ゼミ: Computer Networks 5th edition: 6.5.10

久野　靖 *

2014.12.19

5.10　TCP の混雑制御

□ TCP の主要機能の 1 つ、混雑制御の話は最後までとってあった。

- どんなネットでも、容量以上の負荷が来たら混雑。インターネットも同じ。
- ネットワーク層はキューの伸びを検出し対処しようとする→できることといえば、パケットを捨てること
- ネットワーク層のこのふるまいを感知して通信を絞りネットに入る負荷を調節するのはトランスポート層の仕事
- インターネットでは TCP が、エラーのない伝送に加えて、この混雑への対応を主に受け持っている

□ 混雑制御の一般的な話は 6.3 でやった。

- 鍵となるのは　AIMD(additive increase, multicative decrease) →これにより公平かつ効率のよい帯域割り当てに収束
- TCP はこれを窓制御と併せ、パケット喪失という 2 値信号に基づき行う
- このため TCP は混雑窓 (congestion window) を管理
- そのサイズは送信側がネットワーク中を流しているバイト数
- 送信速度は「混雑窓サイズ÷ RTT」で、これを AIMD で調整

□ 混雑窓は流量制御窓 (flow control window) と「一緒に」維持

- 後者は受信側が持っているバッファのバイト数
- 両方を維持し、送信できる量はその両者の小さい方の値
- 例:　受信側は「64k 送っていいよ」しかし送信側は「32k より沢山送ったら詰まる」と思えば 32k だけ送る
- 送信側が「128k まで送っても詰まらない」と思っても受信側に 64k しかバッファがなければやっぱり 64k だけ送る

□ 混雑制御は Van Jacobson(1988) のすばらしい物語の結果

- 1986 に開始した初期のインターネットは急速に利用拡大→混雑崩壊 (congestion collapse) で goodput(有効スループット) が 1/100 に
- Jacobson らは何が起きているのか、どうやって克服するか研究

□ Jacobson らがやった手直しとは AIMD 混雑窓を近似的に入れること

- 面白いのは既存の TCP 実装にどうやってこれを入れたか
- メッセージ形式とかは変えられない (すぐに設置できないから)
- まず気づいたこと:　パケット喪失は混雑を的確に表す信号
- 実際にはこの信号は「混雑してしまってから」来るのでやや遅いが、極めて信頼できる
- 結局、過負荷になってもパケットを落とさないルータなんて作れないからそうなる (ルータに 1TB 積んでも 1Tbit/sec の時代になってすぐに満杯に)

□ ただし、喪失が混雑の信号として信頼できるためには、伝送エラーはごく少なくないといけない

- 無線の場合はそうは行かないので→無線リンクはそのリンクだけの再送機能を持たせて喪失を回復
- この結果、無線でも有線 (銅線、ファイバー) でもエラー率は低い

□ というわけで、TCP はパケット喪失があったら混雑をうたがい、注意深く兆候を調べる

- 的確かつ迅速にパケット喪失に対処できるためには、よい再送タイマーが必要だが → 既に TCP では RTT の平均や変動を見積もってあった
- Jacobson の重要な 1 歩はこのタイマーに変動率を追加したこと
- よい再送タイムアウトを得ることはネットを飛んでいるデータ量を知ること (送信バイトと ACK バイトの差がデータ量)

*筑波大学ビジネスサイエンス系

□ とすれば話は簡単? 混雑窓を追跡して AIMD で制御すればいい? そんな簡単ではない。

- まず、パケット注入率は (短期的にでも) ネットに合ったものである必要
- 例: ホストの混雑窓が 64K で、1G のイーサネットに接続→ 64K 一気に送信?
- 途中に 1Mbps の ADSL がはさまっていたら、その入口を 0.5 秒もふさいでしまう → VoIP とかが流れていたら致命的
- 一気に送るというのは混雑を産もうとしてるのに等しい

□ 一方で、小さいバーストは望ましい。図 6-43 参照。

- 少数のパケット (4 個) をバーストで送る→ 1M の入口で待たされるがキューは長くない→1M 線の上ではのびてる
- 受信側に来て ACK が返される→ ACK の時間も伸びている→どれくらい伸びてるかで最も遅いリンクがどれだけか分かる! (ack clock)
- 以後、この間隔で送信すれば待ちなしで通過することになる
- ack clock によって送信率を制御することは TCP の心臓部の 1 つ

□ 次の問題は、AIMD は小さい窓サイズから始めるとバランスまで時間が掛かるということ

- 例: 10Mbps で RTT が 100msec だと、混雑窓サイズは 1Mbit (1250 バイトパケットであれば 100 個)
- 1 パケットからはじめて 1 ずつ増加させたら、100RTT(10 秒) たたないとバランス点まで来ない→遅すぎ
- じゃあ 50 パケットから始めるか? → 遅いリンクには致命的に大きい

□ Jacobso 解法: 線形と定数倍の両方をまぜる

- リンク確立→混雑窓は小さい初期値として 4 セグメントに (RFC399) →到達して (タイムアウトなしで)ACK が帰ってきたら、1 つ帰るごとにもう 1 セグメント増やす→ RTT ごとに倍々になっていく
- これは slow start と呼ばれるが倍々なので全然 slow ではない (図 6-44)
- これを見ると、常に混雑窓サイズは現在送られつつある (ACK が来ていない) パケットの数に一致
- 連続して送ったパケットの間隔は遅いリンクがあると開いている (ack clock) →これの間隔を守って送る

□ 倍々で増やすので速やかにネットワークの容量にぶつかる

- キューが積み上がり、待ちやパケット喪失が発生→送信側でタイムアウト→そうしたらそれ以上増やさなくなる
- 毎回ぶつかるまでやらないため、slow start threshold という値を維持→ 最初は大きい値 → タイムアウトしたら現在の混雑窓サイズの半分をこのしきい値として設定
- 最後に倍にしたことで超えたので、半分というのは適正量のよい見積もり

□ しきい値を超えたら slow start から AIMD の定数増加に切り替え

- 1RTT ごとに 1 ずつ混雑窓を増加。M を最大セグメントサイズ、C を混雑窓サイズとして、C/M パケット ACK されるごとに M*M/C だけ C を増やすとかが典型的
- すでに最適値に近づいてきているので、ゆっくり調節でよい (図 6-45)

□ 性能のための工夫はほかにも可能

- タイムアウトを待つのは (タイムアウトは保守的に設定するため長めなので) 遅くなる
- パケットが喪失するとその後 ACK の番号が増えなくなり、混雑窓が一杯なのでそれ以上送れない状態がしばらく続く
- タイマーが発火して再送されるとそこから再度 slow start になる

□ 「同じ番号の ACK が続いて来た」(重複 ACK, duplicate ack)

- 受信側にパケットが到着したが喪失パケットがあるため番号が進まないと分かる
- パケットの経路が 1 つでないため追い越す場合もあるが実際にはまれ
- そこで TCP では 3 つ重複 ACK があったらその番号のパケットを再送 (fast retransmission)
- その場合も喪失を仮定するので混雑窓サイズを半分にして slow start
- 1RTT 後に喪失パケットが ACK されたらその先のパケットが送られる

□ ここまでの混雑制御の様子→図 6-46 (1988 年の 4.2BSD-Tahoe リリースに搭載されたものがこれ)

- セグメントは 1KB。最初混雑窓は 64KB だがタイムアウトのため 32KB に減少させ、1 パケットから slow start、32 で 1 ずつ増加
- ラウンド 13 のセグメントが再び喪失し、3 重複 ACK で検出されて 20KB で再度 slow start、以下これが繰り返される

- TCP-Tahoe(再送タイマーも適切な値) により混雑崩壊が回避された

□ Jacobson はさらに改善できると認識

- 最初の再送のとき、混雑窓サイズは大きすぎるが、適切な ack clock で進んではいる→重複 ACK が来るたび、パケットが 1 つ到着したと分かる→ネットに残っているパケットが見積もれる
- fast recovery: 重複 ACK(最初の 3 連続を含む) を数え、残っているパケットが新たな (半分にした) しきい値より下がったら以後の重複 ACK に対応して 1 パケット送信する
- fast retransmission から 1RTT たつと喪失パケットが ACK されて重複 ACK は止まる→ fast retransmission を終わって 1 ずつ増加

□ fast retransmission により、再度の slow start を回避できる

- ただし 2 つ以上喪失した場合はやはり slow start になる
- 通常の場合は slow start が回避され、のこぎり状に進む (図 6-47)
- 1990 の 4.3BSD-Reno に搭載。TCP-Reno は TCP-Tahoe + fast recovery
- これによりほとんどの時間、混雑窓サイズは最適値近辺で推移
- この基本方式で 20 年以上 (30 年以上?) 進んで来ている (曖昧さの除去とかチューニングとかはあったが)

□ いくつかの実装の改良

- 2 つ以上のパケット喪失に対する改良 (TCP-NewReno, RFC3782)
- いくつか別バージョン: CUBIC TCP(Linux)、Compound TCP(Windows)

□ TCP そのものに対する改良 1 --- SACK

- 重複 ACK から喪失を推測するところが複雑 (最終番号では情報不足)
- SACK(selective ACK) --- 受信ずみのバイト範囲を 3 組まで通知 →これを見れば適切な再送と伝送中パケットの正確に見積もりが可能
- 接続時に SACK 可能とのオプションを交換すると使えるようになる
- 実際に使うようすは図 6-48
- SACK の情報はあくまでもオプションで、無しでもこれまで同様動作
- しかしこれがあると再送や見積もりが正確になる → 広く実装

□ TCP そのものに対する改良 2 --- ECN

- ECN(explicit congestion notification) --- IP の機能
- これも接続時にオプションで合意すれば使える → IP ヘッダに ECN 使用のフラグを立てる
- ルータは混雑が近づいている場合には ECN ビットを立てる
- 受信側は ECN が立っていたら ECE(ECN Echo) ビットを立てたセグメントを返送
- 送信側は CWR(Congestion Window Reduced) ビットを立てたセグメントで応答
- 送信側の動作は重複 ACK の場合と同じだが、まだパケットロスは出ていないので再送など必要なく有利 ← RFC3168、まだ広まってない

5.11 The Future of TCP

□ TCP はインターネットの担い手として広く使われて来た

- 広い範囲のネットワークでよい性能、細かな改良が続いている→ これからもたぶんそうだが。話題が 2 つ

□ TCP は全てのプログラムが欲するトランスポートセマンティクスを提供しているわけではない

- レコードバウンダリのあるメッセージが欲しい人もいる
- 複数の通信を束ねたいという人もいる (Web とか)
- ネットワーク上の経路をもっと制御したい場合もある

□ 現状ではこれらのミスマッチの解消はアプリケーション任せ

- →少し違うインタフェースを持つものの提案
- SCTP(Stream Control Transmission Protocol, RFC4980)
- STP(Structured Stream Transport)
- しかし、これまでうまく動いて来たものを変えることに抵抗もある

□ 混雑の問題→これまでので解決されたというわけではない

- スピードが速くなるとパケットロス率は非常に小さい必要
- 例: 速度 1Gbps、RTT 100ms、1500 バイトパケットだと、パケットロスはおおよそ 10 分に 1 回 (ロス率 2*10**-8) でないといけない

- これでは小さすぎて混雑シグナルとして有効に機能しない
- 他の原因によるロスによって撹乱されてしまう
- 今のところは問題になっていないが、さらにネットが速くなると分からない
- 別の方法が多く研究されている。喪失ではなく RTT の変化で検出とか← FAST TCP