付録２

Transaction Internet Protocol Version 3.0

本ドキュメントは、RFC2371 の Transaction Internet Protocol Version 3.0 を日本語訳したものである。原文を参照する場合は、RFC2371 を入手して頂きたい。

このメモの位置付け

このドキュメントは、インターネットコミュニティのためのインターネット標準化過程プロトコルを記すと同時に、改善のための討論及び提案を要求するものである。標準化情勢と、このプロトコルの位置付けは、”インターネット公式プロトコル標準”（STD 1）の現行版を参照のこと。なお、このメモの配布に制限はない。

著作権表示

概要

別々のノードが共同で処理する多くのアプリケーションに対し、その処理の原子性（atomically）を保証する必要がある。すなわち、障害に直面しても、その処理を完了すべきかどうか、各ノードとも同じ結論に達する必要がある。このドキュメントは、上記処理を成し遂げるための、シンプルで容易に実装されるプロトコルを提案する。

1.はじめに

原子性を持つコミットメントの標準的方法として、2 フェーズ・コミット・ プロトコルがある。原子性を持つコミットメントと、2 フェーズ・コミット・プロトコルを紹介する[1]を参照せよ。

多くの２フェーズ・コミット・プロトコルが数年にわたり実装されてきたが、インターネット上で広く使われるようになったものは１つもない。その原因は主にそれらプロトコルの複雑さにある。複雑になった理由は、ほとんどの場合、2 フェーズ・コミット・プロトコルが特定のプログラム間通信プロトコルと一緒にバンドルされ、そのプロトコルが非常に大きなインフラの最上位に位置するためである。

このメモは、非常にシンプルな 2 フェーズ・コミット・プロトコルを提案する。そのシンプル性は、どのようにして別々のノードがトランザクションの結果に合意するか、だけを明確にすることにより成し遂げられる。すなわち、ノードが合意する内容が他のプロトコルを通して通信されることを許す（それどころか必要とする）。それにより、アプリケーション通信セマンティクス（semantics）と（ほんの少数個を名付ける）データ表現に関する発表が不要となる。トランザクション・マネージャーは、他のトランザクション・マネージャーと暗号メッセージの真正を証明するために、アプリケーション通信プロトコル

に関係なくトランスポート層セキュリティ・プロトコル[3]を使用してよい。

このプロトコルは、主に、1 つのインターネット・ノード上のトランザクション・マネージャが、別ノード上のトランザクション・マネージャと通信するために使われることが想像される。一方、トランザクション・マネージャーと会話するために、アプリケーション・プログラムと（または）リソース・マネージャーが、このプロトコルを使用可能な間は、この通信は一般的に内部ノードで行われ、ほとんどのトランザクション・マネージャーは、既にそのタスクのために充分なインタフェースを持つ。

このプロトコルが既存プロトコルから置き換わることを期待されていない間は、お互いに通信するために、多くの既存の異種トランザクション・マネージャーにとって、このプロトコルの実装が比較的簡単であることを強く期待する。

TIP プロトコルに関する補足情報が[5]にある。

2.使用例

今日では、電子ショッピング・バスケットは、多くの電子ストア・フロント（store-fronts）において、共通の暗喩（metaphor）である。顧客は電子カタログや選りすぐりの商品にざっと目を通して、電子ショッピング・バスケットの中にそれらを入れる。HTTP サーバ[2]は、クライアントのコンテキスト（例えば、顧客のショッピング・バスケット）の跡を追い、その後の顧客の要求でそれを再開するために、コンテキスト・クッキーへ符号化した URL で変動する多様な手段を提供する。

一度顧客がショッピングを終えたなら、顧客はその選択をコミットするか判断し、注文してよい。ほとんどの注文は、同じショッピング・トランザクションの一部として実行される場合を除き、お互いに関連しないかもしれない。それ以外の注文は、お互いに依存するかもしれない（例えば、特別な売り出しの一部として処理される場合）。その詳細にかかわりなく、顧客は全ての注文が明確に承認された上で首尾よく発行されることを期待する。今日の電子ストア・フロント（store-fronts）は、全ての注文のそのような発行を調整するために、独自で特別なプロトコルを実装しなければならない。このプログラム作成は、多様な電子ストア・フロント（store-fronts）を通して発行される場合に特に複雑になる。この複雑さは、インターネット・アプリケーションの潜在的な有用性を制限し、成長を抑制する。このドキュメントに記述されたプロトコルは、インターネット・サーバーが１つの協同作業（例えば、電子ショッピング・バスケットで注文発行）の合意を成し遂げるための、標準を提供するつもりである。注文を発行するサーバー（例えば CGI プログラム）は、ローカル・トランザクション・マネージャーを呼び出すトランザクションを開始して、その作業に参加する他のサーバーへそのトランザクションに加わるよう要求する。注文を発行するサーバーは、他のサーバーへ HTTP リクエスト上のユーザーデーターとして、トランザクションに問合わせ（reference）を渡す。他のサーバーは、ローカル・トランザクションを開始するために、そのサーバーのトランザクション・マネージャーを呼び出し、

このドキュメントで定義されるプロトコルを使って、マネージャーにリモート・トランザクションへ加わるよう要求する。一度、全ての注文が発行されたなら、２フェーズ・コミット・プロトコルの実行は、関係するトランザクション・マネージャーに委任される。もしトランザクションがコミットしたなら、全ての注文は首尾よく発行され、顧客は明確な承認を得る。もしトランザクションがアボートしたなら、注文は発行されず、顧客にはその問題が通知されるだろう。

トランザクション・サポートにより、例外処理と障害復旧が特別なコンポーネントに委任されるので、アプリケーションのプログラム作成がとても簡単になる。エンド・ユーザーも一部のみ成功した結果を扱う必要がなくなる。この例は、本プロトコルがどのようにHTTP サーバーによって使用されるか示したが、アプリケーションは（例えば ODBC を通して）リモート・データベースにアクセスするときや、他の既存プロトコル（例えば RPC）を使用してリモート・サービスを起動するときに本プロトコルを使用してよい。本プロトコルは、異なる通信プロトコルを使ったとしても、複数のアプリケーションが多様なネットワークで同じトランザクションに参加することを容易にする。

3.トランザクション

”トランザクション”は、実行されるコンピューター上の作業が、原子性のセマンティクス（semantics）を持つプログラミング・モデルに与えられる専門用語である。すなわち、全作業が完全に成功して変更が永久不変となる（トランザクション・コミット）か、一部でも作業が成功しなかったなら変更されない（トランザクション・アボート）か、のどちらかになる。トランザクションを構成する作業（1 つの作業）はアプリケーションによって定義される。

4.コネクション

トランザクション・インターネット・プロトコル（TIP）は、少ないコネクション設定コストとともに信頼できる順序保証のストリーム・トランスポートを要求する。インターネット（IP）環境では、TIP は TCP 上で動作し、安全で真正が証明されるコネクションを提供するために任意で TLS を使い、同じ TCP または TLS の上のライト・ウェイト（light-weight）コネクションを多重化するプロトコルを任意で使う。

トランザクションを共有するトランザクション・マネージャーが TCP（そして任意で TLS）コネクションを確立する。本プロトコルは、2 つのトランザクション・マネージャー間で共有される、それぞれの同時発生トランザクションに対し、異なるコネクションを使う。トランザクションが終了した後、そのコネクションは別のトランザクションでの再利用が可能である。

任意で、TCP または TLS コネクションを単一トランザクションのみに関連づける代わりに、2 つのトランザクション・マネージャーは、同じ TCP または TLS コネクション上のラ

イト・ウェイト（light-weight）コネクションを多重化するプロトコルに合意して、それぞれの同時発生トランザクションを別々のライト・ウェイト・コネクションに関連づけてもよい。ライト・ウェイト・コネクションを使用すると、実行中の同時発生トランザクションと関連づけられるレイテンシ（latency）と資源消費量を減らす。ここで記述された技術と同様の技術が、既存トランザクション処理システムによって広く使用されている。そのようなプロトコルの 1 例として付録 A を参照せよ。

TIP プロトコル自身が TCP と TLS の条件でのみ記述されているけれども、TIP と他のトランスポート・プロトコルの使用を排除していないことに注意せよ。しかし、選択されたトランスポートが、TCP と等しいセマンティクス（semantics）の提供を保証し、適切に TIP プロトコルへマッピングすることは実装者次第である。

このドキュメントにおいて、用語の ”コネクション ”または ”TCP コネクション ”は、TIP TCP コネクション、TIP TLS コネクション、または（TCP または TLS 上の）TIP 多重コネクションを指す。どれも違いは生じず、振る舞いは各ケースで同じものになる。コネクション・タイプで動作に違いがある箇所は、明確に示される。

5.トランザクション識別子

残念ながら、トランザクション識別子のフォーマットにおいて世界的に受け入れられる唯一の標準は存在しない。多様な標準と複数の専用フォーマットが存在する。TIP トランザクション識別子として許されるフォーマットは、以降の ”TIP URL ”セクションで記述される。トランザクション・マネージャーは、内部トランザクション識別子を適切な方法でこの TIP フォーマットへマッピングしてよい。さらに、上位（スーペリア）／下位（サブオーディネート）関係の中の各パーティは、トランザクションへ所有識別子を割り当て始める。それらの識別子は、関係が最初に確立されるときに交換される。このようにして、トランザクション・マネージャーは、内部でトランザクション識別子の独特なフォーマットを使用し始めるが、属する各上位（スーペリア）／下位（サブオーディネート）関係のために、外部のトランザクション識別子を覚えていなければならない。

6.プッシング（Pushing）対プリング（Pulling）トランザクション

ノード ”A ”上のプログラムがトランザクションを生成し、そのトランザクションの一部作業をノード ”B ”上のプログラムに実施してもらいたいケースを仮定せよ。これを実施する 2 つの古典的方法がある。 ”プッシュ（push） ”モデルと、 ”プル（pull） ”モデルを参照せよ。

”プッシュ（push） ”モデルでは、A 上のプログラムは最初に、ノード B へトランザクションをエクスポート（export）するよう、A のトランザクション・マネージャーへ依頼する。A のトランザクションマネージャーは、B の TM に対し A の下位（サブオーディネート）としてトランザクションをただちに始めるよう依頼するメッセージを送信し、そのト

ランザクション名を返す。A 上のプログラムは次に、“ ある作業を実施しなさい、そしてあなたのトランザクション・マネージャーが既に．．．という名前で知っている、トランザクションの一部に属しなさい ”という命令メッセージを、B 上の相当するプログラムへ送信する。A の TM は、B の TM へトランザクションを送信したことを知っているので、A の TM は 2 フェーズ・コミット処理において B の TM を含むことを知る。

“ プル（pull）”モデルでは、A 上のプログラムは単に、“ ある作業を実施しなさい、そして私の TM が．．．という名前で知るトランザクションの一部に属しなさい ”という命令メッセージを、B へ送信するだけである。B 上のプログラムは B の TM へ、そのトランザクションに参加するよう依頼する。その時 B の TM は、A から A 上のトランザクションを“ プル（pull）”する。このプルの結果として A の TM は、B の TM が 2 フェーズ・コミット処理に含まれることを知る。

ここで記述されるプロトコルは、“ プッシュ（push）”と、“ プル（pull）”モデルの両方をサポートする。

7.TIP トランザクション・マネージャー同一証明とコネクション確立

TIP トランザクション・マネージャーが接続するために、互いに身元確認と位置確認が実施できなければならない。これを実施するために必要な情報は、TIP トランザクション・マネージャー・アドレスによって示される。

［この仕様は、TIP トランザクション・マネージャーが初めにトランザクション・マネージャー・アドレスをどのように得るかについては規定しない（恐らく実装特有のコンフィグレーション・メカニズムによって実現する）。］

TIP トランザクション・マネージャー・アドレスは、次の形式になる。

<hostport><path>

<hostport>コンポーネントは、次の構成になる。

<host>[:<port>]

<host>は<dns name>または<ip address>のどちらかになる。そして<port>は、トランザクション・マネージャ（またはプロキシー（proxy））が TIP コネクションの確立要求を聴取（listen）するポートを指定する 10 進数である。ポート番号が省略されるなら、標準 TIP ポート番号（3372）が使用される。

<dns name>は、ドメイン名サービスに受け入れられる標準名である。それは、コマンドの受信者に便利であるように、十分に修飾されなければならない。

<ip address>は、IP アドレスであり、通常の形式では、ピリオド文字により分けられる 4 つの 10 進数になる。

<hostport>コンポーネントは、<path>コンポーネントの範囲（場所）を定義する。トランザクション・マネージャー・アドレスの<path>コンポーネントは、<hostport>により定義される位置において、特定の TIP トランザクション・マネージャーを見分けるデーターを含む。

<path>コンポーネントは、次の形式になる。

"/" [path_segments]

path_segments = segment *（ "/" segment ）
segment = *pchar *（ ";" param ）
param = *pchar

pchar = unreserved | escaped | ":" | "@" | "&" | "=" | "+"
unreserved = ASCII character octets with values in the range
（inclusive）: 48-57, 65-90, 97-122 | "$" | "-" | "_" |
"." | "!" | "~" | "*" | "'" | "（" | "）" | ","
escaped = "%" hex hex
hex = "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9" |
"A" | "B" | "C" | "D" | "E" | "F" | "a" | "b" | "c" | "d" |
"e" | "f"

<path>コンポーネントはシングル・スラッシュ”/”文字により分けられるパス・セグメントの連続から構成されてよい。パス・セグメントの内部では、”/”、”;”、”=”、そして”?”の文字は、リザーブされている。各パス・セグメントは、セミコロン”;”文字により示される、連続するパラメータを含んでよい。パラメータは、対応する問合せの解読（parse）には重要でない。

［トランザクション・マネージャー・アドレスの形式が URI[8]のために提案されたスキーム（scheme）に従うことを意図する。］

TIP トランザクション・マネージャー・アドレスは従って、TCP コネクションのために使用されるエンドポイント識別子（<hostport>）をコネクション開始（プライマリ）側へ与えて､特定の TIP トランザクション･マネージャー位置のために使用されるパス(<path>)をコネクション受信（セカンダリ）側へ与える。これがプライマリとセカンダリ TIP トランザクション・マネージャーの間で確立されるコネクションに必要とされる全ての情報である。

コネクションが確立された後、プライマリ・パーティは IDENTIFY コマンドを発行する。このコマンドは 2 つのトランザクション・マネージャー・アドレスである、プライマリ・

トランザクション・マネージャー・アドレスと、セカンダリ・トランザクション・マネージャー・アドレスを、パラメータとして含む。

プライマリー・トランザクション・マネージャー・アドレスは、コネクションを開始した TIP トランザクション・マネージャーを見分ける。この情報はコネクション障害後のケースで必要とされる。そのとき、コネクションの１パーティが２フェーズ・コミット・プロトコルを完了するために、他のパーティと新しいコネクションを再確立しなければならない。もしプライマリ・パーティがコネクション再確立を必要とするなら、そのジョブは簡単である。コネクションはオリジナル・コネクションと同じ方法で確立される。しかし、もしセカンダリ・パーティがコネクションの再確立を必要とするなら、オリジナル・コネクションの開始側とコンタクトをとる方法を知らされなければならない。この情報は、IDENTIFY コマンド上のプライマリ・トランザクション・マネージャー・アドレスを通してセカンダリに渡される。もしプライマリ・トランザクション・マネージャー・アドレスが渡されないなら、プライマリ・パーティーは再確立されるコネクションを必要とするアクションを実行してはいけない（例えば、回復アクションの実行）。

セカンダリ・トランザクション・マネージャー・アドレスは、受信側 TIP トランザクション・マネージャーを見分ける。中間プロキシーサーバーを通した TIP 通信のケースでは、正確に見分けた上で必要とされる TIP トランザクション・マネージャーに接続するために、プロキシーサーバーがこの URL を使用してよい。

8.TIP URL

トランザクションとトランザクション・マネージャーは、TIP プロトコルに関連付けられる資源である。トランザクション・マネージャー・アドレス・スキームを使用して、トランザクション・マネージャーとトランザクションの位置が決められる。一度コネクションが確立されたなら、それぞれのトランザクション・マネージャーに関連付けられるトランザクション上で処理する TIP コマンドを送信してよい。

リモート・ノードからトランザクションをプル（pull）したいアプリケーションは、リモート・トランザクションへ問合せを発行しなければならない。リモート・トランザクションは、ローカル・トランザクション・マネージャ（すなわち、トランザクションをプルするトランザクション・マネージャ）がリモート・トランザクション・マネージャへ接続して特定のトランザクションを識別することを許可する。トランザクションをプッシュしたいアプリケーションは、リモート・トランザクション・マネージャ（すなわち、トランザクションがプッシュされるトランザクション・マネージャ）へ問合せを発行しなければならない。リモート・トランザクション・マネージャは、ローカル・トランザクション・マネージャがリモート・トランザクション・マネージャの位置を突き止めることを許す。TIP プロトコルは URL スキーム[4]を定義する。URL スキームは、アプリケーションとトランザクション・マネージャが、トランザクション・マネージャとトランザクションに対する

問合せを交換することを許可する。

TIP URL は次の形式になる。

tip://<transaction manager address>?<transaction string>

<transaction manager address>は、TIP トランザクション・マネージャを識別して（上記７章で定義）、<transaction string>はトランザクション識別子を指定する。トランザクション識別子は 2 つの書式（標準または非標準）の 1 つを使用してよい。

ⅰ. "urn:" <NID> ":" <NSS>

URN のインターネット標準化提案（proposed Internet Standard）である RFC2141 に一致する標準トランザクション識別子。<NID>はネーム・スペース識別子（Namespace Identifier）であり、<NSS>はネーム・スペース特有文字列（Namespace Specific String）である。ネーム・スペース識別子は、ネーム・スペース特有文字列のシンタックス解釈方法を決める。 ネーム・スペース特有文字列は、（<NID>によって定義される）トランザクション識別子を表す文字列である。これらフィールドのコンテンツに対するルールは[6]により明確に示される（有効な文字、符号化等）。

標準的表現（standard representations）の見地から、グローバル・トランザクション識別子を表すために、この<トランザクション文字列>のフォーマットを使用してよい。<NID>の例として、<iso>や<xopen>がある。次に例を示す。

tip://123.123.123.123/?urn:xopen:xid

ネーム・スペース ID は、登録が必要なことに注意せよ。登録方法の詳細は[7]を参照せよ。

ⅱ. <transaction identifier>

トランザクション識別子を表す印刷可能な ASCII 文字列（値の範囲が 32 から 126 で“ : ”を除く）。この非標準のケースでは、<トランザクション・マネージャ・アドレス>と、グローバルに一意であることを保証する<トランザクション識別子>の組合せである。例を以下に示す。

tip://123.123.123.123/?transid1

トランザクション識別子から非標準 TIP URL を生成するために、最初にトランザクション識別子で予約された文字を対応するエスケープ・シーケンスに置き換えて、次

に適切なトランザクション・マネージャ・アドレスを書入れなさい。トランザクション識別子が、あなたが生成したものなら、あなたが所有するトランザクション・マネージャ・アドレスを書入れなさい。トランザクション識別子が、あなたが開始した TIP コネクション上で受信したものなら、IDENTIFY コマンドで送信されたセカンダリ・トランザクション・マネージャ・アドレスを使いなさい。トランザクション識別子が、あなたが開始していない TIP コネクション上で受信したものなら、IDENTIFY コマンドで受信されたプライマリ・トランザクション・マネージャ・アドレスを使いなさい。

TIP URL は常に唯一であることが全世界的に保証されなければならない。この唯一性は TIP URL が決して二重利用されず、従って起こり得る未確定の振る舞いを避けることを保証する。唯一性の実現方法は、特定の実装による。例えば、Universally Unique Identifier（Globally Unique Identifier または GUID（[9]を参照せよ）として知られる UUID は、<トランザクション文字列>の一部として使用することができた。また、いくつかの標準トランザクション識別子は、全世界で唯一であることを保証するためのルールを定義するかもしれないことに注意せよ（例えば、OSI CCR アトミック識別子）。
上記以外の方法を除くと、TIP URL スキーム（scheme）は、[4]で定義される予約文字のルールに従う。そして、[4]のセクション 5 で定義されるエスケープ・シーケンスを使用する。

TIP プロトコル自身が TIP URL スキームを使用しないことに注意せよ（トランザクション・マネージャ・アドレス・スキームを使用する）。TIP URL スキームは、トランザクション同一証明（identification）情報を、例えば協力関係にあるアプリケーション・プロセス間において、他プロトコルを通して渡すための標準的方法として提案されている。その時、特定の TIP トランザクション、例えばリモート・トランザクション・マネージャからトランザクションを PULL するための TIP トランザクションと、アプリケーションを関連付けるために必要な情報を、ローカル・トランザクション・マネージャと通信するために TIP URL が使用される。各 TIP 実装はこの目的のために、いくつかの API セットを提供することが予想されている（その API 例を含む[5]を参照せよ）。

9.コネクションの状態

いかなる瞬間にも、１コネクション上の１パーティだけがコマンド送信を許される。その間他のパーティは、受信するコマンドに応答することだけ許される。このドキュメントを通して、コマンドを送信することが許されるパーティを“プライマリ（primary）”と呼ぶ。他のパーティは“セカンダリ（secondary）”と呼ぶ。最初に、コネクションを始めたパーティがプライマリになる。しかしながら、いくつかのコマンドは切替の役割を担う。Idle 状態のときはいつでも、コネクションは元々の極性（polarity）に戻る。

（多重化状態である）土台となるコネクションの極性に関わらず、多重化が実施されて

いるとき、そのルールは、各“仮想”コネクションに独立して適用する。

パイプライン化によって、コマンドに対しセカンダリが“帯域外（out of band）”を送信する可能性があることに注意せよ。これはコネクションの極性に影響を与えない（すなわち、プライマリとセカンダリの役割は切替わらない）。詳細はセクション 12 を見よ。

正常なケースでは、TIP コネクションは（初期状態が）プライマリ側のみがクローズすべきである。一般的にセカンダリがクローズすることは望ましくないが、ある異常なケースでは必要となる。

いかなる瞬間でも、コネクションは次の状態の一つとなる。セカンダリ・パーティの観点では、状態は応答送信時に変わる。プライマリ・パーティの観点では、状態は応答受信時に変わる。

Initial： 最初のコネクションは Initial 状態で始まる。この状態へ入ると、コネクションを始めたパーティはプライマリになり、他のパーティはセカンダリになる。この状態のコネクションに関係付けられるトランザクションは存在しない。この状態から、プライマリは IDENTIFY または TLS コマンドを送信できる。

Idle： この状態では、プライマリとセカンダリはプロトコル・バージョンに同意し、プライマリは障害後に再接続するための識別子をセカンダリ・ パーティへ供給した。この状態でコネクションに関連付けられるトランザクションは存在しない。この状態へ入ると、コネクションを始めたパーティはプライマリになり、他のパーティはセカンダリになる。この状態から、プライマリは次のコマンドのいくつかを送信できる：BEGIN, MULTIPLEX, PUSH, PULL,QUERY, そして RECONNECT。

Begun： この状態では、コネクションはアクティブなトランザクションに関連付けられる。そのトランザクションは 1 フェーズ（one-phase）プロトコルによってのみ完了することが可能である。BEGIN コマンドに対する BEGUN レスポンスは、コネクションをこの状態にする。Begun 状態でのコネクション障害は、トランザクションをアボートする必要がある。この状態から、プライマリは ABORT または COMMIT コマンドを送信できる。

Enlisted： この状態では、コネクションはアクティブなトランザクションに関連付けられる。そのトランザクションは 1 フェーズ（one-phase）または 2 フェーズ（two-phase）プロトコルによって完了することができる。PUSH コマンドに対する PUSHED レスポンス、または PULL コマンドに対する PULLED レスポンスは、コネクションをこの状態にする。Enlisted 状態でのコネクション障害は、トランザクションをアボートする必要がある。この状態から、プライマリは ABORT,COMMIT または PREPARE コマンド

を送信できる。

Prepared： この状態では、コネクションは準備されたトランザクションに関連付けられる。PREPARE コマンドに対する PREPARED レスポンス、または RECONNECT コマンドに対する RECONNECTED レスポンスは、コネクションをこの状態にする。他の状態と違い、この状態でのコネクション障害はトランザクションを自動的にアボートしない。この状態から、プライマリは ABORT または COMMIT コマンドを送信できる。

Multiplexing：この状態では、コネクションは多重化プロトコルによって使用されている。多重化プロトコルは、コネクション・セットを提供する。この状態で、そのコネクション上で可能な TIP コマンドは無い。（もちろん、多重化プロトコルによって供給されるコネクション上では、TIP コマンドは可能である。）そのコネクションは決してこの状態を変えることができない。

Tls： この状態では、コネクションは TLS プロトコルによって使用されている。そのプロトコルは安全な（secured）コネクションを提供する。この状態では、そのコネクション上で可能な TIP コマンドは無い。（もちろん、TLS プロトコルによって供給されるコネクション上では、TIP コマンドは可能である。）そのコネクションは決してこの状態を変えることができない。

Error： この状態では、プロトコル・エラーが起きて、コネクションがもはや使用できない。そのコネクションは決してこの状態を変えることができない。

10.プロトコル版数設定

このドキュメントは第 3 版のプロトコルについて記述する。将来の版に適応させるために、プライマリ・パーティは理解する最小と最大版数を示すメッセージを送信する。セカンダリは理解する最大版数で応答する。

その交換の後に、最大版数（すなわち、両方が理解する最大版数）より小さな版数を使った通信を行うことができる。交換は必須で、IDENTIFY コマンド（そして IDETIFIED レスポンス）を使う。

１パーティによりサポートされる最大版数が時代遅れとみなされ、もはや他のパーティによりサポートされないなら、有用な通信を行うことができない。このケースでは、新しい方のパーティが単にコネクションを止める（drop）べきである。

11.コマンドとレスポンス

全てのコマンドとレスポンスは、値 32 から 126 の範囲のオクテットだけを使って、CR（値 13 のオクテット）または LR（値 10 のオクテット）のどちらかが後に続く、ASCII テキスト１行から構成される。各行は１つ以上の“単語（words）”に分けることができる。各単語は１つ以上のスペース・オクテット（値 32）により分けられる。

各行の始めと（または）終わりの自由な幅のスペースが許されていて、無視される。空の行、または全てスペースで構成される行は無視される。（これは、望むなら CR と LF 両方で行を終わらせることが可能で、LF が空の行を終わらせて、無視されることを示唆する。）

全てのケースで、各行の最初の単語は、コマンドまたはレスポンスのタイプを示す。全ての定義されたコマンドとレスポンスは大文字だけで構成される。

いくつかのコマンドとレスポンスに対し、後の単語がコマンドとレスポンスのパラメータを運ぶ。各コマンドとレスポンスは固定数のパラメータを持つ。

未定義の単語を含むコマンドまたはレスポンス行の全ての単語は全て無視される。デバックと他の目的のために、人が読める（human-readable）情報を 渡すために、それら単語を使用できる。

12.コマンド・パイプライン化

通信の呼び出し時間（latency）を減少し能力を改善するために、多重 TIP “ 行 ”（コマンドまたはレスポンス）がパイプライン化（連結化）され、単一のメッセージとして送信されることが可能である。また行が “ 先に ” 送信されてもよい（セカンダリにより、プライマリによる後の処理のために）。例えば、ABORT コマンドの直後に BEGIN コマンド、または COMMITTED レスポンスの直後に PULL コマンドが続いてよい。

パイプライン化された行の送信は、実装オプションである。行のパイプライン化も同様である。一般的に、受信側によって処理される時間に、パイプライン化された行が有効なコネクション状態であることを必ず確かめなければならない。これを確定するのは、送信側の責任である。

全ての実装はパイプライン化された行をサポートしなければならない。次のパラグラフに処理ルールを記述する。

（コマンドまたはレスポンス）行が読まれるべきコネクション状態のとき、受信した行が処理される。もう一度その状態になるまで、行はコネクションから読まれない。他のパーティの送信が許されない時間に行を受信してもその行は拒否（reject）されない。その代わり、行は保存され、その行を再要求するコネクション状態のときに処理される。受信側のパーティは行を処理しなければならず、受信した行の順番でレスポンスを発行しなければ ならない。コネクションがエラー状態に入るなら、そのコネクション上の エラー発生後の全ての行が廃棄される。

13.TIP コマンド

コマンドはコネクションかトランザクションのどちらかに関係する。コネクションに属するコマンドは、IDENTIFY, MULTIPLEX、そして TLS である。トランザクションに関

係するコマンドは、ABORT, BEGIN, COMMIT, PREPARE, PULL, PUSH, QUERY, そして RECONNECT である。
有効なコマンドと、コマンドに対して可能なレスポンスの全リストを次に示す。

ABORT

このコマンドは Begun、Enlisted、そして Prepared 状態で有効である。そのコネクションの現トランザクションがアボートすることをセカンダリへ知らせる。可能なレスポンスは次の通り。

ABORTED

トランザクションはアボートした。コネクションは Idle 状態に入る。

ERROR

コマンドが誤った状態で発行された、または奇形（malform）だった。コネクションは Error 状態に入る。

BEGIN

このコマンドは Idle 状態で有効である。セカンダリに新しいトランザクションの生成を依頼し、そのコネクションを関係付ける。新たに生成したトランザクションは１フェーズ（one-phase）で完了するだろう。可能なレスポンスは次の通り。

BEGUN <transaction identifier>

新しいトランザクションが首尾よく始まり、そのコネクションの今現在のトランザクションとなる。そのコネクションは Begun 状態に入る。

NOTBEGUN

新しいトランザクションを始めることができなかった。コネクションは Idle 状態のままとなる。

ERROR

コマンドが悪い状態で発行された、または奇形（malform）だった。コネクションはエラー状態に入る。

COMMIT

このコマンドは Begun、Enlisted、または Prepared 状態で有効である。Begun または Enlisted 状態で、トランザクションのコミットの試みをセカンダリへ依頼する。Prepare

状態で、トランザクションがコミットされたことをセカンダリへ知らせる。Enlisted 状態では、このコマンドは１フェーズプロトコルを表し、送信側が、1）そのトランザクションに含まれる回復可能なローカル資源を持たず、2）1 つのサブオーディネート（subordinate）のみ持つ場合だけ実行されなければならないことに注意せよ（送信側はいかなるトランザクション回復処理にも含まれない）。可能なレスポンスは次の通り。

ABORTED

このレスポンスは Begun と Enlisted 状態からのみ可能である。いくつかのパーティがトランザクションのコミットメントを拒否し、そのためにコミットの代わりにアボートしたことを示す。コネクションは Idle 状態に入る。

COMMITTED

このレスポンスは、トランザクションがコミットされ、プライマリがそのトランザクションに関するセカンダリに対し、もはや責任を持たないことを示す。コネクションは Idle 状態に入る。

ERROR

コマンドが悪い状態で発行された、または奇形（malform）だった。コネクションは Error 状態に入る。

ERROR

このコマンドはいかなる状態でも有効である。前のレスポンスが認められない、または異常な形式だったことをセカンダリへ知らせる。セカンダリはこのコマンドに対し応答すべきではない。コネクションは Error 状態に入る。

IDENTIFY　　<lowest protocol version>
　　<highest protocol version>
　　<primary transaction manager address> | "-"
　　<secondary transaction manager address>

このコマンドは Initial 状態でのみ有効である。プライマリ・パーティはセカンダリ・パーティへ次の情報を知らせる。1）サポートされる最小と最大プロトコル版数（最小と最大の間の全ての版数がサポートされなければならない）。2）オプションとして、必要とするならセカンダリがコネクションを再確立できる、プライマリ・パーティの識別子（プライマリ・トランザクション・マネージャ・アドレス）。3）必要な TIP トランザクション・マネージャに接続するため中間プロキシ・サーバにより使用される識別子（セカンダリ・トランザクション・マネージャ・アドレス）。プライマリ・トランザクション・マネージャ・

アドレスが供給されないなら、セカンダリ・パーティは、PREPARE コマンドに対し、ABORTED または READONLY と一緒に応答するだろう。可能なレスポンスは次の通り。

IDENTIFIED <protocol version>

セカンダリ・パーティは首尾よく連絡が取れて、プライマリ・トランザクション・マネージャ・アドレスを保存した。レスポンスはセカンダリ・パーティによりサポートされる最大プロトコル版数を含む。将来の全通信において、IDENTIFY コマンドと IDENTIFIED レスポンスのプロトコル版数より小さい番号を使用することが仮定されている。コネクションは Idle 状態に入る。

NEEDTLS

セカンダリ・パーティは、TLS セキュア（secured）コネクションを通して自発的に通信するだけである。コネクションは Tls 状態に入り、その後の全通信が TLS プロトコルにより定義される。

このプロトコルは、（プライマリ・パーティにより送信されるデータのために）IDENTIFY コマンド行末に続く最初のオクテットと、（セカンダリ・パーティにより送信されるデータのために）NEEDTLS レスポンス行末に続く最初のバイトで始まる。これは、LF オクテットが TLS プロトコルの最初のバイトであると誤らないように、IDENTIFY コマンドまたは NEEDTLS レスポンスのどちらかの後に、CR と LF 両方を送信してはいけないことを含んでいる。TLS プロトコルにより提供されるコネクションは Intial 状態で開始する。TLS が折衝された後、プライマリ・パーティは IDENTIFY コマンドを再送しなければならない。もしプライマリ・パーティが TLS プロトコルをサポートできない（または使用を拒否する）なら、コネクションをクローズする。

ERROR

コマンドは悪い状態で発行された、または奇形（malform）だった。このレスポンスはまた、プライマリ・パーティによりサポートされる範囲のいかなるプロトコル版数もセカンダリ・パーティがサポートしない場合にも発行される。そのコネクションはエラー状態に入る。プライマリ・パーティはコネクションをクローズすべきである。

MULTIPLEX <protocol-identifier>

このコマンドは Idle 状態でのみ有効である。コマンドは多重化プロトコルをそのコネクションで使用することに同意することを要求する。多重化プロトコルは存在するコネクション上で多数のコネクションを供給する。 プライマリは特定の多重化プロトコルを提案する。セカンダリ・パーティはこのプロトコルの使用を受け入れるか拒否することができる。

現在のところ、定義済みのプロトコル識別子は、TIP 多重化プロトコル第２版を意味する“ TMP2.0 ”のみである。このプロトコルの詳細は Appendix A を見よ。他のプロトコル識別子は将来定義されるかもしれない。

MULTIPLEX コマンドが受け入れられるなら、指定された多重化プロトコルは、土台となるコネクションを全て制御する。このプロトコルは（開始側により送信されるデータが）MULTIPLEX コマンドの行末に続く最初のオクテットと、（開始がわにより受信されるデータが）MULTIPLEXING レスポンスの行末に続く最初のバイトで始まる。これは多重化プロトコルの最初のバイトが LF オクテットであると誤らないように、MULTIPLEX コマンドか MULTIPLEXING レスポンスのどちらかの後に続いて、CR と LF を両方とも送信してはいけないことを含む。

TMP V2.0 使用する時、単一の TIP コマンド（TMP アプリケーション・メッセージ）が単一の TMP パケットに全て含まれなければならない（TMP PUSH フラグは TIP により使用されない）ことに注意せよ。MULTIPLEX コマンドに対して可能なレスポンスは次の通りである。

MULTIPLEXING

セカンダリ・パーティは指定される多重化プロトコルを使用することに同意する。コネクションは Multiplexing 状態に入り、その後の通信は全てこのプロトコルにより定義される。多重化プロトコルにより生成されたコネクションは全て Idle 状態で開始する。

CANTMULTIPLEX

セカンダリ・パーティは、指定された多重化プロトコルをサポートできない（または使用を拒否する）。コネクションは Idle 状態のままである。

ERROR

コマンドは悪い状態で発行された、または奇形（malform）だった。コネクションは Error 状態に入る。

PREPARE

このコマンドは Enlisted 状態でのみ有効である。コミット（２フェーズのフェーズ１コミット）に対してトランザクションを準備することをセカンダリへ要求する。可能なレスポンスは次の通りである。

PREPARED

サブオーディオネートはトランザクションを準備した。コネクションは Prepared 状態に入る。

ABORTED

サブオーディオネートはトランザクションをコミットすることを拒否した。コネクションは Idle 状態に入る。このレスポンスの後に、スーペリアはトランザクションに関してサブオーディオネートに対する責任を持たない。

READONLY

サブオーディネートは、トランザクションがコミットとアボートのどちらでも、もはや構わない。コネクションは Idle 状態に入る。このレスポンスの後に、スーペリアはトランザクションに関するサブオーディネートにのはや責任を持たない。

ERROR

コマンドは悪い状態で発行された、または奇形（malform）だった。コネクションは Error 状態に入る。

PULL　<superior's transaction identifier> <subordinate's transaction identifier>

このコマンドは Idle 状態でのみ有効である。このコマンドは、サブオーディネートとしてコネクションのプライマリ・パーティと一緒に、トランザクションでスーペリア/サブオーディネート関係を確立することを要求する（すなわち、セカンダリ・パーティからトランザクションを PULL する）。

<トランザクション文字列>（“ TIP URL ”セクションで定義される）の全ての値がトランザクション識別子として指定されなければならないことに注意せよ。可能なレスポンスは次の通り。

PULLED

関係が確立された。このレスポンスを受信すると、指定されたトランザクションはそのコネクションの現在のトランザクションになり、そのコネクションは Enlisted 状態に入る。さらに、プライマリとセカンダリの役割は予約される。（すなわち、スーペリアはそのコネクションのプライマリになる。）

NOTPULLED

関係は確立されなかった（セカンダリ・パーティが、もはや要求されたトランザクションを持たないためかもしれない）。コネクションは Idle 状態のままとなる。

ERROR

コマンドは悪い状態で発行された、または奇形（malform）だった。そのコネクションは Error 状態に入る。

PUSH <superior's transaction identifier>

このコマンドは Idle 状態でのみ有効である。スーペリアとしてプライマリと一緒にトランザクションのスーペリア/サブオーディネート関係を確立することを要求する。<トランザクション文字列>（“ TIP URL ”セクションで定義されている）の全ての値は、トランザクション識別子として指定されなければならないことに注意せよ。可能なレスポンスは次の通りである。

PUSHED <subordinate's transaction identifier>

関係は確立され、サブオーディネートがトランザクションを知る識別子が返された。トランザクションはそのコネクションの現在のトランザクションになり、そのコネクションは Enlisted 状態に入る。

ALREADYPUSHED <subordinate's transaction identifier>

関係が確立され、サブオーディネートがトランザクションを知る識別子が返された。しかし、サブオーディネートは既にそのトランザクションについて知っていて、違うコネクションを通して受信する２フェーズ・コミット・プロトコルを期待している。このケースでは、そのコネクションは Idle 状態のままである。

NOTPUSHED

関係は確立されることができなかった。コネクションは Idle 状態のままである。

ERROR

コマンドは悪い状態で発行された、または奇形（malform）だった。そのコネクションは Error 状態に入る。

QUERY <superior's transaction identifier>

このコマンドは Idle 状態でのみ有効である。サブオーディネートは、特定のトランザクションがスーペリアにまだ存在するかどうか確定するために、このコマンドを使用する。可能なレスポンスは次の通りである。

QUERIEDEXISTS

トランザクションは、まだ存在する。そのコネクションは Idle 状態のままとなる。

QUERIEDNOTFOUND

トランザクションは、もはや存在しない。そのコネクションは Idle 状態のままとなる。

ERROR

コマンドは悪い状態で発行された、または奇形（malform）だった。そのコネクションは Error 状態に入る。

RECONNECT <subordinate's transaction identifier>

このコマンドは Idle 状態でのみ有効である。スーペリアは、前のコネクションが Prepare 状態の間にロストした場合、トランザクションのためのコネクションを再確立するために、このコマンドを使用する。可能なレスポンスは次の通り。

RECONNECTED

サブオーディネートは再接続を受け入れる。そのコネクションは Prepared 状態に入る。

NOTRECONNECTED

サブオーディネートは、もはやトランザクションについて知らない。そのコネクションは Idle 状態のままとなる。

ERROR

コマンドは悪い状態で発行された、または奇形（malform）だった。そのコネクションは Error 状態に入る。

TLS

このコマンドは Initial 状態でのみ有効である。プライマリはこのコマンドを、TLS を使った安全なコネクション確立を試みるために使用する。

もし TLS コマンドが受け入れられるなら、TLS プロトコルがその後のコネクションを全て制御する。このプロトコルは、（プライマリにより送信されるデータが）TLS コマンド行末の後に続く最初のオクテットと、（プライマリが受信するデータが）TLSING レスポンス行末の後に続く最初のバイトで始まる。これは実装上、LF オクテットが TLS プロトコルの最初のバイトと誤らないように、TLS コマンドか TLSING レスポンスの後に、CR と LF の両方を送信できないことを含む。

TLS コマンドに対する可能なレスポンスは次の通りである。

TLSING

セカンダリ・パーティは TLS プロトコル（[3]を参照せよ）を使用することを受け入れる。そのコネクションは Tls 状態に入り、後に続く全ての通信が TLS プロトコルにより定義されている。TLS プロトコルにより提供されるそのコネクションは、Initail

状態で開始する。

CANTTLS

セカンダリ・パーティは TLS プロトコルをサポートできない（または使用を拒否する）。そのコネクションは Initail 状態のままである。

ERROR

コマンドが悪い状態で発行された、または奇形（malform）だった。そのコネクションは Error 状態に入る。

14.エラー処理

どちらのパーティが理解できない行を受信したとしても、そのパーティはコネクションをクローズする。どちらかのパーティが ERROR 指示または ERROR レスポンスを受信したとしても、そのコネクションは Error 状態に入り、それ以上の通信がそのコネクションで不可能となる。ある実装はコネクションをクローズすることを決めるかもしれない。コネクションのクローズは、他のパーティに通信障害としてみなされる。

ERROR 指示または ERROR レスポンスの受信は、自パーティを不適切な実装であること他のパーティが示している。

15.コネクション障害とリカバリー

コネクション障害は、通信障害またはコネクションをクローズするパーティにより引き起こされる。TIP 実装が TIP コネクション障害を見つけ出すプライベート・メカニズムを使用することが仮定されている（例えば、ソケット・キープアライブまたはタイムアウト・スキーム）。

コネクションの状態に従って、コネクション障害時に、トランザクション・マネージャは多用なアクションをとることが必要である。

Initial または Idle 状態でコネクション障害が発生するなら、そのコネクションはトランザクションに関係しない。アクションは不要である。

Multiplexing 状態でコネクション障害が発生するなら、多重化プロトコルにより提供される全てのコネクションに障害が発生したと仮定される。それらの各々は他と無関係で扱われる。

コネクション障害が Begun または Enlisted 状態で発生して COMMIT が送信されたなら、そのトランザクション完了処理（completion）はサブオーディネートに委任され（スーペリアは含まれない）、トランザクションの結果をスーペリアは知らされない（サブオーディネートは知る）。スーペリアは、トランザクションの結果を確定するために、アプリケーションに特有な手段を使用する（スーペリアがトランザクションに含まれる回復可能な

資源を持たないため、このケースではトランザクションの完全性（integrity）は妥協されないことに注意せよ）。コネクション障害が Begun または Enlisted 状態で発生して COMMIT が送信されていないなら、トランザクションはアボートされる。

コネクション障害が Prepared 状態で発生する場合、適切なアクションはそのトランザクションのスーペリア側とサブオーディネート側で異なる。

スーペリアがそのトランザクションのコミットを決定するなら、最後にサブオーディネートへ新しいコネクションを確立して、そのトランザクションに対し RECONNECT コマンドを送信しなければならない。NOTRECONNECTED レスポンスを受信するなら、それ以上何もする必要がない。しかし、RECONNECTED レスポンスを受信するなら、COMMIT 要求を送信して COMMITTED レスポンスを受信しなければならない。

スーペリアがそのトランザクションのアボートを決定するなら、新しいコネクションを確立して、そのトランザクションに対し RECONNECT コマンドを送信することが許されている（しかし必要とされているわけではない）。RECONNECTED レスポンスを受信した場合、ABORT コマンドを送信するべきである。

上記定義は、便利さを認めるなら、トランザクションの結果を知る前に、コネクションを再確立することをスーペリアに許している。RECONNECT コマンドに成功したなら、コネクションは Prepared 状態に戻り、スーペリアはトランザクションの結果を知る時、適切なコマンドとして COMMIT または ABORT を送信できる。

その前のコネクション障害に気づく前に、RECONNECT コマンドをサブオーディネートが受信可能なことに注意せよ。

このケースではサブオーディネートは RECONNECT コマンドを、障害通知、そのコネクションに関係付けられる全資源の解放、そして新しいコネクションをそのトランザクション状態に関連付けるコマンドとして扱う。

サブオーディネートが Prepared 状態でコネクション障害に気づいたなら、定期的にスーペリアへ新しいコネクション生成を試みて、そのトランザクションに対する QUERY コマンドを送信すべきである。次の２イベントのうち１つが起きるまで、実行し続けるべきである。

1．スーペリアから QUERIEDNOTFOUND レスポンスを受信する。このケースでは、サブオーディネートはトランザクションをアボートすべきである。
2．スーペリアは、開始したいくつかのコネクション上で、サブオーディネートに対しそのトランザクションのための RECONNECT コマンドを送信する。このケースでは、サブオーディネートが新しいコネクション上でそのトランザクションの結果を知ることが期待できる。サブオーディネートがそのトランザクションの結果を知る前に、その新しいコネクションで障害が発生した場合、再度 QUERY コマンドを送信し始めるべきである。

TIP システムが QUERY または RECONNECT コマンドのどちらかを受信して、いくつかの理由により要求を満たすことができないなら（例えば、必要なリカバリ情報が現在利用できない場合）、コネクションはドロップされるべきである。

16.セキュリティの考察

このセクションはアプリケーション開発者、トランザクション・マネージャ開発者、そしてこのドキュメントに記述されている TIP のセキュリティ関連ユーザに伝えることを意図している。記述される問題点に対し、セキュリティ・リスクを減少するためにいくつか提案しているが、完全なソリューションは含まれていない。

全ての２フェーズ・コミット・プロトコルと同様、関連するメカニズムがコミットメント・プロトコルに適用されないなら、アプリケーション通信プロトコルに適用されるいかなるセキュリティ・メカニズムも覆されやすい。例えば、アプリケーション・プロトコルを使うパーティ間のいかなる証明も、少なくとも同じレベルの確実性を持つ TIP 交換のセキュリティによりサポートされなければならない。

16.1 TLS, 相互証明と権限付与

TLS は、任意のクライアント側証明、任意のサーバ側証明、そして任意のパケット暗号化を提供する。

TIP 実装は、もし TLS が使用されないなら、サービス提供を拒否するかもしれない。もしパケット暗号化が使用されないなら、サービス提供を拒否するかもしれない。もしリモート・パーティが（TLS を通して）証明されないなら、サービス提供を拒否するかもしれない。

TIP 実装は（TLS を通して）自分自身が証明されることを自発的にすべきである。これは実装がクライアントまたはサーバのどちらであるかに関わらず真実である。

一度リモート・パーティが証明されると、TIP トランザクション・マネージャは、何のオペレーションを許すべきか決めるためにリモート・パーティの ID（identity）を使用してよい。

TLS がコネクション上で使用されるかどうか、使用されるなら、どのように TLS が使用されるか、そして何のオペレーションがその後に許されるかについては、お互いに接続している TIP トランザクション・マネージャのセキュリティ・ポリシーにより決定される。そのセキュリティ・ポリシーがどのように定義されるか、そしてどのようにトランザクション・マネージャがそれを知るかについては、全て実装に任されていて、このドキュメントの範囲外である。

16.2 PULL に基づくサービス拒絶アタック

悪意のあるユーザが、いくつかのトランザクション・マネージャにおいて現在アクティ

ブであるトランザクション ID を知っていると仮定せよ。悪人がトランザクション・マネージャへ TIP コネクションをオープンして、PULL コマンドを送信し、そのコネクションをクローズするなら、その悪人はそのトランザクションのアボートを引き起こすことができる。これは結果として、そのトランザクションの正式なオーナーに対するサービス拒絶になる。

ある実装は、TLS が使用されず、リモート・パーティが証明されず、リモート・パーティが信頼できない場合は、PULL コマンドを拒否することによりこのアタックを避けるかもしれない。

16.3 PUSH に基づくサービス拒絶アタック

2 つのトランザクション･マネージャ間のコネクションが､トランザクションが Prepared 状態でクローズされるとき、各トランザクション・マネージャはコネクションが再確立されるまで、そのトランザクションについての情報を覚えておく必要がある。

悪意のあるユーザが､トランザクションを繰り返し生成するこの事実を活用し､Prepared 状態でトランザクションに入り、そのコネクションをドロップするなら、そのユーザはトランザクション・マネージャに資源枯渇の被害を与えるかもしれない。従って、トランザクション・マネージャの全ての正式なユーザに対するサービスを拒否することになる。

ある実装は、もし TLS が使用されず、リモート・パーティが証明されず、リモート・パーティが信頼されないなら、PUSH コマンドを拒否することによりこのアタックを避けるかもしれない。

16.4 トランザクション改ざんアタック

サブオーディネート・トランザクション・マネージャが特定の準備されたトランザクションのためのコネクションを失ったなら、悪意のあるユーザがトランザクション・マネージャへ TIP コネクションを開始して、そのトランザクションのための COMMIT または ABOR t コマンドが後に続く RECONNECT コマンドを送信できる。悪意のあるユーザは従ってアボートされるべきときに、トランザクションの一部をコミットすることができる。

ある実装は、トランザクションのスーペリアの証明された ID を記録して、もし TLS が使用されず、リモート・パーティの証明された ID がスーペリアのオリジナル ID と同じでなければ､RECONNECT コマンドを拒否することにより､このアタックを避けるかもしれない。

16.5 パケット盗聴アタック

悪意のあるユーザが TIP コネクション上のトラフィックを傍受することができるなら、他のアタックを計画するときにその情報を役立てることができるかもしれない。例えば、コメント・フィールドがトランザクション・マネージャの製品名とバージョン番号を含む

なら、悪意のあるユーザはこの情報をその実装に何のセキュリティバグが存在するか決定するために使用できるかもしれない。

ある実装は暗号化されたセッションを提供するために TLS を常に使用して、TLS/TLSING コマンド/レスポンス・ペア上に個人情報を置かないことにより、このアタックを避けるかもしれない。

16.6 中間アタック

悪意のあるユーザが傍受して TIP コネクション上のトラフィックを変えることができるなら、そのユーザはいくつかの方法で大混乱を加えることができる。例えば、そのユーザはコミットコマンドと ABORT コマンドを置き換えることができる。

ある実装は、暗号化セッションを提供してリモート・パーティを証明するために、常に TLS を使用して、このアタックを避けるかもしれない。

17.参照

[1]Gray, J. and A. Reuter（1993）, Transaction Processing: Concepts and Techniques. San Francisco, CA: Morgan Kaufmann Publishers.（ISBN 1-55860-190-2）.

[2]Fielding, R., Gettys, J., Mogul, J., Frystyk, H., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC2068, January 1997.

[3]Dierks, T., et. al., "The TLS Protocol Version 1.0", Work in Progress.

[4]Berners-Lee, T., Masinter, L., and M. McCahill, "Uniform Resource Locators（URL）", RFC 1738, December 1994.

[5]Evans, K., Klein, J., and J. Lyon, "Transaction Internet Protocol - Requirements and Supplemental Information", RFC 2372, July 1998.

[6]Moats, R., "URN Syntax", RFC 2141, May 1997.

[7]Faltstrom, P., et. al., "Namespace Identifier Requirements for URN Services", Work in Progress.

[8]Berners-Lee, T., et. at., "Uniform Resource Identifiers（URI）: Generic Syntax and Semantics", Work in Progress.

[9]Leach, P., and R. Salz, "UUIDs and GUIDs", Work in Progress.

18.著者のアドレス

Jim Lyon

Microsoft Corporation

One Microsoft Way

Redmond, WA　98052-6399, USA

Phone: +1（206） 936 0867

Fax:　+1（206） 936 7329
EMail: JimLyon@Microsoft.Com

Keith Evans
Tandem Computers, Inc.
5425 Stevens Creek Blvd
Santa Clara, CA 95051-7200, USA
Phone: +1（408） 285 5314
Fax:　+1（408） 285 5245
EMail: Keith.Evans@Tandem.Com

Johannes Klein
Tandem Computers Inc.
10555 Ridgeview Court
Cupertino, CA 95014-0789, USA
Phone: +1（408） 285 0453
Fax:　+1（408） 285 9818
EMail: Johannes.Klein@Tandem.Com

19.コメント

次の著者、または TIP メーリングリスト<Tip@Lists.Tandem.Com>へ、このドキュメントのコメントを送信して下さい。
<JimLyon@Microsoft.Com>、<Keith.Evans@Tandem.Com>、
<Johannes.Klein@Tandem.Com>

あなたは、メッセージ・ボディのどこかに“ subscribe tip <フルネーム> ”行を入れて<Listserv@Lists.Tandem.Com>へメール送信することにより TIP メーリングリストへ登録できる。

付録 A. TIP 多重プロトコル第２版

この付録は TIP 多重プロトコル（TMP）の第 2 版について記述する。TMP は専ら TIP プロトコルと一緒に使用することが意図されていて、TIP プロトコル使用の一部を構成する（その実装は任意であるけれども）。TMP V2.0 は TIP V3.0 によりサポートされる唯一の多重化プロトコルである。

要約

TMP は、TCP コネクション上で多数の軽いコネクションを生成する簡単なメカニズムを

提供する。いくつかのそのような軽いコネクションは同時にアクティブになること可能である。TMP はバイト志向のサービスを提供するが、マークされるメッセージ境界を許す。

A.1. はじめに

各トランザクションのために１つの TCP コネクションを生成するいくつかのプロトコルが、インターネット上で広く使用されている。不運にも、それらのトランザクションは短い間生きているので、使用される資源と、TCP 輻輳制御メカニズムに関連づけれられる遅延の見地から、設定コストとその TCP コネクションの破壊が重要になる。

TIP 多重化プロトコル（TMP）は、一つのトランスポートコネクション上に多数の軽いコネクションを生成する、TCP の上で実行される簡単なプロトコルである。TMP は従って、TCP コネクションのもっと効率的な使用方法を提供する。いくつかの異なる TMP コネクションからのデータはインターリーブされることが可能で、メッセージ境界とストリーム目印の終わりが両方とも提供される。

TMP は順番が信頼できるトランスポート・サービスの上で実行されるので、TCP が信頼性を保証するために通らなければならない余分な作業のほとんどを避けることができる。例えば、TMP コネクションは確認（comfirm）される必要がない。だから、データが送信される前にハンドシェーク完了を待つ必要がない。

TMP は一般化された多重化プロトコルとしては意図されていない。多重化が必要な別のプロトコルをデザインしたいなら、TMP が適切なプロトコルになるかもしれないし、ならないかもしれない。大きなメッセージを伴うプロトコルは、受信側のバッファリング能力を超えて、ある条件でデッドロックを引き起こすことがありうる。TIP と一緒に使われるとき、TIP は要求応答プロトコルで全てのメッセージが短いので、TMP はこの問題を経験しない。

A.2. プロトコル・モデル

基本プロトコル・モデルは、信頼できるバイト・ストリーム上で動作する多数の軽いコネクションである。コネクションを開始したパーティは、プライマリとして参照され、コネクションを受け入れたパーティは、セカンダリとして参照される。

コネクションは、単向性または双方向性であるかもしれない。双方向性コネクションの各端は別々にクローズされるかもしれない。コネクションは普通にクローズされるか、異常な版であることを示すためにリセットされる。コネクションをアボートすることは、両方のデータ・ストリームをクローズする。

一度コネクションがオープンされたなら、アプリケーションはその上にメッセージを送信し、アプリケーション・レベル・メッセージの終わりを知らせることができる。アプリケーション・メッセージは、TMP パケットにカプセリングされ、バイト・ストリームを通して転送される。一つの TIP コマンド（TMP アプリケーション・メッセージ）は、一つの

TMP パケット内部に全て含まれなければならない。

A.3. TMP パケット・フォーマット

TMP パケットは0以上のデータが後に続く 64 ビット・ヘッダから構成される。ヘッダは3つのフィールドを含む。フラグ・バイト、コネクション識別子、そしてパケット長である。コネクション識別子とパケット長は両方とも整数で、ネットワーク・バイト順で送信されなければならない。

FLAGS

SFPR0000	Connection ID
	Length

A.3.1. フラグの詳細

Name	Mask	Description
SYN	1xxx|0000	Open a new connection
FIN	x1xx|0000	Close an existing connection
PUSH	xx1x|0000	Mark application level message boundary
RESET	xxx1|0000	Abort the connection

A.4. コネクション識別子

各 TMP コネクションは、24 ビット整数により識別される。下位の TCP コネクションを開始したパーティにより生成された TMP コネクションは、偶数の識別子を持たなければならない。他のパーティにより生成された TMP コネクションは奇数の識別子を持たなければならない。

A.5. TMP コネクション状態

TMP コネクションはいくつかの異なる状態で存在できる。Closed, Open Write, Open SynRead, Open SynReset, Open Read Write, Close Write、そして CloseRead。コネクションは、SYN, FIN または RESET ビットがセットされたパケット受信に応答してその状態を変えることができる。利用できる API 呼出は open、close、そして abort である。

ほとんどの状態の意味は明らかである（例えば、OpenWrite は、コネクションが書き込むために開かれたことを意味する）。状態 OpenSynRead と OpenResetRead の意味は追加説明が必要である。

OpenSynRead 状態では、プライマリが開いて、直ぐにコネクションの出力データ・ストリームを閉じて、読み出すための入力データ・ストリームを開くために、現在セカンダリ

から SYN 応答を待っている。

OpenResetRead 状態では、プライマリが開いて、直ぐにコネクションをアボートして、コネクションを最後に閉じるために、現在セカンダリから SYN 応答を待っている。

A.6. イベント優先度と状態移行

以降に示す状態テーブルは、アクションと、与えられるイベントの応答で起きる状態移行を記述する。各状態により受け入れられるイベントは、最も高い優先度を最初の順番として優先順に一覧を示している。複数のイベントがメッセージ中に表れるので、それらの一覧に適合しているイベントが処理される。多数のイベントが適合するなら、最も優先度の高いイベントが受け入れられ、最初に処理される。いかなる残りのイベントも結果として後に続く状態で処理される。

例えば、セカンダリの TMP コネクションが Closed 状態で、セカンダリが SYN イベント、FIN イベント、そして入力データ・イベント（すなわち DATA-IN）を含むパケットを受信したなら、セカンダリは最初に SYN イベントを受け入れる（Closed 状態で唯一適合するので）。セカンダリはコネクションを受け入れ、SYN イベントを送信し、ReadWrite 状態に入る。SYN イベントは延期イベントの一覧から削除される。残るイベントは FIN と DATA-IN である。ReadWrite 状態では、セカンダリは入力データを読む（FIN イベントより優先度が高いので、DATA-IN イベントが処理される）。一度データが読まれたなら、DATA-IN イベントは延期イベントの一覧から削除され、FIN イベントが処理され、セカンダリは CloseWrite 状態に入る。

セカンダリが SYN イベントを含むパケットを受信して、いくつかの理由でそのコネクションを受け入れることができないなら（例、資源が不充分）、RESET イベントが後に続く SYN イベントを送信することで要求を拒否すべきである。両方のイベントが同じ TMP パケットの一部として送信できることに注意せよ。

どちらかのパーティが、理解できない TMP パケットや、不適切な状態でイベントを受信したなら、そのパーティは TCP コネクションを閉じる。

Entry State	Event	Action	Exit State
Closed	SYN	SYN	ReadWrite
	OPEN	SYN	OpenWrite
OpenWrite	SYN	Accept	ReadWrite
	WRITE	DATA-OUT	OpenWrite
	CLOSE	FIN	OpenSynRead
	ABORT	RESEt	OpenSynReset
OpenSynRead	SYN	Accept	CloseRead
OpenSynReset	SYN	Accept	Closed
ReadWrite	DATA-IN	Accept	ReadWrite
	FIN	Accept	CloseWrite
	RESET	Accept	Closed
	WRITE	DATA-OUT	ReadWrite
	CLOSE	FIN	CloseRead
	ABORT	RESET	Closed
CloseWrite	RESET	Accept	Closed
	WRITE	DATA-OUT	CloseWrite
	CLOSE	FIN	Closed
	ABORT	RESET	Closed
CloseRead	DATA-IN	Accept	CloseRead
	FIN	Accept	Closed
	RESET	Accept	Closed
	ABORT	RESEt	Closed

TMP イベント優先度と状態移行

著作権表示全文

TASK FORCE DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.