある。計算による DoS 攻撃を防ぐために、再試行パケットは安価なトーク

ン交換メカニズムを提供する。これにより、サーバは、1 回のラウンドトリップのコストで、コストのかかる計算を実行する前に、クライアントの IP アドレスを検証できる。ハンドシェイクが成功した後、サーバはクライアントに新しいトークンを発行できる。これにより、このコストを発生させることなく、新しいコネクションを確立できる。

21.1.1.3　オンパスハンドシェイクの終了

オンパスまたはオフパスの攻撃者は、Initial パケットを置換または競合させることによって、ハンドシェイクを強制的に失敗させることができる。有効な Initial パケットが交換されると、後続の Handshake パケットは Handshake キーで保護される。オンパスの攻撃者は、パケットを破棄してエンドポイントに試行を放棄させる以外に、ハンドシェイクエラーを強制することはできない。

オンパスの攻撃者は、いずれかの側のパケットのアドレスを置換することもできるため、クライアントまたはサーバがリモートアドレスを正しく認識できなくなる。このような攻撃は、NAT によって実行される機能と区別できない。

21.1.1.4　パラメータネゴシエーション

ハンドシェイク全体は暗号的に保護される。Initial パケットはバージョンごとのキーで暗号化され、Handshake 以降のパケットは TLS キー交換から派生したキーで暗号化される。さらに、パラメータネゴシエーションは TLS トランスクリプトに組み込まれるため、通常の TLS ネゴシエーションと同じ整合性が保証される。攻撃者は (バージョン固有の salt を知っている限り) クライアントのトランスポートパラメータを監視できるが、サーバのトランスポートパラメータは監視できず、パラメータネゴシエーションに影響を与えることはできない。

コネクション ID は暗号化されないが、すべてのパケットで整合性が保護される。

このバージョンの QUIC にはバージョンネゴシエーション機構は組み込まれていない。互換性のないバージョンの実装は単にコネクションの確立に失敗する。

21.1.2　保護されたパケット

パケット保護(第 12.1 節)は、Version Negotiation パケットを除くすべてのパケットに認証された暗号化を適用する。ただし、Initial パケットと Retry パケットは、バージョン固有のキー情報を使用するために保護が制限されている。詳細については、[QUIC-TLS] を参照。この項では、保護されたパケットに対するパッシブ攻撃とアクティブ攻撃について説明する。

オンパスとオフパスの両方の攻撃者は、パッシブ攻撃を行うことができる。この攻撃では、監視されたパケットを、将来のパケット保護に対するオフライン攻撃のために保存する。これは、任意のネットワーク上の任意のパケットのすべてのオブザーバに当てはまる。

コネクションの有効なパケットを監視できずにパケットを挿入する攻撃者は、成功する可能性が低い。これは、パケット保護によって、ハンドシェイク中に確立されたキーマテリアルを持つエンドポイントによってのみ有効なパケットが生成されることが保証されるためである(第 7 章および第 21.1.1 項を参照)。同様に、パケットを監視し、それらのパケットに新しいデータを挿入したり、既存のデータを変更しようとするアクティブな攻撃者は、Initial パケット以外の受信エンドポイントによって有効と見なされるパケットを生成できないようにする必要がある。

アクティブな攻撃者が転送または挿入するパケットの保護されていない部分 (送信元または宛先アドレスなど) を書き換えるスプーフィング攻撃は、攻撃者が元のエンドポイントにパケットを転送できる場合にのみ有効である。パケット保護により、パケットペイロードはハンドシェイクを完了したエンドポイントによってのみ処理され、無効なパケットはそれらのエンドポイントによって無視される。

攻撃者は、パケットと UDP データグラム間の境界を変更して、複数のパケットを 1 つのデータグラムに

結合したり、結合されたパケットを複数のデータグラムに分割したりすることもできる。パディングを必要とする Initial パケットを含むデータグラムを除き、データグラム内のパケットの配置方法を変更しても、一部のパフォーマンス特性が変更される可能性はあるが、コネクションに機能的な影響はない。

21.1.3 コネクションの移行

コネクションの移行 (第 9 章) は、非プローブフレームの送受信に一度に 1 つのパスを使用して、複数のパス上の IP アドレスとポート間を移行する機能をエンドポイントに提供する。パス検証(第 8.2 節)は、ピアが特定のパスで送信されたパケットを受信する意思があり、受信できることを確立する。これは、スプーフィングされたアドレスに送信されるパケットの数を制限することで、アドレススプーフィングの影響を軽減するのに役立つ。

この項では、さまざまな種類の DoS 攻撃下でのコネクション移行の意図されたセキュリティプロパティについて説明する。

21.1.3.1 オンパスアクティブ攻撃

監視対象のパケットが意図した宛先に到達しなくなる可能性がある攻撃者は、オンパス攻撃者と見なされる。クライアントとサーバの間に攻撃者が存在する場合、エンドポイントは、特定のパスでコネクションを確立するために、攻撃者を介してパケットを送信する必要がある。

オンパス攻撃者は次のことができる。

- パケットの検査
- IP および UDP パケットヘッダの変更
- 新しいパケットの挿入
- パケットの遅延
- パケットの並べ替え
- パケットの破棄
- パケット境界に沿ったデータグラムの分割とマージ

オンパス攻撃者は次のことができない。

- パケットの認証済み部分を変更し、受信側にパケットを受け入れさせる

オンパス攻撃者は、監視しているパケットを変更することができる。ただし、パケットの認証済み部分を変更すると、受信側エンドポイントによって無効として破棄される。これは、パケットペイロードが認証され、暗号化されているためである。

QUIC は、オンパス攻撃者の機能を次のように制限することを目的としている。

1. オンパス攻撃者は、コネクションにパスを使用できないようにして、攻撃者が含まれていない別のパスを使用できない場合にコネクションを失敗させることができる。これを実現するには、すべてのパケットを削除するか、復号化に失敗するようにパケットを変更するか、またはその他の方法を使用する。
2. オンパス攻撃者は、新しいパスでパスの検証を失敗させることによって、攻撃者もオンパスである新しいパスへの移行を防ぐことができる。
3. オンパス攻撃者は、攻撃者がオンパスではないパスへのクライアントの移行を防ぐことはできない。
4. オンパス攻撃者は、パケットを遅延または破棄することによって、コネクションのスループットを低下させることができる。
5. オンパス攻撃者は、パケットの認証部分を変更したパケットをエンドポイントに受け入れさせることはできない。

21.1.3.2　オフパスアクティブ攻撃

オフパス攻撃者は、クライアントとサーバ間のパス上に直接存在しないが、クライアントとサーバ間で送信されたパケットの一部またはすべてのコピーを取得できる可能性がある。また、これらのパケットのコピーをいずれかのエンドポイントに送信することもできる。

オフパス攻撃者は、次のことを行うことができる。

- パケットの検査
- 新しいパケットの注入
- 注入されたパケットの順序変更

オフパス攻撃者は、次のことができない。

- エンドポイントによって送信されたパケットの変更
- パケットの遅延
- パケットの破棄
- 元のパケットの並べ替え

オフパス攻撃者は、監視したパケットの変更されたコピーを作成し、それらのコピーをネットワークに挿入できる。これには、スプーフィングされた送信元アドレスと宛先アドレスが含まれる可能性がある。

この説明では、オフパス攻撃者が、攻撃者によって監視された元のパケットが到着する前に、宛先エンドポイントに到達するパケットの変更されたコピーをネットワークに挿入できることを前提としている。つまり、攻撃者はエンドポイント間の正当なパケットとの競合に一貫して"勝つ"ことができ、受信側によって元のパケットが無視される可能性がある。

また、攻撃者が NAT 状態に影響を与えるために必要なリソースを持っていることも想定される。特に、攻撃者はエンドポイントの NAT バインドを失わせ、独自のトラフィックで使用するために同じポートを取得する可能性がある。

QUIC は、次のようにオフパス攻撃者の機能を制限することを目的としている。

1. オフパス攻撃者は、パケットを競合させ、「限定された」オンパス攻撃者になろうとする可能性がある。
2. オフパス攻撃者は、クライアントとサーバ間のコネクションを改善できる限り、オフパス攻撃者としてリストされているソースアドレスを持つ転送されたパケットのパス検証を成功させることができる。
3. オフパス攻撃者は、ハンドシェイクが完了した後にコネクションを閉じることはできない。
4. オフパス攻撃者は、新しいパスを監視できない場合、新しいパスへの移行を失敗させることはできない。
5. オフパス攻撃者は、オフパス攻撃者でもある新しいパスへの移行中に、制限付きオンパス攻撃者になることができない。
6. オフパス攻撃者は、クライアントが最初に使用したのと同じ IP アドレスとポートからサーバにパケットを送信するように共有 NAT 状態に影響を与えることで、制限付きオンパス攻撃者になることができる。

21.1.3.3　制限付きオンパスアクティブ攻撃

制限付きオンパス攻撃者は、サーバとクライアントの間で元のパケットを複製して転送することでパケットのルーティングを改善するオフパス攻撃者である。これにより、元のパケットが宛先エンドポイントによって破棄されるように、元のコピーよりも先にこれらのパケットが到着する。

制限付きオンパス攻撃者は、エンドポイント間の元のパスに存在しないため、エンドポイントによって送信された元のパケットが宛先に到達するという点で、オンパス攻撃者とは異なる。つまり、コピーされたパケットを元のパスよりも速く宛先にルーティングすることに失敗しても、元のパケットが宛先に到達するの

を防ぐことはできない。

制限付きオンパス攻撃者は、次のことを実行できる。

- パケットの検査
- 新しいパケットの挿入
- 暗号化されていないパケットヘッダの変更
- パケットの並べ替え

制限されたオンパス攻撃者は、次のことができない。

- 元のパスで送信されたパケットよりも後に到着するようにパケットを遅延させる
- パケットを破棄する
- パケットの認証および暗号化された部分を変更し、受信側がそのパケットを受け入れるようにする

制限付きオンパス攻撃者は、元のパケットが重複パケットよりも前に到着する時点までしかパケットを遅延させることができない。つまり、元のパスよりも遅延が悪いルーティングを提供することはできない。制限付きオンパス攻撃者がパケットを破棄しても、元のコピーは宛先エンドポイントに到着する。

QUIC は、制限付きオフパス攻撃者の機能を次のように制限することを目的としている。

1. 制限付きオンパス攻撃者は、ハンドシェイクが完了した後にコネクションを閉じることはできない。
2. 制限付きオンパス攻撃者は、クライアントが最初にアクティビティを再開する場合、アイドル状態のコネクションを閉じることはできない。
3. 制限付きオンパス攻撃者は、サーバが最初にアクティビティを再開する場合、アイドル状態のコネクションが失われたと見なさせることができる。

これらの保証は、同じ理由ですべての NAT に提供される保証と同じであることに注意すること。

21.2 ハンドシェークサービス拒否

暗号化および認証されたトランスポートとして、QUIC はサービス拒否に対するさまざまな保護を提供する。暗号ハンドシェイクが完了すると、QUIC エンドポイントは認証されていないほとんどのパケットを破棄し、攻撃者が既存のコネクションに干渉する能力を大幅に制限する。

コネクションが確立されると、QUIC エンドポイントは一部の認証されていない ICMP パケットを受け入れる場合があるが (第 14.2.1 項を参照)、これらのパケットの使用は非常に制限される。エンドポイントが受け入れる可能性のある他の種類のパケットは、ステートレスリセット(第 10.3 節)のみである。これは、トークンが使用されるまで秘密に保持されることに依存する。

コネクションの作成中、QUIC はネットワークパス外からの攻撃に対する保護のみを提供する。すべての QUIC パケットには、受信側がピアからの先行パケットを見たという証拠が含まれる。

アドレスはハンドシェイク中に変更できないため、エンドポイントは異なるネットワークパスで受信したパケットを破棄できる。

Source フィールドおよび Destination Connection ID フィールドは、ハンドシェイク中のパス外攻撃に対する保護の主な手段である（第 8.1 節を参照）。これらはピアによって設定されたものと一致する必要がある。Initial Reset と Stateless Reset を除き、エンドポイントは、エンドポイントが以前に選択した値と一致する Destination Connection ID フィールドを含むパケットのみを受け入れる。これは、Version Negotiation パケットに対して提供される唯一の保護である。

Initial パケットの Destination Connection ID フィールドは、予測できないようにクライアントによって選択される。これは、追加の目的を果たす。暗号ハンドシェイクを伝送するパケットは、このコネクション ID から派生したキーと、QUIC バージョンに固有のソルトで保護される。これにより、エンドポイントは、暗号ハンドシェイクの完了後に使用するのと同じプロセスを使用して、受信したパケットを認証できる。認証で

きないパケットは破棄される。この方法でパケットを保護すると、パケットの送信側が Initial パケットを確認して理解していることが確実になる。

これらの保護は、コネクションが確立される前に QUIC パケットを受信できる攻撃者に対して効果的であることを意図したものではない。このような攻撃者は、QUIC エンドポイントによって受け入れられるパケットを送信する可能性がある。このバージョンの QUIC はこの種の攻撃を検出しようとするが、エンドポイントが回復するのではなく、コネクションの確立に失敗することを想定している。ほとんどの場合、暗号化ハンドシェイクプロトコル [QUIC-TLS] はハンドシェイク中の改ざんの検出を担当する。

エンドポイントは、他の方法を使用してハンドシェイクの干渉を検出し、回復を試みることができる。無効なパケットは、他の方法を使用して識別および破棄できるが、このドキュメントでは特定の方法は規定されていない。

21.3 増幅攻撃

攻撃者は、サーバからアドレス検証トークン (第 8 章) を受信し、そのトークンを取得するために使用した IP アドレスを解放できる可能性がある。その後、攻撃者は、この同じアドレスをスプーフィングすることによって、サーバとの 0-RTT コネクションを開始できる。これは、別の (被害者) エンドポイントをアドレス指定する可能性がある。したがって、攻撃者は、サーバが最初の輻輳ウィンドウに相当するデータを被害者に送信する可能性がある。

サーバは、アドレス検証トークンの使用と有効期間を制限することによって、この攻撃の軽減策を提供する必要がある[SHOULD]（第 8.1.3 項を参照）。

21.4 楽観的 ACK 攻撃

受信していないパケットを確認するエンドポイントは、ネットワークがサポートする速度を超えて送信することを許可する輻輳コントローラを引き起こす可能性がある[MAY]。エンドポイントは、この動作を検出するためにパケットを送信するときにパケット番号をスキップする場合がある。エンドポイントは、PROTOCOL_VIOLATION タイプのコネクションエラーを使用して、すぐにコネクションを閉じることができる(第 10.2 節を参照)。

21.5 リクエスト偽造攻撃

リクエスト偽造攻撃は、エンドポイントによって制御されるリクエストを使用して、エンドポイントのピアが被害者に向けてリクエストを発行するように仕向けることで発生する。リクエスト偽造攻撃の目的は、攻撃者が利用できない可能性のあるピアの機能にアクセスできるようにすることである。ネットワークプロトコルの場合、リクエスト偽造攻撃は、多くの場合、ネットワーク内のピアの場所によって被害者がピアに付与した暗黙的な承認を悪用するために使用される。

リクエスト偽造が効果的であるためには、攻撃者はピアが送信するパケットとそのパケットの送信先に影響を与える必要がある。攻撃者が制御されたペイロードで脆弱なサービスを標的にできる場合、そのサービスは、攻撃者のピアに起因するが攻撃者によって決定されたアクションを実行する可能性がある。

たとえば、Web 上のクロスサイトリクエスト偽造 [CSRF] エクスプロイトは、クライアントが承認クッキー [COOKIE] を含む要求を発行し、あるサイトが別のサイトに制限されることを意図した情報とアクションにアクセスできるようにする。

QUIC は UDP 上で実行されるため、主に懸念される攻撃方式は、攻撃者が、ピアが UDP データグラムを送信するアドレスを選択し、それらのパケットの保護されていないコンテンツの一部を制御できる攻撃方式である。QUIC エンドポイントによって送信されるデータの多くは保護されるため、これには暗号テキストの制御が含まれる。攻撃者がピアに UDP データグラムを送信させ、そのホストがデータグラム内のコンテンツに基づいて何らかのアクションを実行する場合、攻撃は成功する。

この節では、QUICがリクエスト偽造攻撃に使用される可能性のある方法について説明する。

この節では、QUIC エンドポイントによって実装できる制限された対策についても説明する。これらの軽減策は、リクエスト偽造攻撃の潜在的なターゲットがアクションを実行することなく、QUIC の実装または展開によって一方的に採用できる。ただし、UDP ベースのサービスがリクエストを適切に承認しない場合、これらの対策は不十分である可能性がある。

第 21.5.4 項で説明されている移行攻撃は非常に強力であり、適切な対策がないため、QUIC サーバの実装では、攻撃者が任意の宛先に対して任意の UDP ペイロードを生成できることを想定する必要がある。QUIC サーバは、イングレスフィルタリング [BCP38] を展開しておらず、UDP エンドポイントのセキュリティが不十分なネットワークに展開するべきではない[SHOULD NOT]。

通常、クライアントが脆弱なエンドポイントと共存していないことを保証することはできないが、このバージョンの QUIC ではサーバの移行が許可されていないため、クライアントに対するなりすまし移行攻撃を防ぐことができる。サーバの移行を可能にする将来の拡張では、偽造攻撃に対する対策も定義しなければならない[MUST]。

21.5.1　エンドポイントの制御オプション

QUIC は攻撃者が、ピアが UDP データグラムを送信する場所に影響を与えたり制御したりする機会を提供する。

- 初期コネクションの確立 (第 7 章)。サーバは、クライアントがデータグラムを送信する場所を選択できる。たとえば、DNS レコードにデータを入力する。
- 優先アドレス(第 9.6 節)。サーバは、クライアントがデータグラムを送信するアドレスを選択できる。
- スプーフィングされたコネクション移行 (第 9.3.1 項) 。クライアントは、ソースアドレススプーフィングを使用して、サーバが後続のデータグラムを送信するアドレスを選択できる。
- サーバに Version Negotiation パケットを送信させるスプーフィングされたパケット (第 21.5.5 項) 。

すべての場合において、攻撃者はピアに、QUIC を理解していない可能性のある被害者にデータグラムを送信させることができる。つまり、これらのパケットはアドレス検証の前にピアによって送信される(第 8 章を参照)。

パケットの暗号化された部分以外で、QUIC はエンドポイントに、ピアが送信する UDP データグラムの内容を制御するためのいくつかのオプションを提供する。Destination Connection ID フィールドは、ピアが送信するパケットの最初に現れるバイトを直接制御する(第 5.1 節を参照)。Initial パケットの Token フィールドは、Initial パケットの他のバイトをサーバが制御する(第 17.2.2 項を参照)。

このバージョンの QUIC には、パケットの暗号化された部分に対する間接的な制御を防ぐ手段はない。エンドポイントは、ピアが送信するフレーム、特に STREAM フレームなどのアプリケーションデータを伝送するフレームの内容を制御できると想定する必要がある。これはアプリケーションプロトコルの詳細にある程度依存するが、多くのプロトコル使用状況である程度の制御が可能である。攻撃者はパケット保護キーにアクセスできるため、ピアが将来のパケットを暗号化する方法を予測できる可能性がある。データグラムコンテンツの制御に成功するには、攻撃者がパケット内のパケット番号とフレームの配置をある程度の信頼性で予測できることが必要である。

この項では、データグラムコンテンツの制御を制限することは不可能であると想定している。以降の項の軽減策の焦点は、アドレス検証の前に送信されるデータグラムが要求の偽造に使用される方法を制限することである。

21.5.2　クライアントの Initial パケットを使用した要求の偽造

サーバとして機能する攻撃者は、その可用性をアドバタイズする IP アドレスとポートを選択できるため、

クライアントからの Initial パケットはこの種の攻撃で使用できると想定される。ハンドシェイクで暗黙的に行われるアドレス検証により、新しいコネクションの場合、クライアントは QUIC を理解していない、または QUIC コネクションを受け入れない宛先に他の種類のパケットを送信しない。

Initial パケット保護 ([QUIC-TLS] のセクション 5.2) は、サーバがクライアントによって送信される Initial パケットの内容を制御することを困難にする。クライアントが予測できない Destination Connection ID を選択すると、サーバはクライアントからの Initial パケットの暗号化された部分を制御できなくなる。

ただし、Token フィールドはサーバの制御に開放されており、サーバはクライアントを使用してリクエスト偽造攻撃を行うことができる。NEW_TOKEN フレーム (第 8.1.3 項) で提供されるトークンを使用すると、コネクションの確立中にリクエスト偽造の唯一のオプションが提供される。

ただし、クライアントは NEW_TOKEN フレームを使用する義務はない。サーバアドレスが NEW_TOKEN フレームを受信したときから変更されている場合に、クライアントが空の Token フィールドを送信すると、Token フィールドに依存するリクエスト偽造攻撃を回避できる。

サーバアドレスが変更された場合、クライアントは NEW_TOKEN フレームの使用を回避できる。ただし、Token フィールドを含めないと、パフォーマンスに悪影響を及ぼす可能性がある。サーバは NEW_TOKEN フレームに依存して、データ送信の制限の 3 倍を超えるデータを送信できる(第 8.1 節を参照)。特に、クライアントがサーバにデータを要求するために 0-RTT を使用する場合に影響する。

Retry パケットの送信 (第 17.2.5 項) は、Token フィールドを変更するオプションをサーバに提供する。Retry パケットを送信した後、サーバはクライアントからの後続の Initial パケットの Destination Connection ID フィールドを制御することもできる。これにより、Initial パケットの暗号化された内容を間接的に制御できる場合もある。ただし、Retry パケットの交換によってサーバのアドレスが検証されるため、後続の Initial パケットがリクエスト偽造に使用されるのを防ぐことができる。

21.5.3　優先アドレスによるリクエスト偽造

サーバは優先アドレスを指定でき、クライアントはハンドシェイクを確認した後に優先アドレスに移行する(第 9.6 節を参照)。クライアントが優先アドレスに送信するパケットの Destination Connection ID フィールドは、リクエスト偽造に使用できる。

クライアントは、アドレスを検証する前に、非プローブフレームを優先アドレスに送信してはならない [MUST NOT]（第 8 章を参照）。これにより、データグラムの暗号化された部分を制御するためにサーバが持つオプションが大幅に減少する。

このドキュメントでは、優先アドレスの使用に固有で、エンドポイントで実装できる追加の対策は提供しない。第 21.5.6 項で説明されている一般的な対策は、さらなる軽減策として使用できる。

21.5.4　偽装された移行によるリクエストの偽造

クライアントは、明らかなコネクション移行の一部として偽装された送信元アドレスを提示し、サーバにそのアドレスにデータグラムを送信させることができる。

サーバがその後、この偽装されたアドレスに送信するすべてのパケットの Destination Connection ID フィールドは、リクエストの偽造に使用できる。クライアントは暗号文に影響を与えることもできる。

アドレス検証の前にプローブパケット(第 9.1 節)のみをアドレスに送信するサーバは、攻撃者にデータグラムの暗号化された部分に対する制限された制御しか提供しない。しかし、特に NAT 再バインドの場合、これはパフォーマンスに悪影響を与える可能性がある。サーバがアプリケーションデータを含むフレームを送信する場合、攻撃者はデータグラムのほとんどの内容を制御できる可能性がある。

このドキュメントでは、第 21.5.6 項で説明されている一般的な対策を除き、エンドポイントで実装できる特定の対策は提供しない。ただし、ネットワークレベルでのアドレススプーフィングの対策、特にイングレスフィルタリング [BCP38] は、スプーフィングを使用し、外部ネットワークから発信される攻撃に対して特

に効果的である。

21.5.5　バージョンネゴシエーションを使用したリクエストの偽造

パケットにスプーフィングされた送信元アドレスを提示できるクライアントは、サーバに Version Negotiation パケット (第 17.2.1 項) をそのアドレスに送信させることができる。

不明なバージョンのパケットのコネクション ID フィールドにサイズ制限がないため、結果のデータグラムからクライアントが制御するデータの量が増加する。このパケットの最初のバイトはクライアントの制御下になく、次の 4 バイトは 0 であるが、クライアントは 5 番目のバイトから始まる最大 512 バイトを制御できる。

この攻撃に対しては、一般的な保護 (第 21.5.6 項) が適用される可能性があるが、特定の対策は提供されていない。この場合、イングレスフィルタ処理 [BCP38] も有効である。

21.5.6　一般的なリクエスト偽造対策

リクエスト偽造攻撃に対する最も効果的な防御は、脆弱なサービスを変更して強力な認証を使用することである。ただし、これは常に QUIC の展開で制御できるものではない。この項では、QUIC エンドポイントが一方的に実行できるその他の手順について説明する。状況によっては、正当な使用を妨害または妨げる可能性があるため、これらの追加手順はすべて任意である。

ループバックインタフェース経由で提供されるサービスには、適切な認証がないことがよくある。エンドポイントは、コネクションの試行またはループバックアドレスへの移行を防ぐことができる[MAY]。エンドポイントは、同じサービスが以前に別のインタフェースで使用可能であった場合、またはアドレスがループバックアドレス以外のサービスによって提供された場合に、ループバックアドレスへのコネクションまたは移行を許可すべきではない[SHOULD NOT]。これらの機能に依存するエンドポイントは、これらの保護を無効にするオプションを提供できる。

同様に、エンドポイントは、グローバルアドレス、一意のローカルアドレス [RFC4193]、または非プライベートアドレスからリンクローカルアドレス [RFC4291] またはプライベートアドレス範囲 [RFC1918] 内のアドレスへの変更を、リクエスト偽造の可能性のある試行と見なすことができる。エンドポイントは、これらのアドレスの使用を完全に拒否することができるが、正当な使用を妨げる重大なリスクがある。エンドポイントは、特定の範囲内の未検証のアドレスにデータグラムを送信することが安全ではないことを示すネットワークに関する特定の知識がない限り、アドレスの使用を拒否すべきではない[SHOULD NOT]。

エンドポイントは、Initial パケットに NEW_TOKEN フレームからの値を含めないか、アドレスの検証を完了する前にパケット内のプローブフレームのみを送信することによって、リクエスト偽造のリスクを軽減することを選択できる[MAY]。これにより、攻撃者が Destination Connection ID フィールドを攻撃に使用することを防ぐことはできないことに注意すること。

エンドポイントは、要求偽造攻撃の脆弱なターゲットとなる可能性のあるサーバの場所に関する特定の情報を持っているとは限らない。ただし、時間の経過とともに、攻撃の一般的なターゲットである特定の UDP ポートや、攻撃に使用されるデータグラムの特定のパターンを特定できる可能性がある。エンドポイントは、宛先アドレスを検証する前に、これらのポートへのデータグラムの送信を回避するか、これらのパターンに一致するデータグラムを送信しないように選択することができる[MAY]。エンドポイントは、問題があることがわかっているパターンを含むコネクション ID を使用せずに破棄することができる[MAY]。

> 注意:エンドポイントは、検証済みのアドレスに送信するときに追加の処理を実行する必要がないため、エンドポイントを変更してこれらの保護を適用する方が、ネットワークベースの保護を展開するよりも効率的である。

21.6 Slowloris 攻撃

Slowloris [SLOWLORIS] として一般的に知られている攻撃は、ターゲットエンドポイントへの多くのコネクションを可能な限り長く開いたままにしようとする。これらの攻撃は、非アクティブのために閉じられないようにするために必要な最小限のアクティビティを生成することによって、QUIC エンドポイントに対して実行できる。これには、少量のデータを送信したり、送信側レートを制御するためにフロー制御ウィンドウを徐々に開いたり、高い損失率をシミュレートする ACK フレームを作成したりすることが含まれる。

QUIC の展開では、サーバが許可するクライアントの最大数の増加、1 つの IP アドレスで許可されるコネクションの数の制限、コネクションが許可される最小転送速度の制限、エンドポイントがコネクションを維持できる時間の制限など、Slowloris 攻撃の軽減策を提供すべきである[SHOULD]。

21.7 ストリームの断片化および再構成攻撃

敵対的な送信者(送信側)は、ストリームデータの一部を意図的に送信せず、受信者(受信側)が未送信データのリソースをコミットする可能性がある。これにより、受信バッファのメモリコミットメントが不釣り合いになったり、受信者(受信側)で大規模で非効率的なデータ構造が作成されたりする可能性がある。

敵対的な受信者(受信側)は、受信確認されていないストリームデータを再送信のために送信者(送信側)に格納するように強制するために、ストリームデータを含むパケットを意図的に受信確認しない可能性がある。

フロー制御ウィンドウが使用可能なメモリに対応している場合、受信者 (受信側) に対する攻撃は軽減される。ただし、一部の受信者 (受信側) はメモリをオーバーコミットし、実際に使用可能なメモリを超える集約内のフロー制御オフセットをアドバタイズする。オーバーコミット戦略は、エンドポイントが適切に動作している場合にパフォーマンスを向上させることができるが、エンドポイントをストリーム断片化攻撃に対して脆弱にする。

QUIC 展開では、ストリーム断片化攻撃の軽減策を提供すべきである[SHOULD]。軽減策は、メモリのオーバーコミットの回避、追跡データ構造のサイズの制限、STREAM フレームの再構築の遅延、再構築ホールの経過時間と期間に基づくヒューリスティックの実装、またはこれらの組み合わせで構成される。

21.8 ストリームコミットメント攻撃

敵対的エンドポイントは、多数のストリームをオープンし、エンドポイントの状態を使い果たす可能性がある。敵対的エンドポイントは、TCP の SYN フラッディング攻撃と同様の方法で、多数のコネクションに対してプロセスを繰り返す可能性がある。

通常、第 2.1 節で説明するように、クライアントはストリームを順番にオープンする。ただし、短い間隔で複数のストリームが開始されると、ストリームをオープンする STREAM フレームが順番どおりに受信されない可能性がある。より大きい番号のストリーム ID を受信すると、受信側は、間にある同じタイプのすべてのストリームをオープンする必要がある(第 3.2 節を参照)。したがって、新しいコネクションでは、ストリーム 400 万をオープンすると、100 万および 1 のクライアントが開始した双方向ストリームがオープンする。

アクティブなストリームの数は、第 4.6 節で説明されているように、受信したすべての MAX_STREAMS フレームによって更新される initial_max_streams_bidi および initial_max_streams_uni トランスポートパラメータによって制限される。これらの制限を適切に選択すると、ストリームコミット攻撃の影響が緩和される。ただし、制限を低く設定しすぎると、アプリケーションが多数のストリームをオープンすることが予想される場合にパフォーマンスに影響する可能性がある。

21.9 ピアによるサービス拒否

QUIC と TLS は両方とも、いくつかのコンテキストで正当な用途を持つフレームまたはメッセージを含むが、これらのフレームまたはメッセージは、コネクションの状態に観察可能な影響を与えることなく、ピア

に処理リソースを消費させるために悪用される可能性がある。

メッセージは、フロー制御制限に小さな増分を送信するなど、小さなまたは重要でない方法で状態を変更したり元に戻したりするためにも使用できる。

帯域幅の消費や状態への影響と比較して処理コストが不釣り合いに大きい場合、悪意のあるピアが処理能力を使い果たす可能性がある。

すべてのメッセージには正当な用途があるが、実装は進行状況に対する処理コストを追跡し、非生産的なパケットの過剰な量を攻撃の兆候として扱うべきである[SHOULD]。エンドポイントは、コネクションエラーまたはパケットの破棄によってこの状態に応答してもよい[MAY]。

21.10 明示的な輻輳通知攻撃

オンパス攻撃者は、送信側のレートに影響を与えるために、IP ヘッダの ECN フィールドの値を操作できる。[RFC3168] では、操作とその影響について詳しく説明している。

制限されたオンパス攻撃者は、送信側のレートに影響を与えるために、ECN フィールドを変更したパケットを複製して送信できる。重複したパケットが受信側によって破棄された場合、攻撃者はこの攻撃に成功するために、元のパケットに対して重複したパケットを競合させる必要がある。したがって、IP パケット内の少なくとも 1 つの QUIC パケットが正常に処理されない限り、QUIC エンドポイントは IP パケット内の ECN フィールドを無視する(第 13.4 節を参照)。

21.11 ステートレスリセットオラクル

ステートレスリセットは、TCP リセットインジェクションに類似したサービス拒否攻撃の可能性を生み出す。この攻撃は、攻撃者が選択されたコネクション ID を持つコネクションに対してステートレスリセットトークンを生成できる場合に可能である。このトークンを生成できる攻撃者は、同じコネクション ID を持つアクティブなコネクションをリセットできる。

たとえば、IP アドレスまたはポートを変更することによって、静的キーを共有する異なるインスタンスにパケットをルーティングできる場合、攻撃者はサーバにステートレスリセットを送信させることができる。この種のサービス拒否から防御するには、ステートレスリセットの静的キーを共有するエンドポイント (第 10.3.2 項を参照) は、コネクションがアクティブでなくなっていない限り、特定のコネクション ID を持つパケットが常にコネクション状態を持つインスタンスに到着するように配置する必要がある[MUST]。

より一般的には、対応するコネクション ID を持つコネクションが同じ静的キーを使用する任意のエンドポイントでアクティブになる可能性がある場合、サーバはステートレスリセットを生成してはならない[MUST NOT]。

動的負荷分散を使用するクラスターの場合、アクティブなインスタンスがコネクション状態を保持している間にロードバランサーの構成が変更される可能性がある。インスタンスがコネクション状態を保持している場合でも、ルーティングの変更とその結果としてのステートレスリセットにより、コネクションが終了する。パケットが正しいインスタンスにルーティングされる可能性がない場合は、コネクションがタイムアウトするのを待つよりもステートレスリセットを送信することを推奨する。ただし、これはルーティングが攻撃者の影響を受けない場合にのみ許容される。

21.12 バージョンダウングレード

このドキュメントでは、2 つのエンドポイント間で使用される QUIC バージョンをネゴシエートするために使用できる QUIC Version Negotiation パケット (第 6 章) を定義する。ただし、このドキュメントでは、このバージョンとそれ以降の将来のバージョンの間でこのネゴシエーションがどのように実行されるかは指定しない。特に、Version Negotiation パケットには、バージョンダウングレード攻撃を防ぐためのメカニズムは含まれていない。Version Negotiation パケットを使用する QUIC の将来のバージョンでは、バージョンダウン

グレード攻撃に対して堅牢なメカニズムを定義しなければならない[MUST]。

21.13　ルーティングによる標的型攻撃

展開では、攻撃者が特定のサーバインスタンスへの新しいコネクションを標的にする機能を制限する必要がある。理想的には、ルーティングの決定は、アドレスを含むクライアントが選択した値とは無関係に行われる。インスタンスが選択されると、後のパケットが同じインスタンスにルーティングされるように、コネクション ID を選択できる。

21.14　トラフィック解析

QUIC パケットの長さは、それらのパケットの内容の長さに関する情報を明らかにすることができる。PADDING フレームは、エンドポイントがパケットの内容の長さを隠すことができるように提供されている(第 19.1 節を参照)。

トラフィック解析を破ることは挑戦的であり、活発な研究の対象となっている。情報が漏洩する可能性があるのはパケット長だけではない。エンドポイントは、パケットのタイミングなど、他の偶発的なチャネルを通じて機密情報が漏洩する可能性もある。

２２．　IANA に関する考慮事項

このドキュメントは、QUIC でコードポイントを管理するためのいくつかのレジストリを確立する。これらのレジストリは、第 22.1 節で定義されている共通のポリシーセットで動作する。

22.1　QUIC レジストリの登録ポリシー

すべての QUIC レジストリは、コードポイントの仮登録と永久登録の両方を許可する。この節では、これらのレジストリに共通するポリシーについて説明する。

22.1.1　仮登録

コードポイントの仮登録は、QUIC の拡張機能を私的に使用および実験できるようにすることを目的としている。仮登録に必要なのは、コードポイント値と連絡先情報のみである。ただし、仮登録は、別の目的のために再利用および再割り当てすることができる。

仮登録には、[RFC8126] のセクション 4.5 で定義されているように、専門家レビューが必要である。指定された専門家は、残りのコードポイントスペースの過剰な割合または最初の未割り当て値 (第 22.1.2 項参照) の登録のみを拒否できることが推奨される。

仮登録には、登録が最後に更新された日付を示す Date フィールドが含まれる。仮登録の日付を更新する要求は、指定された専門家からのレビューなしで行うことができる。

すべての QUIC レジストリには、仮登録をサポートする次のフィールドが含まれている。

Value:　割り当てられたコードポイント。

Status:　"permanent"または"provisional"。

Specification:　値の公開されている仕様への参照。

Date:　登録の最終更新日。

Change Controller: 登録の定義を担当するエンティティ。

Contact:　登録者の連絡先の詳細。

Notes:　登録に関する補足事項。

仮登録では、Specification フィールドと Notes フィールド、および永久登録に必要な追加フィールドを省略することができる[MAY]。Date フィールドは、登録が作成または更新された日付に設定されるため、登録の要求の一部としては必要ない。

22.1.2 コードポイントの選択

QUIC レジストリからのコードポイントの新しい要求は、既存の割り当てと選択されたスペース内の最初の未割り当てコードポイントの両方を除外するランダムに選択されたコードポイントを使用する必要がある[SHOULD]。複数のコードポイントの要求は、連続した範囲を使用してもよい[MAY]。これにより、異なるセマンティクスが異なる実装によって同じコードポイントに起因するリスクを最小限に抑えられる。

最初の未割り当てコードポイントの使用は、Standards Action ポリシーを使用した割り当てのために予約されている([RFC8126] セクション 4.9 参照)。初期コードポイント割り当てプロセス [EARLY-ASSIGN] をこれらの値に使用することができる。

フレームタイプなど、可変長整数 (第 16 章参照) でエンコードされるコードポイントの場合、使用法が長いエンコードを持つことに特に敏感でない限り、4 または 8 バイト(つまり、2^{14} 以上の値)にエンコードされるコードポイントを使用する必要がある[SHOULD]。

QUIC レジストリにコードポイントを登録するアプリケーションは、登録の一部として要求されたコードポイントを含めることができる[MAY]。コードポイントが割り当てられておらず、登録ポリシーの要件が満たされている場合、IANA は選択されたコードポイントを割り当てなければならない[MUST]。

22.1.3 暫定コードポイントの再利用

レジストリまたはレジストリの一部 (可変長符号化を使用するコードポイントの 64-16383 範囲など) の領域を再利用するために、未使用の暫定登録をレジストリから削除する要求が行われることがある。これは、記録された日付が最も古いコードポイントに対してのみ実行する必要があり[SHOULD]、1 年未満に更新されたエントリは再利用されるべきではない[SHOULD NOT]。

コードポイントの削除要求は、指定された専門家によって確認されなければならない[MUST]。専門家は、コードポイントがまだ使用されているかどうかを判断しなければならない[MUST]。専門家は、コードポイントの使用が既知であるかどうかを判断するために、登録の連絡先に加えて、可能な限り広範なプロトコル実装者に連絡することが推奨される。専門家は、応答に少なくとも 4 週間の期間を与えることも推奨される。

この検索によってコードポイントの使用が特定された場合、または登録の更新が要求された場合、コードポイントは再利用されてはならない[MUST NOT]。代わりに、登録の日付が更新される。登録には、学習した関連情報を記録するメモが追加される場合がある。

コードポイントの使用が特定されず、登録の更新が要求されなかった場合、コードポイントはレジストリから削除されてもよい[MAY]。

このレビューと協議のプロセスは、仮登録を永久登録に変更する要求にも適用される。ただし、コードポイントが使用されていないかどうかを判断するのではなく、登録が展開された使用状況を正確に表しているかどうかを判断することが目的である。

22.1.4 永久登録

QUIC レジストリの永久登録では、特に指定されていない限り、Specification Required ポリシー ([RFC8126] セクション 4.6 参照) が使用される。指定された専門家は、仕様が存在し、容易にアクセスできることを確認する。専門家は、不適切、軽薄、または実質的に有害でない限り (単に美的に不快やアーキテクチャが疑わしいだけではない)、登録を承認することに偏っていることが推奨される。レジストリの作成では、永久登録に追加の制約を指定できる[MAY]。

レジストリの作成では、登録が異なる登録ポリシーによって管理されるコードポイントの範囲を識別できる[MAY]。たとえば、"QUIC Frame Types"レジストリ(第 22.4 節参照)には、0 から 63 の範囲のコードポイントに対してより厳しいポリシーがある。

永久登録に対するより厳しい要件は、影響を受けるコードポイントの仮登録を妨げるものではない。たと

えば、フレームタイプ 61 の仮登録が要求される可能性がある。

Standards Track パブリケーションによって行われるすべての登録は、永続的でなければならない[MUST]。

このドキュメントのすべての登録には永続的なステータスが割り当てられ、IETF の変更管理者と QUIC Working Group (quic@ietf.org) の連絡先が記載されている。

22.2 QUIC Versions レジストリ

IANA は"QUIC"の見出しの下に"QUIC Versions"のレジストリを追加した。

"QUIC Versions"レジストリは 32 ビット空間を管理する(第 15 章参照)。このレジストリは、第 22.1 節の登録ポリシーに従う。このレジストリの永久登録は、Specification Required ポリシー ([RFC8126] セクション 4.6 参照) を使用して割り当てられる。

プロトコルのコードポイント 0x00000001 は、このドキュメントで定義されているプロトコルに永続的な状態で割り当てられる。コードポイント 0x00000000 は永続的に予約されている。このコードポイントの注記は、このバージョンがバージョンネゴシエーション用に予約されていることを示している。

パターン 0x?a?a?a?a は予約されており、IANA によって割り当てられてはならない[MUST NOT]。また、割り当てられた値のリストに表示されてはならない[MUST NOT]。

22.3 QUIC トランスポートパラメータレジストリ

IANA は"QUIC"の見出しの下に"QUIC Transport Parameters"のレジストリを追加した。

"QUIC Transport Parameters"レジストリは、62 ビット空間を管理する。このレジストリは、第 22.1 節の登録ポリシーに従う。このレジストリの永久登録は、Specification Required ポリシー ([RFC8126] セクション 4.6 参照) を使用して割り当てられる。ただし、0x00 と 0x3f (16 進数) の間の値は、[RFC8126] セクション 4.9 およびセクション 4.10 で定義されている Standards Action ポリシーまたは IESG 承認を使用して割り当てられる。

第 22.1.1 項に記載されているフィールドに加えて、このレジストリの永久登録には次のフィールドが含まれていなければならない[MUST]。

Parameter Name: パラメータの短いニーモニック。

このレジストリの初期内容を表 6 に示す。

表 6 初期 QUIC トランスポートパラメータレジストリエントリ

値	パラメータ名	仕様
0x00	original_destination_connection_id	第 18.2 節
0x01	max_idle_timeout	第 18.2 節
0x02	stateless_reset_token	第 18.2 節
0x03	max_udp_payload_size	第 18.2 節
0x04	initial_max_data	第 18.2 節
0x05	initial_max_stream_data_bidi_local	第 18.2 節
0x06	initial_max_stream_data_bidi_remote	第 18.2 節
0x07	initial_max_stream_data_uni	第 18.2 節
0x08	initial_max_streams_bidi	第 18.2 節
0x09	initial_max_streams_uni	第 18.2 節
0x0a	ack_delay_exponent	第 18.2 節
0x0b	max_ack_delay	第 18.2 節
0x0c	disable_active_migration	第 18.2 節

0x0d	preferred_address	第 18.2 節
0x0e	active_connection_id_limit	第 18.2 節
0x0f	initial_source_connection_id	第 18.2 節
0x10	retry_source_connection_id	第 18.2 節

N の整数値の形式 31*N+27 の各値(つまり、27、58、89、...)は予約されている。これらの値は IANA によって割り当てられてはならない[MUST NOT]、また、割り当てられた値のリストに表示されてはならない[MUST NOT]。

22.4 QUIC フレームタイプレジストリ

IANA は"QUIC"の見出しの下に"QUIC Frame Types"のレジストリを追加した。

"QUIC Frame Types"レジストリは 62 ビット空間を管理する。このレジストリは、第 22.1 節の登録ポリシーに従う。このレジストリの永久登録は、Specification Required ポリシー ([RFC8126] セクション 4.6 参照) を使用して割り当てられる。ただし、0x00 と 0x3f (16 進数) の間の値は、[RFC8126] のセクション 4.9 およびセクション 4.10 で定義されている Standards Action ポリシーまたは IESG 承認を使用して割り当てられる。

第 22.1.1 項に記載されているフィールドに加えて、このレジストリの永久登録には次のフィールドを含めなければならない[MUST]。

Frame Type Name: フレームタイプの短いニーモニック。

第 22.1 節のアドバイスに加えて、新しい永久登録の仕様では、エンドポイントが識別されたタイプのフレームを送信できることを判断する手段を記述する必要がある[SHOULD]。ほとんどの登録には、付随するトランスポートパラメータ登録が必要である(第 22.3 節参照)。永久登録の仕様では、フレーム内のフィールドの形式と割り当てられたセマンティクスも記述する必要がある。

このレジストリの初期内容を表 3 に示す。レジストリには、表 3 の"Pkts"列と"Spec"列が含まれていないことに注意。

22.5 QUIC Transport Error Codes レジストリ

IANA は、"QUIC"の見出しの下に"QUIC Transport Error Codes"のレジストリを追加した。

"QUIC Transport Error Codes"レジストリは、62 ビットの領域を管理する。この領域は、異なるポリシーによって管理される 3 つの範囲に分割される。このレジストリの永久登録は、Specification Required ポリシー ([RFC8126] セクション 4.6 参照) を使用して割り当てられる。ただし、0x00 と 0x3f (16 進数) の間の値は、[RFC8126] のセクション 4.9 およびセクション 4.10 で定義されている Standards Action ポリシーまたは IESG 承認を使用して割り当てられる。

第 22.1.1 項に記載しているフィールドに加えて、このレジストリの永久登録には、次のフィールドを含めなければならない[MUST]。

Code: パラメータの短いニーモニック。

Description: エラーコードのセマンティクスの簡単な説明。仕様参照が提供されている場合は、要約になることがある[MAY]。

このレジストリの初期内容を表 7 に示す。

表 7 初期 QUIC トランスポートエラーコードレジストリエントリ

値	コード名	説明	仕様
0x00	NO_ERROR	エラーなし	第 20 章
0x01	INTERNAL_ERROR	実装エラー	第 20 章

0x02	CONNECTION_REFUSED	サーバがコネクションを拒否	第 20 章
0x03	FLOW_CONTROL_ERROR	フロー制御エラー	第 20 章
0x04	STREAM_LIMIT_ERROR	オープンしたストリームの上限超過	第 20 章
0x05	STREAM_STATE_ERROR	無効なストリーム状態でフレーム受信	第 20 章
0x06	FINAL_SIZE_ERROR	最終サイズへの変更	第 20 章
0x07	FRAME_ENCODING_ERROR	フレームエンコードエラー	第 20 章
0x08	TRANSPORT_PARAMETER_ERROR	トランスポートパラメータエラー	第 20 章
0x09	CONNECTION_ID_LIMIT_ERROR	受信したコネクション ID の上限超過	第 20 章
0x0a	PROTOCOL_VIOLATION	汎用プロトコル違反	第 20 章
0x0b	INVALID_TOKEN	無効なトークンの受信	第 20 章
0x0c	APPLICATION_ERROR	アプリケーションエラー	第 20 章
0x0d	CRYPTO_BUFFER_EXCEEDED	CRYPTO データバッファのオーバーフロー	第 20 章
0x0e	KEY_UPDATE_ERROR	無効なパケット保護の更新	第 20 章
0x0f	AEAD_LIMIT_REACHED	パケット保護キーの過剰使用	第 20 章
0x10	NO_VIABLE_PATH	有効なネットワークパスが存在しない	第 20 章
0x0100-0x01ff	CRYPTO_ERROR	TLS アラートコード	第 20 章

２３． 参考資料

23.1 標準参照

[BCP38] Ferguson, P. and D. Senie, "Network Ingress Filtering: Defeating Denial of Service Attacks which employ IP Source Address Spoofing", BCP 38, RFC 2827, May 2000. https://www.rfc-editor.org/info/bcp38

[DPLPMTUD] Fairhurst, G., Jones, T., Tüxen, M., Rüngeler, I., and T. Völker, "Packetization Layer Path MTU Discovery for Datagram Transports", RFC 8899, DOI 10.17487/RFC8899, September 2020, <https://www.rfc-editor.org/info/rfc8899>.

[EARLY-ASSIGN] Cotton, M., "Early IANA Allocation of Standards Track Code Points", BCP 100, RFC 7120, DOI 10.17487/RFC7120, January 2014, <https://www.rfc-editor.org/info/rfc7120>.

[IPv4] Postel, J., "Internet Protocol", STD 5, RFC 791, DOI 10.17487/RFC0791, September 1981, <https://www.rfc-editor.org/info/rfc791>.

[QUIC-INVARIANTS] Thomson, M., "Version-Independent Properties of QUIC", RFC 8999, DOI 10.17487/RFC8999, May 2021, <https://www.rfc-editor.org/info/rfc8999>.

[QUIC-RECOVERY] Iyengar, J., Ed. and I. Swett, Ed., "QUIC Loss Detection and Congestion Control", RFC 9002, DOI 10.17487/RFC9002, May 2021, <https://www.rfc-editor.org/info/rfc9002>.

[QUIC-TLS] Thomson, M., Ed. and S. Turner, Ed., "Using TLS to Secure QUIC", RFC 9001, DOI 10.17487/RFC9001, May 2021, <https://www.rfc-editor.org/info/rfc9001>.

[RFC1191] Mogul, J. and S. Deering, "Path MTU discovery", RFC 1191, DOI 10.17487/RFC1191, November 1990, <https://www.rfc-editor.org/info/rfc1191>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC3168] Ramakrishnan, K., Floyd, S., and D. Black, "The Addition of Explicit Congestion Notification (ECN) to IP", RFC 3168, DOI 10.17487/RFC3168, September 2001, <https://www.rfc-editor.org/info/rfc3168>.

[RFC3629] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, DOI 10.17487/RFC3629, November 2003, <https://www.rfc-editor.org/info/rfc3629>.

[RFC6437] Amante, S., Carpenter, B., Jiang, S., and J. Rajahalme, "IPv6 Flow Label Specification", RFC 6437, DOI 10.17487/RFC6437, November 2011, <https://www.rfc-editor.org/info/rfc6437>.

[RFC8085] Eggert, L., Fairhurst, G., and G. Shepherd, "UDP Usage Guidelines", BCP 145, RFC 8085, DOI 10.17487/RFC8085, March 2017, <https://www.rfc-editor.org/info/rfc8085>.

[RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC8126, DOI 10.17487/RFC8126, June 2017, <https://www.rfc-editor.org/info/rfc8126>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[RFC8201] McCann, J., Deering, S., Mogul, J., and R. Hinden, Ed., "Path MTU Discovery for IP version 6", STD 87, RFC 8201, DOI 10.17487/RFC8201, July 2017, <https://www.rfc-editor.org/info/rfc8201>.

[RFC8311] Black, D., "Relaxing Restrictions on Explicit Congestion Notification (ECN) Experimentation", RFC 8311, DOI 10.17487/RFC8311, January 2018, <https://www.rfc-editor.org/info/rfc8311>.

[TLS13] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <https://www.rfc-editor.org/info/rfc8446>.

[UDP] Postel, J., "User Datagram Protocol", STD 6, RFC 768, DOI 10.17487/RFC0768, August 1980, <https://www.rfc-editor.org/info/rfc768>.

23.2 参考文献

[AEAD] McGrew, D., "An Interface and Algorithms for Authenticated Encryption", RFC 5116, DOI 10.17487/RFC5116, January 2008, <https://www.rfc-editor.org/info/rfc5116>.

[ALPN] Friedl, S., Popov, A., Langley, A., and E. Stephan, "Transport Layer Security (TLS) Application-Layer Protocol Negotiation Extension", RFC 7301, DOI 10.17487/RFC7301, July 2014, <https://www.rfc-editor.org/info/rfc7301>.

[ALTSVC] Nottingham, M., McManus, P., and J. Reschke, "HTTP Alternative Services", RFC 7838, DOI 10.17487/RFC7838, April 2016, <https://www.rfc-editor.org/info/rfc7838>.

[COOKIE] Barth, A., "HTTP State Management Mechanism", RFC 6265, DOI 10.17487/RFC6265, April 2011, <https://www.rfc-editor.org/info/rfc6265>.

[CSRF] Barth, A., Jackson, C., and J. Mitchell, "Robust defenses for cross-site request forgery", Proceedings of the 15th ACM conference on Computer and communications security - CCS '08, DOI 10.1145/1455770.1455782, 2008, <https://doi.org/10.1145/1455770.1455782>.

[EARLY-DESIGN] Roskind, J., "QUIC: Multiplexed Stream Transport Over UDP", 2 December 2013, <https://docs.google.com/document/d/1RNHkx_VvKWyWg6Lr8SZ-saqsQx7rFV-ev2jRFUoVD34/edit?usp=sharing>.

[GATEWAY] Hätönen, S., Nyrhinen, A., Eggert, L., Strowes, S., Sarolahti, P., and M. Kojo, "An experimental study of home gateway characteristics", Proceedings of the 10th ACM SIGCOMM conference on Internet measurement - IMC '10, DOI 10.1145/1879141.1879174, November 2010, <https://doi.org/10.1145/1879141.1879174>.

[HTTP2] Belshe, M., Peon, R., and M. Thomson, Ed., "Hypertext Transfer Protocol Version 2 (HTTP/2)", RFC 7540, DOI 10.17487/RFC7540, May 2015, <https://www.rfc-editor.org/info/rfc7540>.

[IPv6] Deering, S. and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification", STD 86, RFC 8200, DOI 10.17487/RFC8200, July 2017, <https://www.rfc-editor.org/info/rfc8200>.

[QUIC-MANAGEABILITY] Kuehlewind, M. and B. Trammell, "Manageability of the QUIC Transport Protocol", Work in Progress, Internet-Draft, draft-ietf-quic-manageability-11, 21 April 2021, <https://tools.ietf.org/html/draft-ietf-quic-manageability-11>.

[RANDOM] Eastlake 3rd, D., Schiller, J., and S. Crocker, "Randomness Requirements for Security", BCP 106, RFC 4086, DOI 10.17487/RFC 4086, June 2005, <https://www.rfc-editor.org/info/rfc4086>.

[RFC1812] Baker, F., Ed., "Requirements for IP Version 4 Routers", RFC 1812, DOI 10.17487/RFC1812, June 1995, <https://www.rfc-editor.org/info/rfc1812>.

[RFC1918] Rekhter, Y., Moskowitz, B., Karrenberg, D., de Groot, G. J., and E. Lear, "Address Allocation for

Private Internets", BCP 5, RFC 1918, DOI 10.17487/RFC 1918, February 1996, <https://www.rfc-editor.org/info/rfc1918>.

[RFC2018] Mathis, M., Mahdavi, J., Floyd, S., and A. Romanow, "TCP Selective Acknowledgment Options", RFC 2018, DOI 10.17487/RFC2018, October 1996, <https://www.rfc-editor.org/info/rfc2018>.

[RFC2104] Krawczyk, H., Bellare, M., and R. Canetti, "HMAC: Keyed-Hashing for Message Authentication", RFC 2104, DOI 10.17487/RFC2104, February 1997, <https://www.rfc-editor.org/info/rfc2104>.

[RFC3449] Balakrishnan, H., Padmanabhan, V., Fairhurst, G., and M. Sooriyabandara, "TCP Performance Implications of Network Path Asymmetry", BCP 69, RFC 3449, DOI 10.17487/RFC3449, December 2002, <https://www.rfc-editor.org/info/rfc3449>.

[RFC4193] Hinden, R. and B. Haberman, "Unique Local IPv6 Unicast Addresses", RFC 4193, DOI 10.17487/RFC4193, October 2005, <https://www.rfc-editor.org/info/rfc4193>.

[RFC4291] Hinden, R. and S. Deering, "IP Version 6 Addressing Architecture", RFC 4291, DOI 10.17487/RFC4291, February 2006, <https://www.rfc-editor.org/info/rfc4291>.

[RFC4443] Conta, A., Deering, S., and M. Gupta, Ed., "Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification", STD 89, RFC 4443, DOI 10.17487/RFC4443, March 2006, <https://www.rfc-editor.org/info/rfc4443>.

[RFC4787] Audet, F., Ed. and C. Jennings, "Network Address Translation (NAT) Behavioral Requirements for Unicast UDP", BCP 127, RFC 4787, DOI 10.17487/RFC4787, January 2007, <https://www.rfc-editor.org/info/rfc4787>.

[RFC5681] Allman, M., Paxson, V., and E. Blanton, "TCP Congestion Control", RFC 5681, DOI 10.17487/RFC5681, September 2009, <https://www.rfc-editor.org/info/rfc5681>.

[RFC5869] Krawczyk, H. and P. Eronen, "HMAC-based Extract-and-Expand Key Derivation Function (HKDF)", RFC 5869, DOI 10.17487/RFC5869, May 2010, <https://www.rfc-editor.org/info/rfc5869>.

[RFC7983] Petit-Huguenin, M. and G. Salgueiro, "Multiplexing Scheme Updates for Secure Real-time Transport Protocol (SRTP) Extension for Datagram Transport Layer Security (DTLS)", RFC 7983, DOI 10.17487/RFC7983, September 2016, <https://www.rfc-editor.org/info/rfc7983>.

[RFC8087] Fairhurst, G. and M. Welzl, "The Benefits of Using Explicit Congestion Notification (ECN)", RFC8087, DOI 10.17487/RFC8087, March 2017, <https://www.rfc-editor.org/info/rfc8087>.

[RFC8981] Gont, F., Krishnan, S., Narten, T., and R. Draves, "Temporary Address Extensions for Stateless Address Autoconfiguration in IPv6", RFC 8981, DOI 10.17487/RFC8981, February 2021, <https://www.rfc-editor.org/info/rfc8981>.

[SEC-CONS] Rescorla, E. and B. Korver, "Guidelines for Writing RFC Text on Security Considerations", BCP 72, RFC 3552, DOI 10.17487/RFC3552, July 2003, <https://www.rfc-editor.org/info/rfc3552>.

[SLOWLORIS] "RSnake" Hansen, R., "Welcome to Slowloris - the low bandwidth, yet greedy and poisonous HTTP client!", June 2009, <https://web.archive.org/web/20150315054838/http://ha.ckers.org/slowloris/>.

付録 A　擬似コード

この章の擬似コードでは、サンプルアルゴリズムについて説明する。これらのアルゴリズムは、最適なパフォーマンスよりも、正確で明確であることを目的としている。

この章の擬似コードセグメントは、Code Components としてライセンスされている。著作権通知を参照。

A.1　　可変長整数のデコードの例

図 45 の擬似コードは、バイトストリームから可変長整数を読み取る方法を示してる。関数 ReadVarint は、単一の引数 (ネットワークバイトオーダーで読み取ることができるバイトシーケンス) を受け取る。

```
ReadVarint(data):
  // The length of variable-length integers is encoded in the
  // first two bits of the first byte.
  v = data.next_byte()
  prefix = v >> 6
  length = 1 << prefix
  // Once the length is known, remove these bits and read any
  // remaining bytes.
  v = v & 0x3f
  repeat length-1 times:
    v = (v << 8) + data.next_byte()
  return v
```

図 45　可変長整数デコードアルゴリズムの例

たとえば、8 バイトのシーケンス 0xc2197c5eff14e88c は、十進数の値 151,288,809,941,952,652 にデコードされる。4 バイトのシーケンス 0x9d7f3e7d は、494,878,333 にデコードされる。2 バイトのシーケンス 0x7bbd は、15,293 にデコードされる。シングルバイトの 0x25 は、(2 バイトのシーケンス 0x4025 と同様に) 37 にデコードされる。

A.2　　パケット番号のエンコードアルゴリズムの例

図 46 の擬似コードは、実装がパケット番号エンコーディングに適切なサイズを選択する方法を示している。

EncodePacketNumber 関数は、次の 2 つの引数を取る。

- full_pn は、送信されるパケットの完全なパケット番号である。
- largest_acked は、現在のパケット番号空間でピアによって確認応答された最大のパケット番号である。

```
EncodePacketNumber(full_pn, largest_acked):
  // The number of bits must be at least one more
  // than the base-2 logarithm of the number of contiguous
  // unacknowledged packet numbers, including the new packet.
  if largest_acked is None:
    num_unacked = full_pn + 1
  else:     num_unacked = full_pn - largest_acked
  min_bits = log(num_unacked, 2) + 1
  num_bytes = ceil(min_bits / 8)
  // Encode the integer value and truncate to
  // the num_bytes least significant bytes.
  return encode(full_pn, num_bytes)
```

図 46　パケット番号エンコードアルゴリズムの例

たとえば、エンドポイントがパケット 0xabe8b3 の確認応答を受信し、番号 0xac5c02 のパケットを送信している場合、未処理のパケット番号は 29,519 (0x734f) ある。この範囲の少なくとも 2 倍(59,038 パケット、ま

たは 0xe69e)を表すには、16 ビットが必要である。

同じ状態で、数値が 0xace8fe のパケットを送信すると、24 ビットのエンコードが使用される。これは、範囲の倍を表すには少なくとも 18 ビットが必要であるためである(131,222 パケット、または 0x020096)。

A.3 パケット番号のデコードアルゴリズムの例

図 47 の擬似コードには、ヘッダ保護が削除された後にパケット番号をデコードするアルゴリズムの例が含まれている。

DecodePacketNumber 関数は次の 3 つの引数を取る。

- largest_pn は、現在のパケット番号空間で正常に処理された最大のパケット番号である。
- truncated_pn は、Packet Number フィールドの値である。
- pn_nbits は、Packet Number フィールドのビット数である(8、16、24、または 32)。

```
DecodePacketNumber(largest_pn, truncated_pn, pn_nbits):
   expected_pn  = largest_pn + 1
   pn_win       = 1 << pn_nbits
   pn_hwin      = pn_win / 2
   pn_mask      = pn_win - 1
   // The incoming packet number should be greater than
   // expected_pn - pn_hwin and less than or equal to
   // expected_pn + pn_hwin
   //
   // This means we cannot just strip the trailing bits from
   // expected_pn and add the truncated_pn because that might
   // yield a value outside the window.
   //
   // The following code calculates a candidate value and
   // makes sure it's within the packet number window.
   // Note the extra checks to prevent overflow and underflow.
   candidate_pn = (expected_pn & ~pn_mask) | truncated_pn
   if candidate_pn <= expected_pn - pn_hwin and
      candidate_pn < (1 << 62) - pn_win:
      return candidate_pn + pn_win
   if candidate_pn > expected_pn + pn_hwin and
      candidate_pn >= pn_win:
      return candidate_pn - pn_win
   return candidate_pn
```

図 47　パケット番号デコードアルゴリズムの例

たとえば、最も認証に成功したパケットのパケット番号が 0xa82f30ea の場合、16 ビット値 0x9b32 を含むパケットは 0xa82f9b32 としてデコードされる。

A.4 ECN 検証アルゴリズムの例

エンドポイントは、新しいネットワークパスで送信を開始するたびに、そのパスが ECN をサポートしているかどうかを判断する(第 13.4 節参照)。パスが ECN をサポートしている場合、目標は ECN を使用することである。エンドポイントは、ECN をサポートしていないと判断されたパスを定期的に再評価することもある。

この節では、新しいパスをテストするための 1 つの方法について説明する。このアルゴリズムは、パスが ECN をサポートしているかどうかをテストする方法を示すことを目的としている。エンドポイントは、さまざまな方法を実装できる。

パスには、"testing"状態、"unknown"状態、"failed"状態、または"capable"状態のいずれかの ECN 状態が割り当てられる。"testing"状態または"capable"状態のパスでは、エンドポイントは ECT マーキング (デフォルトで

は ECT (0)) を使用してパケットを送信する。それ以外の場合、エンドポイントはマークされていないパケットを送信する。

パスのテストを開始するには、ECN 状態が"testing"状態に設定され、既存の ECN カウントがベースラインとして記憶される。

テスト期間は、エンドポイントによって決定された数のパケットまたは制限された時間だけ実行される。目標は、テスト期間のゴールを制限することではなく、パスがマークされたパケットをどのように扱うかを明確に示すために、受信した ECN カウントに対して十分なマークされたパケットが送信されることを保証することである。第 13.4.2 項では、これを 10 パケットまたは PTO の 3 倍に制限することを提案している。

テスト期間が終了すると、パスの ECN 状態は"unknown"状態になる。"unknown"状態から、ACK フレームで ECN カウントの検証が成功すると (第 13.4.2.1 項参照)、マークされたパケットが確認されない限り、パスの ECN 状態は"capable"状態になる。

ECN カウントの検証がいずれかの時点で失敗した場合、影響を受けるパスの ECN 状態は"failed"状態になる。マークされたパケットがすべて失われたと宣言された場合、またはすべて ECN-CE マークされた場合、エンドポイントはパスの ECN 状態を"failed"状態とマークすることもできる。

このアルゴリズムに従うことで、ECN が適切にサポートされているパスで ECN が無効になることはほとんどない。マーキングを誤って変更したパスは、ECN を無効にする。マークされたパケットがパスによって破棄されるまれなケースでは、テスト期間が短いため、発生する損失の数が制限される。

IV.　RFC 8999 原文の著作権について

V.　RFC 8999 の和訳

RFC 8999　QUIC のバージョンに依存しないプロパティ

概要

本ドキュメントでは、すべてのバージョンの QUIC トランスポートプロトコルに共通するプロパティを定義する。

１．　QUIC の非常に抽象的な説明

QUIC は、2 つのエンドポイント間のコネクション指向プロトコルである。これらのエンドポイントは UDP データグラムを交換する。これらの UDP データグラムには QUIC パケットが含まれている。QUIC エンドポイントは、QUIC パケットを利用して QUIC コネクションを確立する。これは、各エンドポイント間で共有されるプロトコル状態である。

２．　すべての QUIC バージョンに共通する固定プロパティ

安全で多重化されたトランスポートを提供することに加えて、QUIC [QUIC-TRANSPORT] ではバージョンのネゴシエーションを行うオプションを使用できる。これにより、プロトコルは新たな要件に応じて時間の経過とともに変更できる。プロトコルの多くの特性は、バージョン間で変更される場合がある。

本ドキュメントは、新しいバージョンが開発・展開された場合にも安定して維持されることを意図とした QUIC のサブセットについて説明する。これらの不変条件は、すべて IP バージョンに依存しない。

本ドキュメントの主な目的は、新しいバージョンの QUIC をデプロイ可能であることを保証する点にある。変更できないプロパティを文書化することにより、QUIC エンドポイントがプロトコルの他の側面に対する変更をネゴシエートする能力を維持することを目的としている。その結果、エンドポイント以外のエンティティにも最小限の情報が保証される。本ドキュメントで特に禁止されていない限り、プロトコルのあらゆる側面は異なるバージョン間において変更可能である。

付録 A には、QUIC バージョン 1 の知識に基づいてなされがちな誤った仮定の完全ではないリストを含む。これらの仮定はすべての QUIC バージョンに適用されるわけではない。

３．　表記法と定義

本ドキュメントにおける"MUST"、"MUST NOT"、"REQUIRED"、"SHALL"、"SHALL NOT"、"SHOULD"、"SHOULD NOT"、"RECOMMENDED"、"NOT RECOMMENDED"、"MAY"、"OPTIONAL"は、ここに示したようにすべて大文字で記載されている場合に限り、BCP 14 [RFC2119] [RFC8174] で記載されたとおり解釈される。

本ドキュメントでは、規範的な表現が使われていない場合でも、将来の QUIC バージョンに対する要件を定義している。

本ドキュメントは、[QUIC-TRANSPORT] の用語および記法規則を使用している。

４．　表記規則

パケットの形式は、この章で定義する表記法を用いて記述される。この表記法は、[QUIC-TRANSPORT] で使用されているものと同一である。

複合フィールドには名前が付けられ、その後に対になる中かっこで囲まれたフィールドのリストが続く。このリストの各フィールドはカンマで区切られる。

個々のフィールドには、長さ情報に加え、固定値、オプション、または繰り返しに関する情報が含まれる。個々のフィールドでは、次の表記規則を使用する。長さはすべてビット単位である。

x (A) : x が A ビット長であることを示す。

x (A..B) : x が A から B までの任意の長さであることを示す。A を省略すると最小値が 0 ビットであることを示し、B を省略すると上限が設定されていないことを示す。この形式の値は常にバイト境界で終了する。

x (L) = C: x の固定値が C であることを示す。x の長さは L で記述され、上記のいずれの長さ形式も使用できる。

x (L) ...: x が 0 回以上繰り返され、各インスタンスの長さが L であることを示す。

本ドキュメントでは、ネットワークバイトオーダー(つまり、ビッグエンディアン)値を使用する。フィールドは、各バイトの上位ビットから順に配置される。

図 1 に、構造の例を示す。

```
Example Structure {
  One-bit Field (1),
  7-bit Field with Fixed Value (7) = 61,
  Arbitrary-Length Field (..),
  Variable-Length Field (8..24),
  Repeated Field (8) ...,
}
```

図 1　形式の例

５．　QUIC パケット

QUIC エンドポイントは、1 つ以上の QUIC パケットを含む UDP データグラムを交換する。本章では、QUIC パケットの不変特性について説明する。あるバージョンの QUIC では、1 つの UDP データグラム内に複数の QUIC パケットを許容し得るが、不変特性はデータグラム内の最初のパケットのみを記述する。

QUIC は 2 種類のパケットヘッダを定義する。ロングヘッダとショートヘッダである。ロングヘッダを持つパケットは、先頭バイトの最上位ビットが設定されていることで識別され、ショートヘッダを持つパケットは、そのビットがクリアされている。

QUIC パケットは、ヘッダを含め完全性が保護されている場合がある。ただし、QUIC の Version Negotiation パケットは完全性が保護されていない（第 6 章）。

ここで説明する値を除き、QUIC パケットのペイロードはバージョン固有であり、長さは任意である。

5.1　ロングヘッダ

ロングヘッダは、図 2 に示す形式を取る。

```
Long Header Packet {
  Header Form (1) = 1,
  Version-Specific Bits (7),
  Version (32),
  Destination Connection ID Length (8),
  Destination Connection ID (0..2040),
  Source Connection ID Length (8),
  Source Connection ID (0..2040),
  Version-Specific Data (..),
}
```

図 2　QUIC ロングヘッダ

ロングヘッダを持つ QUIC パケットは、先頭バイトの最上位ビットが 1 に設定されている。当該バイトのその他のビットはバージョン固有である。

次の 4 バイトには、32 ビットの Version フィールドが含まれる。Version については第 5.4 節で説明する。

次のバイトには、それに続く Destination Connection ID フィールドのバイト長が含まれる。この長さは 8 ビットの符号なし整数としてエンコードされる。Destination Connection ID フィールドは Destination Connection

ID Length フィールドの後に続き、長さは 0 から 255 バイトである。Connection ID については第 5.3 節で説明する。

次のバイトには、それに続く Source Connection ID フィールドのバイト長が含まれる。この長さは 8 ビットの符号なし整数としてエンコードされる。Source Connection ID フィールドは、Source Connection ID Length フィールドの後に続き、長さは 0 から 255 バイトである。

パケットの残りの部分には、バージョン固有の内容が含まれる。

5.2 ショートヘッダ

ショートヘッダは、図 3 に示す形式を取る。

```
Short Header Packet {
  Header Form (1) = 0,
  Version-Specific Bits (7),
  Destination Connection ID (..),
  Version-Specific Data (..),
}
```

図 3　QUIC ショートヘッダ

ショートヘッダを持つ QUIC パケットは、先頭バイトの最上位ビットは 0 に設定されている。

ショートヘッダを持つ QUIC パケットは、先頭バイトの直後に Destination Connection ID フィールドを含む。ショートヘッダには、Destination Connection ID Length フィールド、Source Connection ID Length フィールド、Source Connection ID フィールド、または Version フィールドの各フィールドは含まれない。Destination Connection ID フィールドの長さは、ショートヘッダを持つパケットでは、Destination Connection ID フィールドの長さはエンコードされず、本仕様では制約しない。

パケットの残りの部分はバージョン固有のセマンティクスを持つ。

5.3 コネクション ID

コネクション ID は、任意の長さの不透明なフィールドである。

コネクション ID の主な機能は、下位のプロトコル層(UDP、IP、その下位の層)でアドレス指定が変更されても、ある QUIC コネクション宛のパケットが間違った QUIC エンドポイントに配信されないようにすることである。コネクション ID は、エンドポイントおよびそれをサポートする中継局によって使用され、各 QUIC パケットがエンドポイントの正しいインスタンスに配信できるようにする。エンドポイントでは、コネクション ID は、そのパケットが意図された QUIC コネクションを識別するために用いられる。

コネクション ID は、各エンドポイントがバージョン固有の方法を用いて選択する。同じ QUIC コネクションに対するパケットであっても、異なるコネクション ID 値を使用する場合がある。

5.4 バージョン

Version フィールドには、4 バイトの識別子が含まれる。この値は、エンドポイントが QUIC バージョンを識別するために使用できる。値が 0x00000000 の Version フィールドは、バージョンネゴシエーションのために予約されている（第 6 章）。その他の値はすべて有効となり得る。

本ドキュメントで説明する特性は、QUIC のすべてのバージョンに適用される。このドキュメントで説明する特性に適合しないプロトコルは QUIC ではない。将来のドキュメントでは、特定の QUIC バージョン、または一定範囲の QUIC バージョンに適用される追加の特性について説明する可能性がある。

６．　バージョンネゴシエーション

ロングヘッダを持ち、かつエンドポイントが理解できない、またはサポートしていないバージョンのパケットを受信した QUIC エンドポイントは、応答として Version Negotiation パケットを送信する場合がある。

ショートヘッダを持つパケットは、バージョンネゴシエーションをトリガしない。

Version Negotiation パケットは、最初のバイトの最上位ビットを設定し、そのため第 5.1 節で定義されているロングヘッダを持つパケットの形式に準拠する。Version Negotiation パケットは、Version フィールドが 0x00000000 に設定されていることにより識別可能である。

```
Version Negotiation Packet {
  Header Form (1) = 1,
  Unused (7),
  Version (32) = 0,
  Destination Connection ID Length (8),
  Destination Connection ID (0..2040),
  Source Connection ID Length (8),
  Source Connection ID (0..2040),
  Supported Version (32) ...,
}
```

図 4　Version Negotiation パケット

Version Negotiation パケットの最初のバイトのうち、定義済みの値を持つのは最上位ビットのみである。残りの 7 ビット（"Unused"）は、送信時には任意の値に設定でき、受信時には無視しなければならない [MUST]。

Source Connection ID フィールドの後に、Version Negotiation パケットには、Supported Version フィールドの一覧を含む。各フィールドは、このパケットを送信するエンドポイントがサポートするバージョンを識別する。Version Negotiation パケットには、他のフィールドは含まれない。エンドポイントは、Supported Version フィールドを含まないパケット、または切り詰められた Supported Version 値を含むパケットを無視しなければならない [MUST]。

Version Negotiation パケットは、完全性または機密性の保護を使用しない。特定の QUIC バージョンには、エンドポイントがサポートされているバージョン集合の改ざんや破損を検出できるようにするプロトコル要素が含まれる場合がある。

エンドポイントは、受信したパケットの Source Connection ID フィールドの値を、Source Connection ID フィールドに含めなければならない [MUST]。Source Connection ID フィールドの値は、受信したパケットの Destination Connection ID フィールドからコピーしなければならない [MUST]。この値は、最初にクライアントによってランダムに選択される。両方のコネクション ID をエコーすることで、クライアントはサーバがパケットを受信したこと、ならびにパケットを監視できない攻撃者によって Version Negotiation パケットが生成されたものではないことについてある程度の保証を得られる。

Version Negotiation パケットを受信するエンドポイントは、以後のパケットに使用するバージョンを変更する場合がある。エンドポイントが QUIC バージョンを変更する条件は、エンドポイントが選択する QUIC のバージョンに依存する。

QUIC バージョン 1 をサポートするエンドポイントが Version Negotiation パケットを生成および処理する方法の詳細については、[QUIC-TRANSPORT]を参照のこと。

7.　セキュリティとプライバシーに関する考慮事項

ミドルボックスが特定のバージョンの QUIC の特性を観察し、他のバージョンの QUIC が類似の特性を示すときに、同じ基本的なセマンティクスが表現されていると仮定する可能性がある。このような特性は多く存在する可能性がある。付録 A を参照。QUIC バージョン 1 では、観測可能な特性の一部を排除または不明瞭化するための取り組みが行われているが、これらの多くは残っている。その他の QUIC バージョンは異なる設計上の判断を行い、異なる特性を示す可能性がある。

QUIC バージョン番号はすべての QUIC パケットに現れるわけではない。つまり、バージョン固有の特性に基づいてフローから情報を確実に抽出するには、ミドルボックスが観測するすべてのコネクション ID ごとに状態を保持する必要がある。

本ドキュメントで説明されている Version Negotiation パケットは完全性が保護されていない。攻撃者による挿入に対しては限定的な保護しか持たない。その結果として異なる QUIC バージョンを試行する場合、エンドポイントは Version Negotiation パケットのセマンティクスコンテンツを認証しなければならない[MUST]。

８．　参考資料

8.1　標準参照

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

8.2　参考文献

[QUIC-TLS] Thomson, M., Ed. and S. Turner, Ed., "Using TLS to Secure QUIC", RFC 9001, DOI 10.17487/RFC9001, May 2021, <https://www.rfc-editor.org/info/rfc9001>.

[QUIC-TRANSPORT] Iyengar, J., Ed. and M. Thomson, Ed., "QUIC: A UDP-Based Multiplexed and Secure Transport", RFC 9000, DOI 10.17487/RFC9000, May 2021, <https://www.rfc-editor.org/info/rfc9000>.

[RFC5116] McGrew, D., "An Interface and Algorithms for Authenticated Encryption", RFC 5116, DOI 10.17487/RFC5116, January 2008, <https://www.rfc-editor.org/info/rfc5116>.

付録 A　誤った仮定

QUIC バージョン 1 [QUIC-TRANSPORT] には、監視から保護されてはいないが、新しいバージョンがデプロイされるときに変更可能であると見なされる特性がいくつか存在する。

この章では、QUIC バージョン 1 の知識に基づいて QUIC について誤って想定され得る仮定の例を示す。これらの記述の中には、QUIC バージョン 1 に対してさえ当てはまらないものもある。これは網羅的なリストではなく、あくまで説明を目的したものである。

特定の QUIC バージョンでは、次の記述はいずれも偽になる可能性がある。

- QUIC は TLS [QUIC-TLS] を使用し、一部の TLS メッセージはネットワーク上で観測可能である。
- QUIC のロングヘッダは、コネクションの確立中にのみ交換される。
- 特定の 5 タプル上のあらゆるフローには、コネクションの確立フェーズが含まれる。
- フローで最初に交換されるパケットは、ロングヘッダを用いる。
- 長い休止期間の直前の最後のパケットには、確認応答のみが含まれる。
- QUIC は、コネクションの確立中に交換するパケットを保護するために、Authenticated Encryption with Associated Data (AEAD) 機能 (AEAD_AES_128_GCM;[RFC5116] 参照)を使用する。
- QUIC パケット番号は暗号化され、最初の暗号化バイトとして現れる。
- QUIC パケット番号は、送信するパケットごとに 1 ずつ増加する。
- QUIC には、クライアントが送信する最初のハンドシェイクパケットの最小サイズがある。
- QUIC は、クライアントが先に送信することを規定している。
- QUIC パケットでは常に、最初のバイトの 2 番目のビット (0x40) が設定されている。
- QUIC の Version Negotiation パケットはサーバのみが送信する。
- QUIC のコネクション ID はまれにしか変更されない。
- QUIC エンドポイントは、Version Negotiation パケットを送信されると、使用するバージョンを変更する。
- QUIC ロングヘッダの Version フィールドは、両方向で同一である。
- Version フィールドに特定の値を持つ QUIC パケットは、対応するバージョンの QUIC が使用中であることを意味する。
- 任意の QUIC エンドポイントのペア間では、一度に確立されるコネクションは 1 つだけである。