=== ch09.docx

[chap]9章　ネットワークの遅延と戦う

ネットワーク管理者は、日々ネットワークの遅延と戦わなくてはいけません。しかしながら、ネットワークの遅延は恥ずべき問題というわけではありません。

ネットワークの遅延に取り組む前に、本当にネットワークが遅いのかということを確認する必要があります。この章ではその確認する方法を紹介しましょう。

まずはTCPのエラーリカバリとフロー制御機能について説明します。次にネットワーク遅延の原因を検出する方法を探ります。最後にネットワークと、ネットワーク上で動作する機器およびサービスのベースライン設定を見ていきます。この章が終わる頃には、ネットワーク遅延を識別、診断し、トラブルシューティングできる力がついているはずです。

注　ネットワーク遅延の解決に利用できるテクニックは複数あります。この章ではもっとも扱うことの多いTCPに的を絞りました。TCPを使えば、余分なトラフィックを増やさずに（ICMPのように）解析を行うことができます。

TCPのエラーリカバリ機能

TCPのエラーリカバリ機能は、ネットワークの高遅延の位置を見つけ、診断、修復してくれる、最良のツールです。コンピュータネットワーキング用語では、遅延（レイテンシ）とはパケットが送信され、受信されるまでの時間のことです。

遅延は一方向（送信元から宛先まで）あるいは往復（送信元から宛先まで、そして宛先から送信元まで）の両方で測定されます。通信機器間での通信速度が速く、パケットがある地点から別の地点へ届くまでの時間が短ければ、低遅延ということになります。反対にパケットが届くまでに相当な時間がかかる場合、高遅延とされます。高遅延はネットワーク管理者の最大の敵です。

6章では、TCPがシーケンスとACK番号を使って、パケットをどのように確実に転送するかを説明しました。本章では、再度シーケンスとACK番号を見て、高遅延によってこれらの番号を受信する順番が乱れた場合（あるいはまったく受信しない場合）、TCPがどう応答するかを見ていきます。

TCP再送

パケットの再送は、TCPのもっとも基本的なエラーリカバリ機能のひとつです。この機能はパケット消失を防ぐのを目的としています。

パケット消失には、アプリケーションの異常、ルータのトラフィック過負荷、あるいは一時的なサービス障害を含む、さまざまな原因が考えられます。パケットレベルでの時間の経過は早く、またパケット消失は一時的である場合が多いため、TCPが迅速に検知し、消失からリカバーするのが重要となります。

パケット再送が必要かどうかを決める主なメカニズムは、再送タイマーと呼ばれています。タイマーはRTO（再送タイムアウト）の値を守るためのものです。TCPを使ってパケットが送信されると再送タイマーがスタートし、そのパケットのACKが受信されると止まります。パケットが送信され、ACKパケットが受信されるまでの時間をRTT（ラウンドトリップタイム）と呼びます。これらの時間の平均値が取られ、最終的なRTO値が算出されます。

RTO値が決まるまで、送信はデフォルトのRTT設定に基づいて行われます。この設定はホスト間の最初の通信のために設定されたもので、受信したパケットのRTTをもとに、実際のRTOが導き出されます。

RTO値が決まると、再送タイマーを使えばパケット消失が起きたかどうかが分かります。図9-1はTCPの再送プロセスを図式化したものです。

図9-1　TCP再送プロセスの概念図

送信元ホスト

受信ホスト

データ

再送1回目（RTO）

再送2回目（RTO×2）

再送3回目（RTO×4）

再送4回目（RTO×8）

再送5回目（RTO×16）

応答なし＝接続終了

パケットが送られても、受信者がTCP ACKパケットを送信しないと、送信元のホストは送ったパケットが消失したと判断し、パケットを再送します。再送が行われると、RTO値は倍になります。その値に達する前にACKパケットが届かないと、再再送されます。そしてまたもACKが届かない場合、値はさらに倍になります。ACKパケットが受信されるまで、または最初に設定した再送回数の最高値に達するまで、このプロセスは繰り返され、RTO値は再送するたびに倍増していきます。

再送回数の最高値は設定によります。Windowsホストの場合のデフォルトの最高再送回数は5回、Linuxの場合は15回です。

TCP再送の例を見るため、ファイルtcp\_ retransmissions.pcapを開くと、パケットが6個入っています。図9-2に示したのが最初のパケットです。

図9-2　データを含んだTCPパケット

これは648バイトのデータ②を含むTCP PSH/ACKパケット①で、10.3.30.1から10.3.71.7へ送られたものです③。典型的なデータパケットです。

通常の状況であれば、最初のパケットが送信されるとすぐ、TCP ACKパケットが応答として送られてくるはずです。しかしここでは、次のパケットが再送されているのが、パケット一覧のペインを見ると分かります。Infoカラムにははっきりと[TCP Retransmission]とあり、パケットは黒字に赤いテキストで表示されています。図9-3はパケット一覧のペインの再送リストの例です。

図9-3　パケット一覧のペインの再送リスト

図9-4のように、パケット詳細またはバイナリのペインでも、パケットが再送されたかどうかがわかります。

図9-4　再送パケット

このパケットは最初のパケットと同じものです（IPのIDと[Checksum]フィールドを除けば）。2つのパケットのバイナリのペインを比較すれば、同一であると検証できます①。

パケット詳細のペインを見ると、再送パケットのSEQ/ACK Analysisヘディングの下に、追加情報があるのに気づくでしょう。これはWiresharkが提供している情報で、パケットそのものには含まれていません。SEQ/ACK解析から、これが確かに再送で③、RTO値は0.206秒④、そしてRTO値は1番目のパケットのデルタタイムをもとにしていると確認できます。

残りのパケットでも同様の結果が出ますが、IPのIDと[Checksum]フィールド、またRTO値だけは違います。各パケットの時間差を目で確認するには、パケット一覧のペインでTimeカラムを見ます（図9-5）。再送でRTO値が倍増するごとに、時間が大幅に延びているのがわかります。

TCPの再送機能は、送信者がパケット消失を検出し、そこからリカバーするのに利用できます。今度は受信者がパケットの消失を検出、リカバーするのに使える重複ACK（duplicate acknowledgments）機能を見てみましょう。

図9-5　RTO値の増加を示すTimeカラム

重複ACKと高速再送

受信者が順番の狂ったパケットを受け取ったときに送るTCPパケットが、重複ACKです。TCPは、データが同じ順番で送信され、再度組み立てられるよう、シーケンス番号とヘッダ内のACK番号フィールドを使っています。

注　TCPパケットは厳密にいえばTCPセグメントですが、一般にはパケットと呼ばれています。

新たなTCP接続が確立されたとき、ハンドシェイクプロセスで交換されるもっとも重要な情報のひとつが、イニシャルシーケンス番号（ISN）です。ISNが接続の両側でセットされると、パケットが送信されるごとに、そのパケットのデータ容量分だけシーケンス番号が増えていきます。

あるホストのISNが5000で、500バイトのパケットを送信するとします。このパケットが受信されると、受信側はACK番号5500のTCP ACKパケットで返信します。

シーケンス番号＋受信されたデータのバイト容量＝ACK番号

送信元へ返されるACK番号は、受信者が次に受け取るシーケンス番号だということになります。これを図式化したのが図9-6です。

図9-6　TCPシーケンス番号とACK番号

送信元ホスト　受信するホスト

シーケンス番号5000　　データ量500

ACK番号5500

シーケンス番号5500　データ量500

ACK番号6000

シーケンス番号6000　データ量500

ACK番号6500

受信者はシーケンス番号を見れば、パケット消失が検出できます。シーケンス番号を追跡していれば、番号が狂っていないかどうか確認できるのです。

予想以外のシーケンス番号を受信した場合、パケットが消失したと仮定できます。データを再度適切に組み立てるには、消失したパケットを受信する必要があるので、消失したパケットのシーケンス番号を含むACKパケットを再送し、送信元に再送を促します。

送信者は3回も重複ACKを受け取ったため、パケットが本当に消失したと確信、即座に高速再送を行います。高速再送が始まると、高速再送パケットが送信されるまで、ほかのすべてのパケットの送信が一旦停止されます。このプロセスを図に現したのが図9-7です。

図9-7　重複ACKが高速再送につながる

ファイルtcp\_dupack.pcapに重複ACKと高速再送の例があります。このキャプチャファイルの最初のパケットが図9-8です。

図9-8　ACKが次のシーケンス番号を示している

データ受信者（172.31.136.85）から送信者（195.81.202.68）へ送信されたこのTCP ACKパケット①には、先ほどのパケットで送られた確認データが入っていますが、キャプチャファイルには含まれていません。

注　Wiresharkのデフォルト設定では番号の解析を簡単にするために、相対的なシーケンス番号を使うようになっていますが、次の項の例やスクリーンショットではこの機能は使っていません。この機能をオフにするには、[Edit]で[Preferennces]を選択、[Preferences]ウィンドウで[Protocols]、次いで[TCP]を選びます。そして[Relatice sequence numbers and window scaling]の横のボックスのチェックを外してください。

このパケットのACK番号は1310973186②で、これが次に受信するパケットのシーケンス番号になるはずです（図9-9）。

図9-9　予測していないシーケンス番号のパケット

残念ながら次のパケットのシーケンス番号は1310984130①でした。これは送信の途中でパケットが消失したことを意味しています。パケットの番号がシーケンスから外れていたので、受信者は3番目のパケットで重複ACKを送信します（図9-10）。

図9-10　最初の重複ACKパケット

次のどちらかのようになっていれば、重複ACKパケットです。

・パケット詳細のペインのInfoカラムが、黒字に赤い文字で表示されている。

・パケット詳細のペインのSEQ/ACK Analysisヘディングを広げてみて、1番目のパケットの重複ACKとして掲載されている。

図9-11のように、次のパケットでもこのプロセスが続きます。

図9-11　シーケンス番号が狂ったために生成された重複ACK

送信元から送られた4番目のパケットには、間違ったシーケンス番号がついていたので①、受信者は2個目の重複ACKを送信します②。するとまた間違った番号のパケットが届いたため③、3個目の最後の重複ACKが送られました④。

送信元は3個目の重複ACKを受け取るとすぐに、すべてのパケットの送信を中断し、消失パケットを再送します。図9-12は消失パケットの高速再送です。

図9-12　重複ACKによる消失パケットの高速再送

繰り返しますが、パケットの再送はパケット詳細のペインのInfoカラムで確認できます。先の例同様、黒字に赤字で書かれているのでよく分かります。SEQ/ACK Analysis項目から、これが高速再送であると推測できます①（再度繰り返しますが、このパケットが高速再送だと示しているのはパケットではなく、Wiresharkの機能です）。最終パケットは、高速再送の受信を示すACKパケットとなります。

注　パケット消失が起きたTCP通信のデータフローに影響を与えると思われるのが、セレクティブACK（Selective Acknowledgement）機能です。上のパケットキャプチャでは、最初の3ウェイハンドシェイクプロセスにおいて、セレクティブACKが有効な機能としてネゴシエートされています。この場合パケットが消失して重複ACKを受け取ると、消失後にほかのパケットの受信が成功していれば、消失したパケットのみが再送されます。セレクティブACKが有効になっていないと、消失したパケットのあとに送信されたすべてのパケットも、消失パケットとともに再送信されます。つまりセレクティブACKによって、データリカバリを効率的に行うことができるのです。最新のTCP/IPスタック実装はセレクティブACKをサポートしているので、通常はこの機能が実装されているはずです。

TCPフロー制御

再送と重複ACKは消失したパケットを回復するためのTCP機能です。幸いTCPには、パケット消失を予防する方法も備わっています。

TCPは、パケット消失が起こりそうになると、データ転送レートを調整してこれを防ぐスライディングウィンドウというメカニズムを実装しています。スライディングウィンドウは、受信者の受信ウィンドウを使ってデータフローを制御します。

受信ウィンドウは受信者が指定する値で、TCPヘッダに格納されており（単位：バイト）、TCPバッファスペースに格納したいデータ量を送信者に伝えるものです。このバッファスペースにはデータが一時的に保管され、その後データはアプリケーション層プロトコルに渡されて処理を待ちます。送信元が1回に送信できるデータ量は、ウィンドウサイズフィールドに指定された数字で決まります。送信者がさらに多くのデータを送信するには、先のデータを受信したという、受信者からのACKを待たなければなりません。またそのデータを処理して、TCPバッファスペースを空にする必要があります。図9-13が受信ウィンドウの仕組みを図式化しています。

図9-13　受信者が受け取るデータ量を制御する受信ウィンドウ

クライアント　サーバ

バッファスペース容量　ウィンドウサイズ

2500バイト

2000バイト

ACK

3000バイト

1000バイト

ACK

2500バイト

500バイト

5000バイト

2000バイト

1000バイト

5000バイト

5000バイト

5000バイト

5000バイト

5000バイト

5000バイト

5000バイト

図9-13では、受信ウィンドウサイズが5000バイトのサーバにクライアントがデータを送信しています。クライアントが2500バイトのデータを送信したためにサーバのバッファスペースは2500バイトへと減り、さらに2000バイトが送られたので500バイトへと減りました。次にサーバはこのデータのACKを送ります。バッファのデータを処理すると、また空きができました。このプロセスが繰り返され、クライアントが3000バイト、1000バイトとデータを送信、サーバのバッファは1000バイトへと減少します。クライアントは再度ACKを送り、バッファのデータを処理します。

ウィンドウサイズの調整

ウィンドウサイズの調整はわかりやすいプロセスですが、常にうまくいくとは限りません。TCPがデータを受け取ると、ACKが生成され応答として送られますが、受信者のバッファにあるデータが常に迅速に処理されるわけではないのです。

サーバが複数のクライアントからのパケット処理に忙しいと、バッファを空ける作業が遅くなり、次のデータを受け取る場所が作れなくなります。フロー制御が行われなければ、これはパケット消失とデータ障害につながります。幸いにも、サーバが忙しすぎてデータ処理が遅い場合、受信ウィンドウのサイズを調整することができます。ACKパケットのTCPヘッダで、ウィンドウサイズの値を小さくすればいいのです。図9-14がその一例を示しています。

図9-14　サーバが忙しくなったらウィンドウサイズを調整できる

クライアント　サーバ

2000バイト

2000バイト

ACK　ウィンドウアップデート　1000バイト

800バイト

150バイト

ACK

バッファスペース容量

3000バイト

1000バイト

5000バイト

200バイト

50バイト

1000バイト

ウィンドウサイズ

5000バイト

5000バイト

1000バイト

1000バイト

1000バイト

1000バイト

図9-14で、サーバは5000バイトのウィンドウサイズで始まっています。クライアントが2000バイトのデータを送り、さらに2000バイト送ったので、バッファスペースの空きは1000バイトになってしまいました。サーバはバッファがすぐに埋まってしまうと認識、このペースでデータが転送されれば、パケット消失が起こると考えました。するとサーバは、ウィンドウサイズを1000バイトにするというACKをクライアントに送ります。その結果クライアントから送信されるデータ量が減り、サーバはデータフローを一定速度で保ちつつ、バッファのデータが処理できるようになりました。

ウィンドウサイズは小さくするだけでなく、大きくする場合もあります。サーバがより高速にデータ処理ができれば、ウィンドウサイズを広げるというACKパケットを送信します。

ゼロウィンドウ通知でのデータフローの一旦停止

サーバがこれ以上、クライアントからのデータを処理できなくなる場合があります。原因としてはメモリ不足、処理能力の不足、あるいは他の問題が考えられます。これらが起きるとパケット消失や通信プロセスの停止につながりますが、受信ウィンドウを使えば被害を最小限にとどめることが可能です。

こうした問題が発生すると、サーバはサイズがゼロのウィンドウを含むパケットを送信します。クライアントがこのパケットを受信すると、データ転送は停止しますが、キープアライブパケットを送り、サーバとの接続は維持します。キープアライブパケットは、サーバの受信ウィンドウの状況を確認するため、一定間隔で送られます。サーバはデータ処理を再開すると、ノンゼロウィンドウサイズのパケットを送り、通信が再開します。図9-15はゼロウィンドウ通知の例を示しています。

図9-15　ウィンドウサイズがゼロに設定されるとデータ転送が停止

クライアント

サーバ

2000バイト

2000バイト

ACK　ウィンドウアップデート　0バイト

キープアライブ

ACK　ウィンドウアップデート　1000バイト

900バイト

バッファスペース容量

3000バイト

1000バイト

0バイト

0バイト

1000バイト

100バイト

ウィンドウサイズ

5000バイト

5000バイト

0バイト

0バイト

1000バイト

1000バイト

図9-15では、サーバはウィンドウサイズ5000バイトでデータ受信を開始しています。クライアントから4000バイトのデータを受信すると、プロセッサ負荷が非常に高くなり、クライアントからのデータを一切処理できなくなってしまいました。サーバはウィンドウサイズフィールドを0に設定したパケットを送信、クライアントはデータ転送を中止してキープアライブパケットを送ります。その後サーバが、データ受信が可能になり、ウィンドウサイズを1000バイトにするという通知を送ったので、クライアントはデータ送信を再開しました。

TCPスライディングウィンドウの実践

TCPスライディングウィンドウの論理を説明したので、キャプチャファイルtcp\_zerowindowrecovery.pcapで実際に見てみましょう。

このファイルは192.168.0.20から192.168.0.30へ送られた複数のTCP ACKパケットで始まっています。一番の関心はウィンドウサイズフィールドの値ですが、これはパケット一覧のペインのInfoカラムと、パケット詳細のペインのTCPヘッダの両方で見られます。最初の3つのパケットで、このフィールドの値が減っていっていることがすぐに分かります（図9-16）。

図9-16　パケットのウィンドウサイズが減っている

最初のパケットの8760バイトが2番目のパケットでは5840バイトに、3番目のパケットではさらに2920バイトへと減っています①。ウィンドウサイズの減少は、ホストからの通信に遅延が起きている典型的な証拠です。Timeカラムを見ると、サイズが非常に早く減らされているのが分かります②。ウィンドウサイズがこれだけ早く減らされると、ゼロまで下がる場合が多く、実際に4番目のパケットがそうなっています（図9-17）。

図9-17　ゼロウィンドウはホストがデータをこれ以上受信できないという意味

4番目のパケットも192.168.0.20から192.168.0.30へ送られたものですが、これ以上データを受け取れないと伝えるのが目的です。TCPヘッダにあるゼロの値①、そしてパケット一覧のペインのInfoカラムと、TCPヘッダのSEQ/ACK Analysisセクション②を見れば、これがゼロウィンドウパケットだということが分かります。

ゼロウィンドウパケットが送信されると、192.168.0.30の通信機器は、192.168.0.20からウィンドウサイズが増えたことを通知するウィンドウアップデートを受け取るまで、データを送信しません。ここでゼロウィンドウを引き起こした問題は一時的なものなので、図9-18のように、次のパケットでウィンドウアップデートが送られています。

これでウィンドウサイズは64240バイトまで増えました①。SEQ/ACK Analysisヘディングを見れば、これがウィンドウアップデートだと分かります。

図9-18　データ送信が可能になったことを伝えるTCPウィンドウアップデートパケット

アップデートパケットが受信されると、192.168.0.30のホストは再びデータ送信が可能になるので、6番目と7番目のパケットを送ります。このプロセスは非常に迅速に行われます。少しでも余分に時間がかかると、データ転送の遅延や障害を引き起こすからです。

もう一度だけスライディングウィンドウを見て、ファイルtcp\_ zerowindowdead.pcapを調べてみましょう。最初のパケットは、195.81.202.68から172.31.136.85への普通のHTTPトラフィックです。このパケットのすぐあとに、172.31.136.85からゼロウィンドウパケットが送られています（図9-19）。

図9-19　データ転送を止めるゼロウィンドウパケット

これは図9-17のゼロウィンドウパケットとよく似ていますが、結果はまるっきり違います。172.31.136.85のホストはウィンドウアップデートを送って通信再開を通知するのではなく、図9-20のようにキープアライブパケットを送っています。

図9-20　キープアライブパケットはゼロウィンドウのホストとの接続を維持する

このパケットには、パケット詳細のペインのTCPヘッダのSEQ/ACK Analysisセクションで、キープアライブというしるしがついています①。Timeカラムを見ると、最後のパケットが受信された3.4秒後に、このパケットが送られたことがわかります。図9-21のように、片方のホストがゼロウィンドウパケットを送り、もう一方がキープアライブパケットを送るというプロセスが、何回か続きます。

図9-21　ゼロウィンドウパケットとキープアライブパケットの送信が何度か繰り返される

キープアライブパケットは、3.4秒、6.8秒、13.5秒の間隔で送信されています①。その通信機器が搭載しているOSによっては、このプロセスがかなり長くなります。ここではTimeカラムの値が加算されているので、接続はほぼ25秒間停止します。ドメインコントローラでの認証や、インターネットからファイルをダウンロードしようとしているときに、25秒もの遅れがあったらどうでしょう。我慢できるわけがありません。

TCPエラー制御とフロー制御パケットから学ぶ

再転送、重複ACK、スライディングウィンドウを関連付けてみましょう。遅延問題を解決するときの注意点をいくつか挙げておきます。

再転送パケット

再転送が行われるのは、クライアントが送ったデータを、サーバが受信していないことをクライアントが検出したからです。つまり解析する側によっては、再転送に気付かないわけです。サーバからのデータをキャプチャする場合、クライアントから送信、また再送信されたパケットは受け取れないので、再転送パケットを目にすることがなく、情報のないままとなってしまいます。サーバ側でのパケット消失の被害者かも知れないと思ったら、クライアント側からのトラフィックをキャプチャすることを考えてみましょう。そうすれば再転送パケットの存在が確認できます。

重複ACKパケット

筆者は重複ACKを、再転送の逆のようなものだと考えていました。というのも、クライアントからのパケットが途中で消失したとサーバが検出したときに送られるからです。重複ACKは、両サイドからのトラフィックをキャプチャしたときに見られるもので、受け取ったパケットがシーケンスから外れているときに発生するものであることを思い出してください。たとえばサーバが3個のパケットのうち、1番目と3番目だけを受け取った場合、重複ACKが送られ、クライアントは2番目のパケットを高速再送します。サーバは1番目と3番目を受け取っているので、2番目のパケットが届かないのは一時的な原因によるものであり、通常重複ACKは無事送信、受信されます。もちろんこのシナリオが常に正しいとは限りませんが、サーバ側でのパケット消失が疑われ、重複ACKが見当たらない場合、クライアント側でのパケットキャプチャを検討してください。

ゼロウィンドウとキープアライブパケット

スライディングウィンドウは、サーバがデータを受信、処理できない事態と直接結びついています。ウィンドウサイズが縮小あるいはゼロウィンドウ状態になるのは、サーバで問題が起きているということなので、どちらかの事態が生じたら、原因を調べるべきです。通常はサーバとクライアントの両サイドで、ウィンドウアップデートのパケットが見つかるはずです。

高遅延の原因を突き止める

パケット消失が遅延の原因でない場合があります。2つのホスト間の通信が遅くても、TCP再送や重複ACKが見つからないことがあるのです。このような場合、高遅延の原因を見つける別のテクニックが必要となります。

高遅延の原因を突き止めるもっとも効果的な方法のひとつは、最初のハンドシェイクと、そのあとに送られるいくつかのパケットを調べることです。仮にクライアントがWebサーバと接続していて、そのWebサーバでホストされているサイトをブラウズしようとしたとしましょう。ここで注意が必要なのは、TCPハンドシェイク、最初のHTTP GETリクエスト、GETリクエストに対するACK、サーバがクライアントに送った最初のデータパケットから構成される、最初の６個のパケットです。

注　この項目を正しく理解するには、Wiresharkで時間表示フォーマットを正しく設定する必要があります。[View]から[Time Display Format]、そして[Seconds Since Previous Displayed Packet]を選択します。

正常な通信

ネットワークベースラインの詳細については、本章の後半で説明します。ここでは、高遅延の状況と比較するために、正常通信のベースラインが必要だということだけを理解しておいてください。このあとの例では、ファイルlatency1.pcapを使用します。TCPハンドシェイクとHTTP通信についてはもう説明したので繰り返しません。またパケット詳細のペインも見ません。ここでは図9-22に示したように、Timeカラムだけに集中します。

図9-22　このトラフィックはすぐに始まっており正常とみなすことができる

この通信シーケンスは非常に速いです。プロセス全体に0.1秒もかかりません。

次のいくつかのキャプチャファイルは、同じトラフィックパターンで成り立っていますが、パケットのタイミングが違います。

遅い通信――ケーブルの遅延

今度はファイルlatency2.pcapを見てみましょう。図9-23からわかるように、2つのファイルのパケットは、時間の値を除けばすべて同じです。

図9-23　2番目と5番目のパケットが高遅延を示している

6個のパケットを見ていくとすぐ、最初の遅延のサインに気づきます。クライアント（172.16.16.128）がTCPハンドシェイクを開始するために最初のSYNパケットを送り、サーバ（74.125.95.104）からSYN/ACKを受け取るまでに、0.87秒の遅れが見られます。これはクライアントとサーバ間の通信機器によって引き起こされるケーブル遅延の最初の兆しです。

転送されるパケットの性質から、これがケーブルの遅延であると判断できます。サーバがSYNパケットを受け取る際、応答を送るのに必要な処理はごくわずかです。というのも、トランスポート層の上の処理はサーバのワークロードに関わってこないからです。サーバのトラフィックがかなり多いとしても、SYN/ACKパケットで迅速な応答が可能です。つまりサーバは高遅延の原因ではありません。

クライアントも遅延を引き起こしていません。この時点ではSNY/ACKパケットを受信する以上の処理が行われていないからです。

クライアントとサーバの両方に問題がないとすると、最初の2個のパケットに、遅延の原因があるということになります。

3ウェイハンドシェイクを完了するACKパケット送信はすぐに行われ、クライアントからHTTP GETリクエストが送られます。この2個のパケットを生成する処理は、SYN/ACKを受け取った後にクライアントでローカルに行われるので、クライアントでの処理負荷が重くない限り、2個のパケットの転送は迅速に行われるはずです。

5番目のパケットの時間値も、非常に高くなっています。最初のHTTP GETリクエストが送信されたあと、サーバから戻されたACKパケットが届くまでに1.15秒かかっています。HTTP GETリクエストを受け取ると、サーバはデータ送信を開始する前にまずTCP ACKを送りますが、この処理もほとんど負担はかかりません。これももうひとつのケーブル遅延のサインです。

ケーブル遅延が発生すると、最初のハンドシェイクでのSYN/ACKだけでなく、ほかのACKパケットでも遅延のしるしが伺えます。この情報はネットワークの高遅延の原因を教えてはくれませんが、クライアントとサーバが原因でないのはわかるので、両者間の他の通信機器が遅延の原因だとわかります。するとさまざまなファイアウォール、ルータ、プロキシを調べ、犯人を見つけることができます。

遅い通信――クライアントの遅延

次の遅延のシナリオは、ファイルlatency3.pcapに含まれています（図9-24）。

図9-24　遅いパケットが最初のHTTP GET

このキャプチャは、迅速なTCPハンドシェイクで正常に始まり、遅延の兆しは見られません。ハンドシェイクが完了したあと、4番目のパケットであるHTTP GETリクエストが送られるまでは、すべてが正常のようです。ところがこのパケットで、1.34秒の遅れが発生しました。

遅れの原因を見極めるには、3番目と4番目のパケットの間で何が起きたかを調べなければなりません。3番目のパケットはクライアントからサーバに送られたTCPハンドシェイクの最後のACKパケットで、4番目のパケットはクライアントからサーバに送られたGETリクエストです。両者に共通しているのは、どちらもクライアントから送信されたもので、サーバから独立している点です。これらはすべてクライアントを中心としているので、ACKが送られたすぐあとに、GETリクエストが行われるべきです。

残念ながら、ACKからGETへの移行はスムーズにいきませんでした。GETパケットの生成と転送にはアプリケーション層での処理が必要であり、この処理の遅れというのは、クライアントの処理が追いつかないという意味になります。つまりこの通信の高遅延の原因は、クライアントにあるのです。

遅い通信――サーバの遅延

最後の遅延のシナリオはサーバの遅延で、ファイルlatency3.pcapを使います（図9-25）。

図9-25　最終パケットまで高遅延の兆しが見えない

2つのホスト間のTCPハンドシェイクはスムーズかつ迅速に行われ、うまくいっています。最初のGETリクエストとその応答のACKパケットの配信もすばやく行われました。このファイルでは最終パケットまで、高遅延の兆しが見えないのです。

この6番目のパケットは、クライアントからのGETリクエストに応えてサーバから送られた最初のHTTPデータパケットですが、サーバがGETリクエストに対して送ったTCP ACKの到着から0.98秒遅れています。5番目から6番目のパケットへの移行は、ハンドシェイクでのACKからGETリクエストへの移行と非常によく似ています。しかしここでは、サーバが問題となります。

5番目のパケットは、クライアントから受け取ったGETリクエストに対するサーバのACKです。サーバはパケットを送信したらすぐ、データ送信を開始しなければなりません。このパケットのデータへのアクセス、パッケージ、転送はHTTPプロトコルによって行われます。そしてこれはアプリケーション層プロトコルなので、サーバによる処理が多少必要となります。このパケットの遅れは、サーバがデータ処理に追いついていないしるしなので、高遅延の理由はサーバだということになります。

遅延を見つけるフレームワーク

6個のパケットを使って、ネットワーク高遅延の原因を探しました。これらのシナリオは少々複雑に思えますが、図9-26が実際の遅延問題を解決するときに役立つはずです。こうした基本はどんなTCP通信にも当てはまります。

注　UDP遅延についてはほとんど触れていません。UDPは高速ですが信頼性が低く、遅延を検出し回復する機能を内蔵していないからです。その代わり、データ配送の信頼性を高めるために組み合わされたアプリケーション層プロトコル（とICMP）が、その役割を担っています。

図9-26　実際の遅延問題の解決に使える図表

クライアント　サーバ

SYN　SYN/ACK　ACK　レイヤ7プロトコルリクエスト　ACK　レイヤ7プロトコルデータ

ケーブル遅延　クライアント遅延　サーバ遅延

ネットワークベースラインの確立

何もかもがうまくいかない場合、ネットワークベースラインが、ネットワークの遅延を解決するための重要なデータとなります。ここではネットワークベースラインを、「正常な」ネットワークトラフィックの大きなかたまりを含む、ネットワーク上のさまざまなポイントから集めたトラフィックのサンプルから構成されているとします。ネットワークベースライン確立とは、ネットワークや通信機器が正しく機能しないときに、比較対象とするためのものです。

たとえばネットワークでつながっている複数のクライアントが、ローカルのWebアプリケーションサーバへログインするのに時間がかかると苦情を言ってきたとしましょう。このトラフィックをキャプチャしてネットワークベースラインと比較すると、Webサーバは正常に応答しているのに、Webアプリケーションに組み込まれた外部コンテンツによる外部DNSリクエストの速度が、正常時より2倍も遅いことが分かりました。

ネットワークベースラインと比べなくても外部DNSサーバが遅いと気づくかも知れませんが、小さな変化だとそうはいきません。10個のDNSクエリの処理が正常時より0.1秒遅いのは、1個のクエリに正常時より1秒以上かかるのと同じくらい問題ですが、ネットワークベースラインがなかったら、前者を検知するのはかなり困難です。

同じネットワークは存在しないため、ネットワークベースラインのコンポーネントもかなり違ってきます。次の項ではネットワークベースラインのコンポーネントの例を挙げます。すべてが自分のネットワークに該当する場合もあれば、ほとんど当てはまらない場合もあるでしょう。いずれにせよ、ベースラインの3つの基本カテゴリーであるサイト、ホスト、アプリケーションに、各コンポーネントを配置することになります。

サイトベースライン

サイトベースラインの目的は、ネットワーク上の物理的なサイトのトラフィックの、全体的なスナップショットを得ることです。WANのすべてのセグメントで実行するのが理想です。

このベースラインに含まれるコンポーネントは次のようになります。

使用しているプロトコル

[Statistics]から[Protocol Hierarchy]を選択して[Protocol Hierarchy Statistics]ウィンドウを開き、ネットワークエッジ（ルータ/ファイアウォール）でセグメントのすべての通信機器のトラフィックをキャプチャして、すべての通信機器からのトラフィックを見ます。これと照らし合わせることで、正常時に見えるプロトコルが存在しているかどうか、あるいは新しいプロトコルが導入されていないかがわかります。プロトコルをベースとした一部のトラフィック量が、大幅に増えていないかどうかも分かります。

ブロードキャストトラフィック

ネットワークセグメント上のすべてのブロードキャストトラフィックが含まれます。サイトのどのポイントでスニッフィングしても、ブロードキャストトラフィックがキャプチャできるので、正常時は誰、または何が大量のブロードキャストトラフィックを送っているかが分かり、ブロードキャストの量が多すぎないかまたは少なすぎないかが分かります。

認証シーケンス

アクティブディレクトリ、Webアプリケーション、組織固有のソフトなど、あらゆるサービスの認証プロセスからのトラフィックが含まれます。一般に認証には時間がかかります。ベースラインによって、認証が通信遅延の原因かどうかが判断できます。

データ転送レート

ネットワーク上のあるサイトから他の様々なサイトへの、大量のデータ転送の測定から構成されています。キャプチャのサマリーとWiresharkのグラフ機能を利用して、転送率と接続の整合性が確認できます。おそらくこのサイトベースラインが一番重要になるでしょう。接続が遅く感じられる場合、ベースラインと同じデータの転送を実行し、その結果を比較します。そうすれば接続が本当に遅いのかどうかがわかり、またどこで問題が起きているのかを探る手がかりにもなります。

ホストベースライン

ホストベースラインといっても、すべてのホストのベースラインを確立する必要はありません。トラフィックがもっとも多い、またはミッションクリティカルなサーバでのみ行えば十分です。サーバが遅いと管理部門から怒りの電話がかかってくるので、そのホストのベースラインがあればいいのです。

ホストベースラインのコンポーネントは以下の通りです。

使用しているプロトコル

[Protocol Hierarchy Statistics]ウィンドウを使って、ホストからのトラフィックをキャプチャします。これと照らし合わせることで、正常時に見えるプロトコルが存在しているかどうか、あるいは新しいプロトコルが導入されていないかがわかります。プロトコルをベースとした一部のトラフィック量が、大幅に増えていないかどうかも分かります。

アイドル/ビジーなトラフィック

ピーク時またオフピーク時の、正常なトラフィックのキャプチャで構成されています。時間帯による接続数や使用帯域を把握しておけば、遅延がユーザー負荷のせいなのか、それともほかに問題があるのかが分かります。

起動／シャットダウン

このベースラインを得るには、ホストの起動とシャットダウンの際に生じるトラフィックをキャプチャする必要があります。コンピュータが起動しない、シャットダウンしない、あるいは起動やシャットダウンの際の速度が極端に遅い場合、このベースラインと照合し、遅延がネットワーク関連かどうかを判断できます。

認証シーケンス

あらゆるサービスの認証プロセスからのトラフィックをキャプチャする必要があります。一般に認証には時間がかかります。ベースラインによって、認証が通信遅延の原因かどうかが判断できます。

関連／依存

このホストがどのホストに依存しているか（またこのホストに依存している他のホスト）を知るために、長い時間をかけたキャプチャを行います。[Statistics]から[Conversations]を選択して[Conversations]ウィンドウを開き、関連と依存を見ます。この例では、WebサーバがSQLサーバホストに依存しています。普段はあまりホスト間の依存関係を意識していないので、ホストベースラインが役立ちます。これによって、ホストが正しく機能しないのが誤動作のためなのか、高遅延の通信機器の依存のためなのかが分かります。

アプリケーションベースライン

ネットワークベースラインの最後のカテゴリーは、アプリケーションベースラインです。ビジネスクリティカルなネットワークベースのアプリケーションには必須です。

アプリケーションベースラインのコンポーネントは以下のとおりです。

使用しているプロトコル

このベースラインでも、Wiresharkの[Protocol Hierarchy Statistics]ウィンドウを使いますが、今回はアプリケーションを実行しているホストからのトラフィックをキャプチャします。このリストと比較すれば、そのアプリケーションが依存するプロトコルが、正常に機能しているかどうかが分かります。

起動／シャットダウン

このベースラインを得るには、アプリケーションの起動とシャットダウンの際に生じるトラフィックをキャプチャする必要があります。アプリケーションが起動しない、あるいは起動やシャットダウンの際の速度が極端に遅い場合、このベースラインと照合し、原因を突き止めることができます。

関連／依存

このアプリケーションがどのホストまたはアプリケーションに依存しているかを知るために、[Conversations]ウィンドウを使って、長い時間をかけたキャプチャを行います。普段はあまりアプリケーション間の依存関係を意識していないので、このベースラインが役立ちます。これによって、アプリケーションが正しく機能しないのが誤動作のためなのか、高遅延の通信機器の依存のためなのかが分かります。

データ転送レート

キャプチャのサマリーとWiresharkのグラフ機能を利用して、正常時のアプリケーションサーバへの転送率と接続の整合性に基づくベースラインが作成できます。アプリケーションが遅いという報告があれば、このベースラインを使って、遅延の原因が、使用度が高すぎるためなのか、ユーザー負荷が高いためなのかを判断できます。

ベースラインについての追記

ネットワークベースラインを作成する際に、心に留めておくべき点を2、3追加しておきます。

・ベースラインを作成する場合、1つにつき最低3回は作成しましょう。トラフィックの少ない時間帯（早朝）、トラフィックの多い時間帯（午後）、トラフィックがほとんどない時間帯（深夜）といった具合です。

・可能なら、ベースラインを作成しているホストから、直接キャプチャするのは避けましょう。トラフィックが多い時間帯だと、キャプチャが負荷を増やし、パフォーマンスを妨げ、パケットの消失を導いて、ベースラインを無効にしてしまう場合があります。

・ベースラインにはネットワークに関する秘匿情報が含まれているので、セキュリティに注意しましょう。許可された人々だけがアクセスできる安全な場所に保存することです。同時にすぐ利用できるよう、手元に置いておきましょう。USBドライブか、暗号化したパーティションに入れておくべきです。

・ベースライン関連の.pcapファイルをまとめ、関連や平均データ転送レートなど、よく参照する値の「カンペ」を作っておきましょう。

まとめ

この章では、ネットワーク遅延のトラブルシューティングに焦点を当てました。TCP問題を検出し、リカバリーするためのより便利な、信頼性の高い方法を説明し、ネットワーク通信の高遅延の原因を探る方法を実演し、ネットワークベースラインの重要性とそのコンポーネントについて説明しました。ここで説明したテクニックと、Wiresharkのグラフおよび解析機能（5章で説明）を使えば、ネットワークが遅いという苦情の電話を受けても、対応することができます。