TCP詳説

西田佳史 Yoshifumi Nishida
nishida＠csl.sony.co.jp

Contents

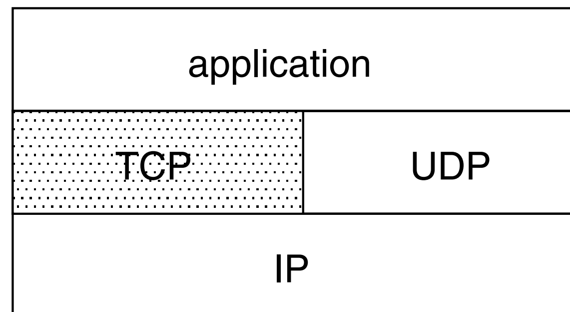
- □Part1: TCP基本機能
- □Part2: TCP詳細機能
- □Part3: TCPと輻輳制御
- □Part4: 採用されつつある新しい機能
- □Part5: TCPとセキュリティ
- □Part6: これからの技術動向

TCPの基本機能

- □TCPとは？
- □TCPの特徴
  - ○非構造化ストリーム
  - ○全二重通信
  - ○コネクション指向
  - ○高信頼性サービス

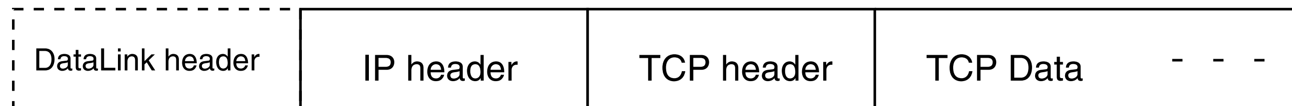
TCPとは？

- Transmission Control Protocol
  - トランスポート層プロトコル
    - IPの上位プロトコル



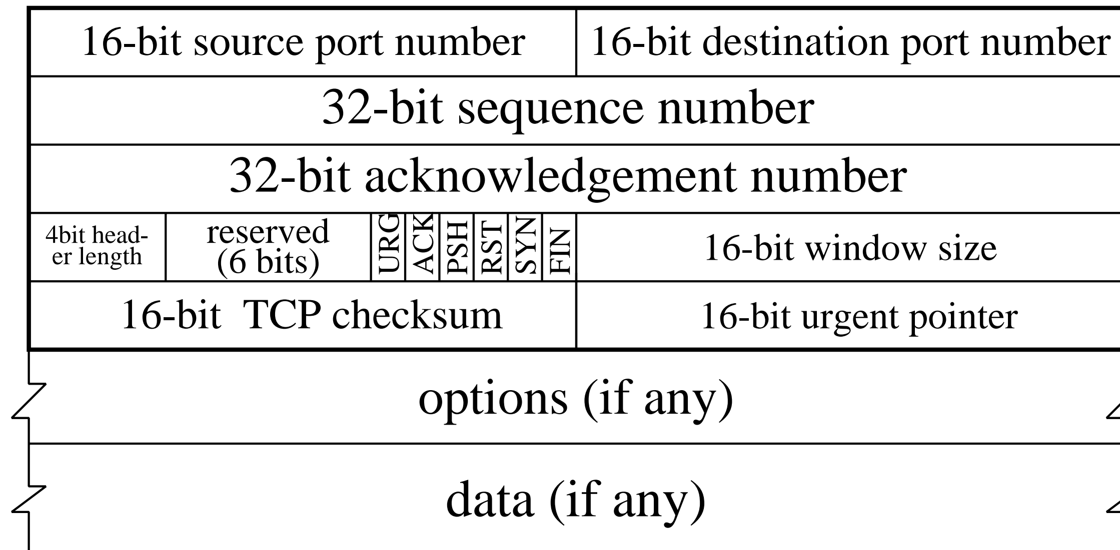
プロトコル階層図

  - パケットフォーマット



TCPの基礎

□ヘッダフォーマット

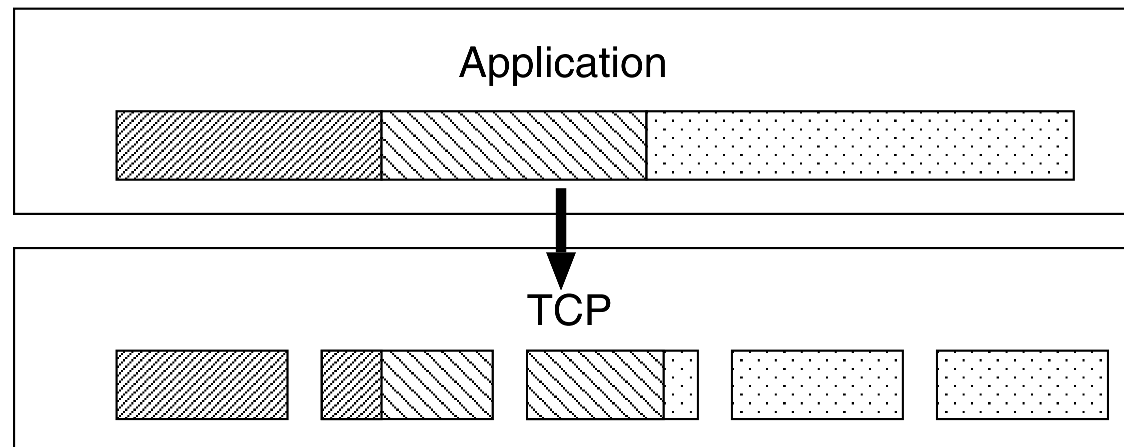


□TCPの仕様:

- RFC793 .. 基本スペック
- RFC2581 .. 輻輳制御アルゴリズム

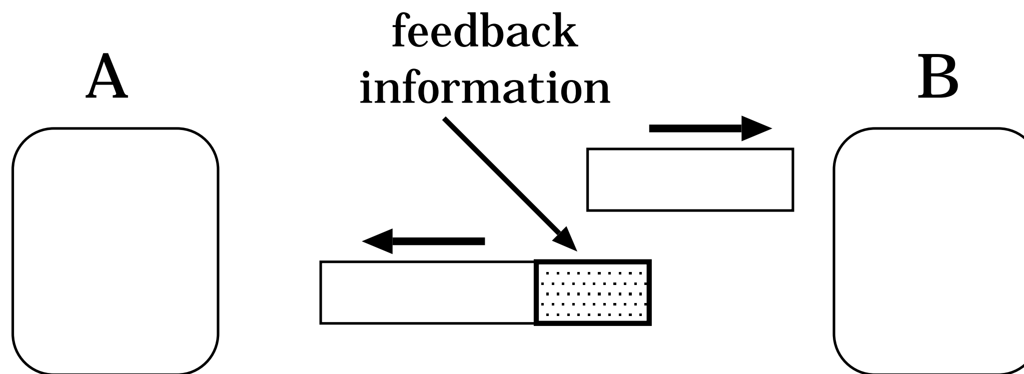
非構造化ストリーム

- □TCPが扱うデータは構造を持たない
  - ○単なるビット列と見做す
    - ▹送信者が送ったストリーム列をそのまま受信者へ
    - ▹データの構造の解釈は上位層にまかせる
  - ○TCPが決めた大きさでストリーム列を分割する
    - ▹通信経路に最適なセグメント長を推測する
  - ○UNIXのファイルシステムと親和性が高い



全二重通信

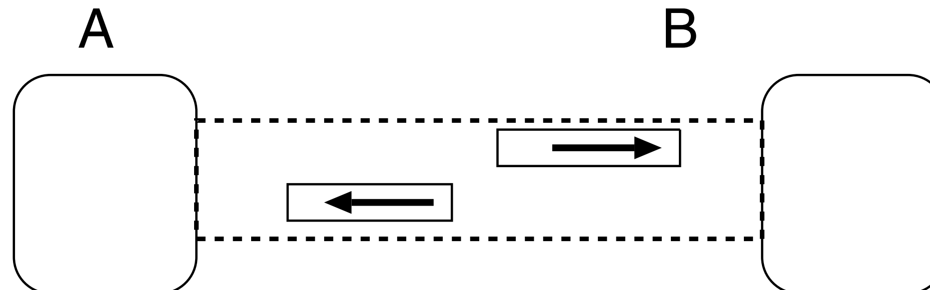
- 通信する両者が同時にデータストリームを送信できる
  - 受信しながら送信が可能
  - Piggybackを使うことにより効率が向上する



  - 半二重にすることも可能
    - コネクション確立時は全二重
    - 片方向づつ通信をshutdownできる

コネクション指向 (1)

- データ送信の前に送信者と受信者がnegotiate
  - 相手の受信能力の把握
    - 最大のウインドウサイズを通知する
  - 通信経路の把握
    - 接続されたインターフェースのMTUを通知
    - 通信経路に最適なパケット長を決定する(Max Segment Size)
      - フラグメントを避ける
- バーチャルサーキット
  - 送信者と受信者の間に仮想的なパイプを確立する

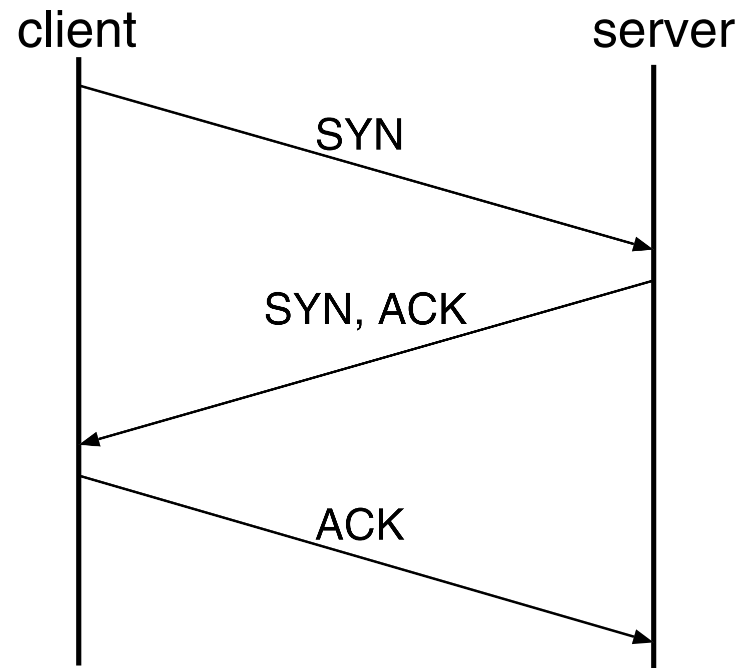


コネクション指向 (2)

- コネクションの端点
  - IPアドレスと ポート番号のペアで識別する
    - 例:
    - (127.0.0.1, 21) - (192.168.0.1, 20)
    - (127.0.0.1, 21) - (192.168.0.1, 21)
  - 全てのTCPパケットは、コネクションの識別情報を含んでいる
    - 送信アドレス、送信ポート
    - 受信アドレス、受信ポート

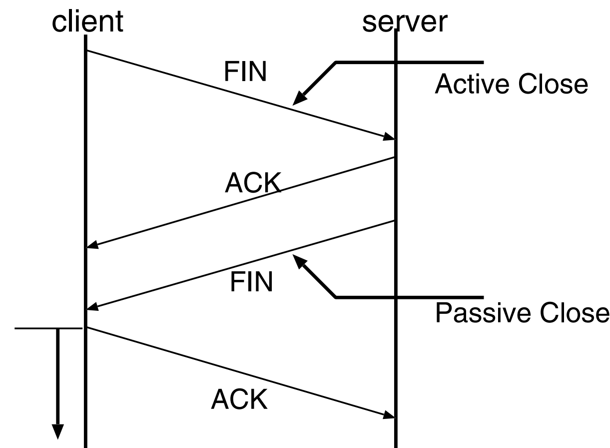
コネクション指向 (3)

- コネクションの確立
  - 3way handshake
    - 確立要求(SYN)と確認応答(ACK)を交換
    - 受信側は「確立要求」と「確立要求の確認応答」を同時に送信
    - 3パケットの交換でコネクションが確立



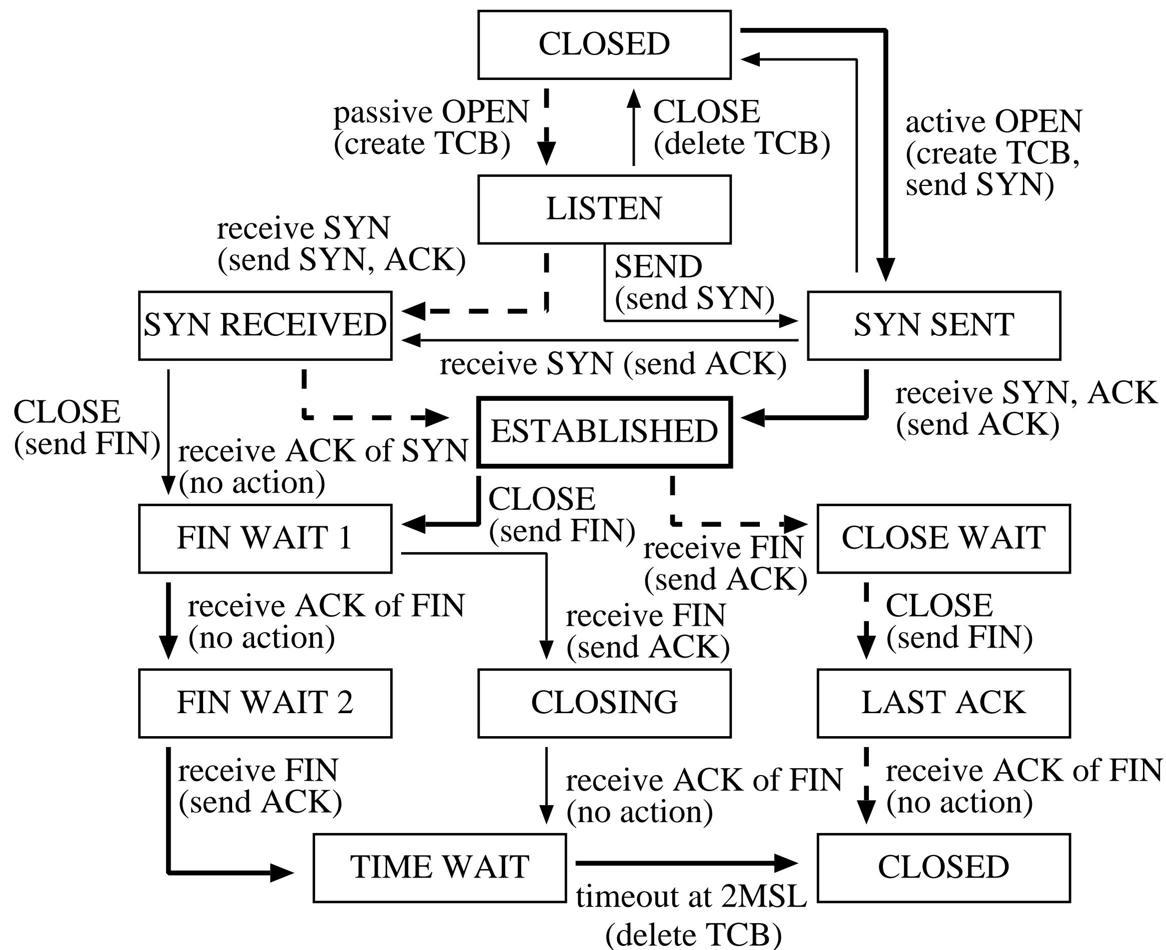
コネクション指向 (4)

- コネクションの終了
  - ハーフクローズ
    - コネクションを片方づつ閉じる
    - active closeする側は最後のACKを送った後、一定時間(2MSL)待つ
      - 最後のACKの送達を確認する
      - passive close側は最後のACKが来なければ FINを再送
      - 通常はクライアントがactive closeする
        - サーバの負担を軽くする



コネクション指向 (5)

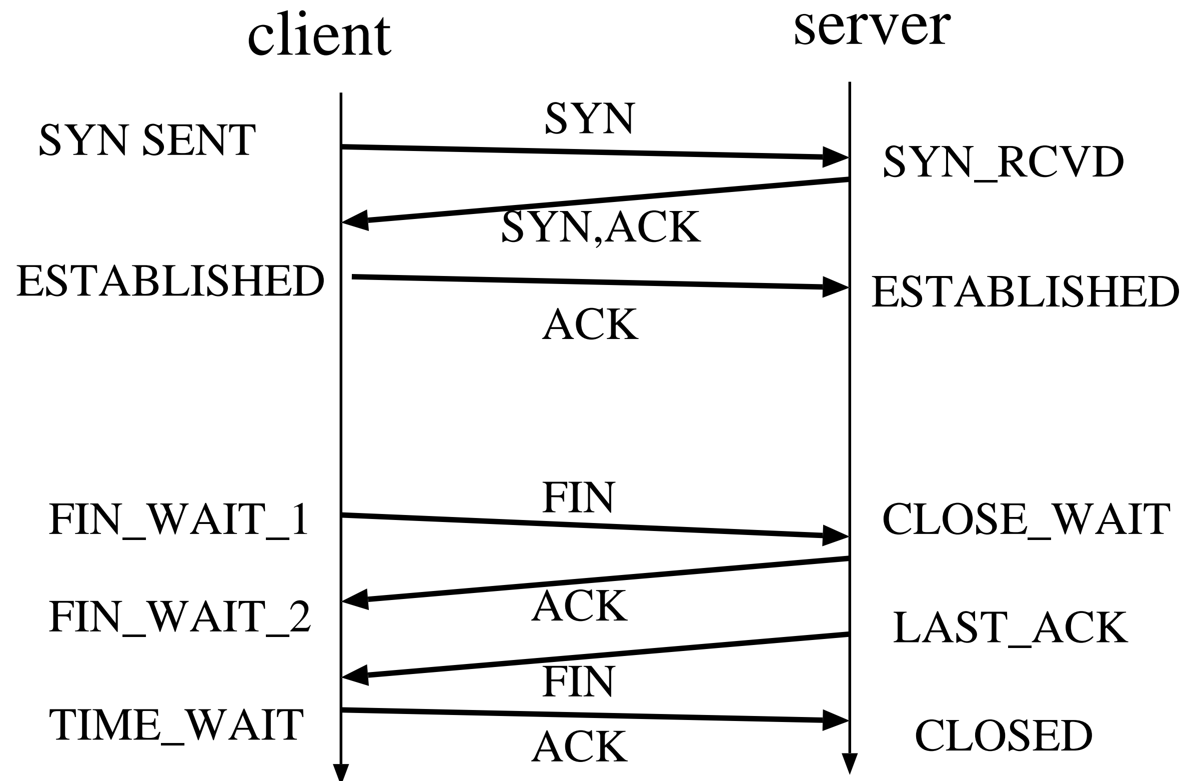
- TCPは状態を持つ
  - 状態遷移図



コネクション指向 (6)

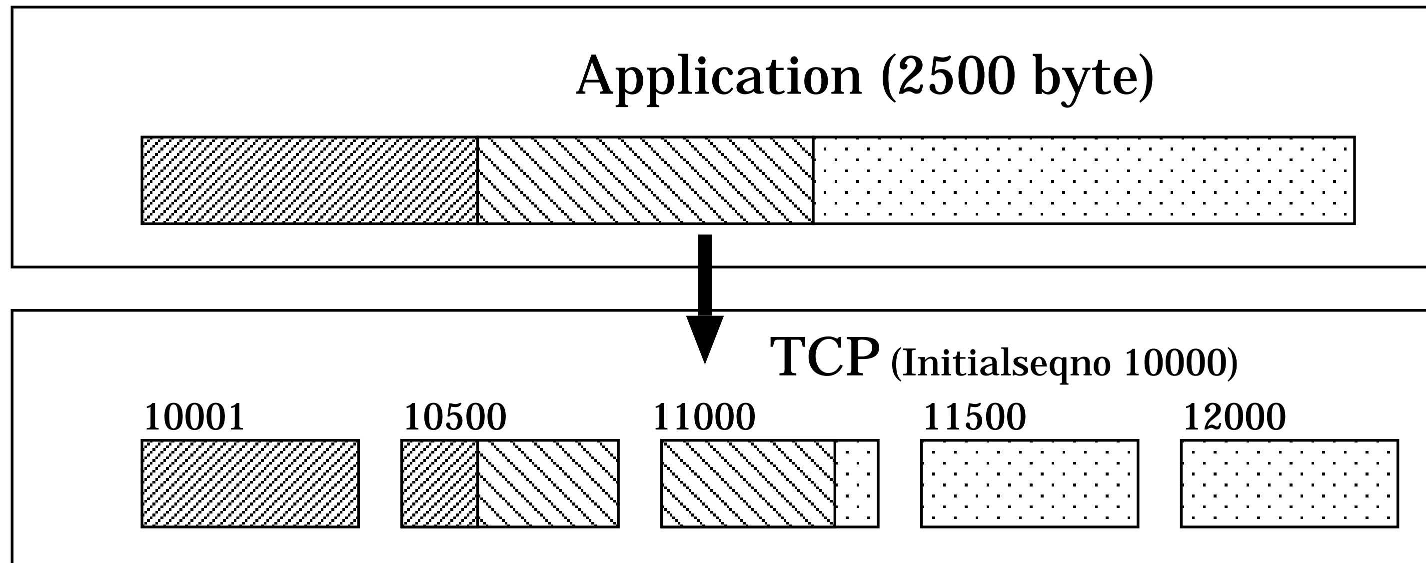
□状態遷移の例

○標準的なコネクション



高信頼性サービス (1)

- シーケンス番号
  - パケット中のデータのバイトストリーム中の位置を示す
    - パケットの順番を保証する
    - 喪失したパケットを検出する
      - 初期シーケンス番号 + ストリーム中の位置



高信頼性サービス (1)

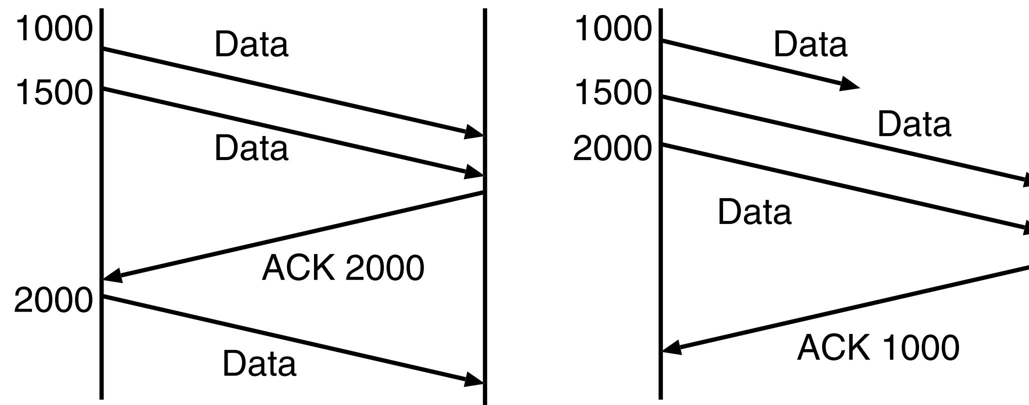
□再送機能

○確認応答(ACK)の利用

▹累積確認応答方式

▵連続して到着した最大のseqno +1を返す

▵確認応答のパケット数を抑える



□再送のタイミング

○再送タイマがタイムアウト

○重複する確認応答が一定数以上到着する

高信頼性サービス (2)

- チェックサム機構
  - 強固なチェックサム機構
    - IPv4はヘッダだけチェックサムを計算する
    - IPv6はチェックサムの計算をしない
  - ヘッダとデータでチェックサムを計算
    - 仮想的なIPヘッダを付加して計算
    - ヘッダ部、データ部全体で計算する

32bit sender IP address		
32bit receiver IP address		
0	proto number	TCP segment length

仮想ヘッダフォーマット

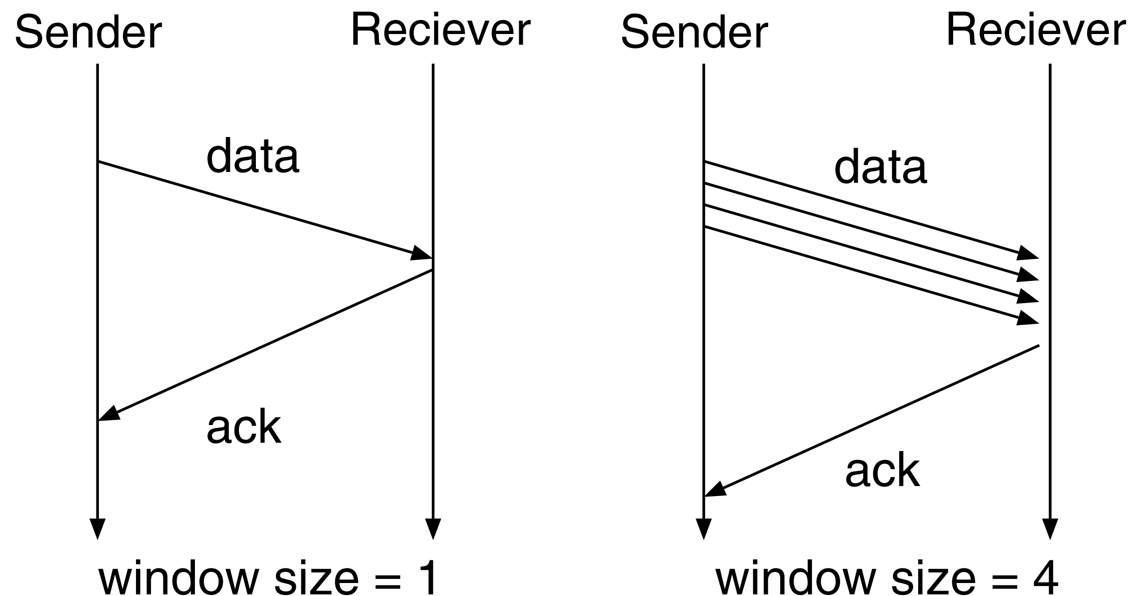
高信頼性サービス (3)

- フロー制御機構
  - 通信状態に合わせてデータの転送速度を調節
  - ２つの役割
    - 相手の処理速度に合わせる
      - 受信者が処理できない速度では転送しない
    - ネットワークの状態に合わせる
      - 低速回線では低速に、高速回線では高速に通信
      - ネットワークが混雑している時は速度を落す →輻輳制御

高信頼性サービス (3)

□スライディングウインドウ方式によるフロー制御

- ○ACKを受け取るまで受け取ることができるパケット数を制御
  - ▹ウインドウサイズを増加→転送レートを上げる
  - ▹ウインドウサイズを減少→転送レートを下げる
- ○受信側が自分のウインドウサイズを通知する
  - ▹受信側が最大の転送レートを決定する



TCP詳細機能

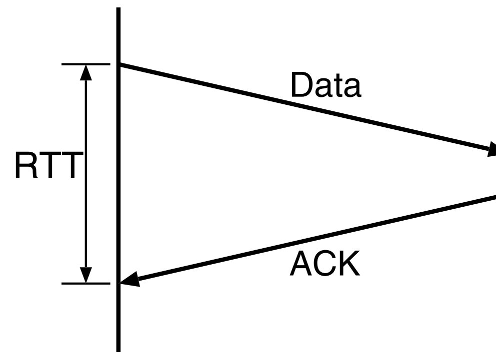
- TCPのタイマ機構
- 遅延確認応答
- Nagleアルゴリズム
- シリーウインドウシンドローム
- TCPのフラグ

TCPのタイマ機構

- 基本機構
  - スロータイマ
    - 遅い間隔で起動される(berkley実装で500msec)
      - 再送タイマ
      - パーシストタイマ
      - キープアライブタイマ
      - 2MSLタイマ
  - ファーストタイマ
    - 早い間隔で起動される(berkley実装で200msec)
      - 遅延確認応答
      - Nagleアルゴリズム

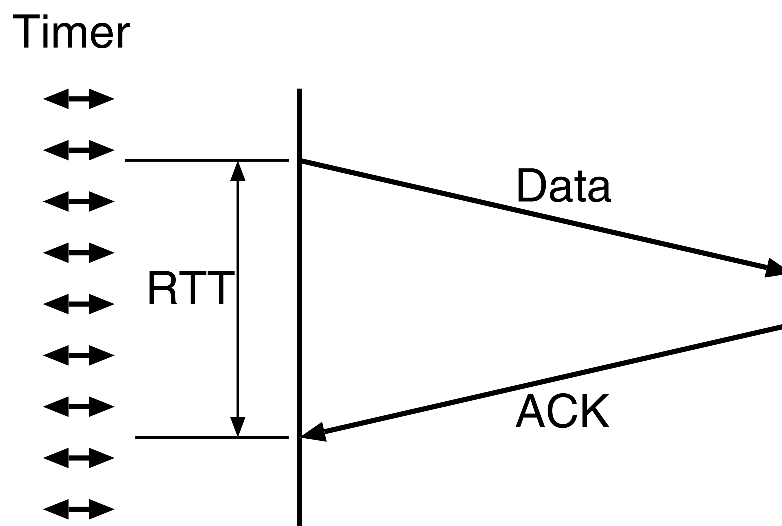
再送タイマ

- パケットが転送される度にセットされる
  - 再送タイマがexpireするとパケットの再送が行われる
  - 適切なタイムアウト値の選択
    - タイムアウトが長すぎる→スループットの低下
    - タイムアウトが短すぎる→不必要な再送の発生
- 再送タイマのexpire時間
  - RTT(Round Trip Time)から算出
    - データが送信してから送達確認を受信するまでの時間



RTTの計算

□スロータイマの割り込み回数を計測



○多くの実装では １RTTに１回計測する

○計測値の平滑化

- 実測値: rtt
- 平滑化した値: srtt

▹srtt = α × srtt + (1 - α) * rtt

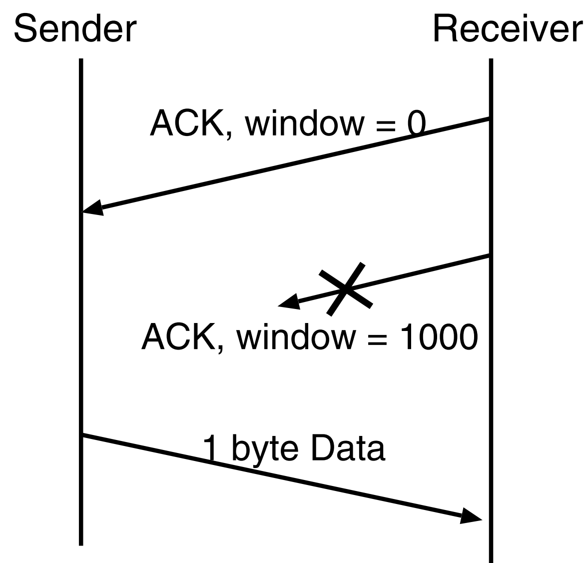
- αの推奨値 0.9

再送タイムアウト値の計算

- 昔のアルゴリズム
  - rto = 2 × srtt
  - ラウンドトリップタイムの急速な変動に弱い
- 新しいアルゴリズム
  - 平均偏差の採用
    - 標準偏差より計算が簡単
  - rto = 平均 rtt + 4 × 平均偏差
- 指数バックオフ
  - タイムアウトが起こる毎にタイムアウト値を２倍に
  - 最大値は 64秒

パーシストタイマ(持続タイマ)

- 送信ウインドウが 0になった場合のデッドロック防止
  - ウインドウの更新を伝えるACKが喪失するなど
- 持続タイマがexpireすると1バイトのデータを転送
  - 送信ウインドウが0でも1バイトのデータを送る
- タイムアウトした場合は指数バックオフ
  - 最大値は60秒



キープアライブタイマ

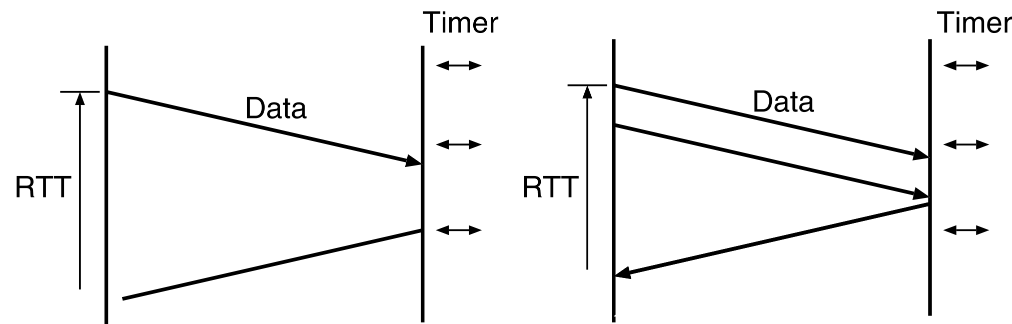
- 一定間隔で通信相手にパケットが到達できるか確認する
- 終了手順なしに通信相手が通信を終了した場合などに有効
  - システムのダウンなど
- ２時間毎にprobeする
- キープアライブ機能に関する議論
  - 帯域幅を不必要に消費する可能性
  - パケット単位の課金システムに対する問題
  - Webサーバなどでの利用

2MSLタイマ

- MSL(Max Segment Lifetime): 最大セグメント生存時間
  - パケットがネットワークに滞留できる最大時間
  - コネクションの終了時などに利用
    - コネクション終了の最後のパケットはACKしない
      - 2MSL待機する
  - RFC793の推奨値は 2分
  - 多くの実装では 30秒、 1分

遅延確認応答

- 受信側の送信遅延
- パケットを受け取ってもすぐにはACKを返さない
  - ACKを送信する条件
    - 次のパケットの到着する
    - ファーストタイマが起動される
  - 逆方向のデータ送信があればPiggyBackする
  - ACKの数を減らす
    - ACKによる輻輳の可能性を減らす
    - パケット受信割り込みの頻度を減らす

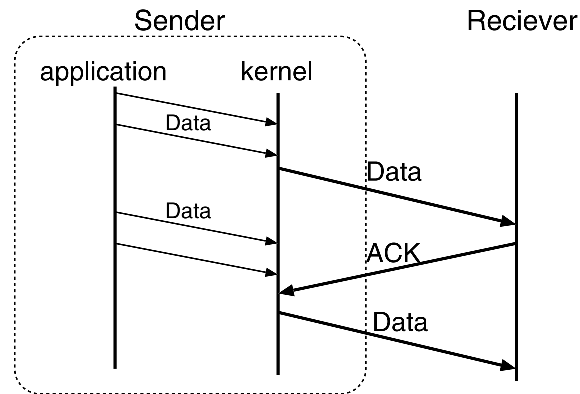


Nagleアルゴリズム (1)

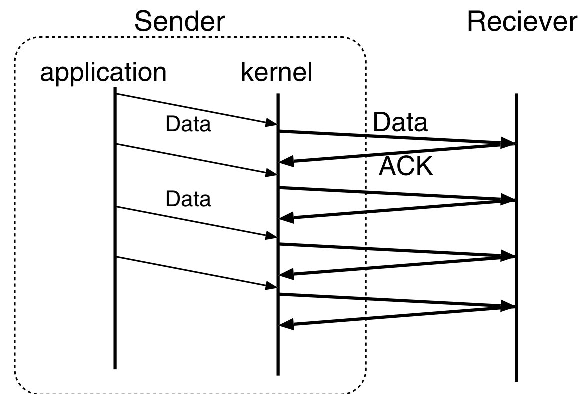
- 小さなパケットはできるだけまとめて転送する
  - アプリケーションからの転送要求を遅延させる
- telnetや rloginなどで有効
- 遅延させる条件
    - 送出した小パケットのACKを受信するまで次の小パケットを送信しない

Nagleアルゴリズム (2)

- ▹低速なネットワークではまとめて送信
  - ▵パケットヘッダのオーバヘッドを減らす



- ▹高速なネットワークでは素早く送信



シリーウインドウシンドローム

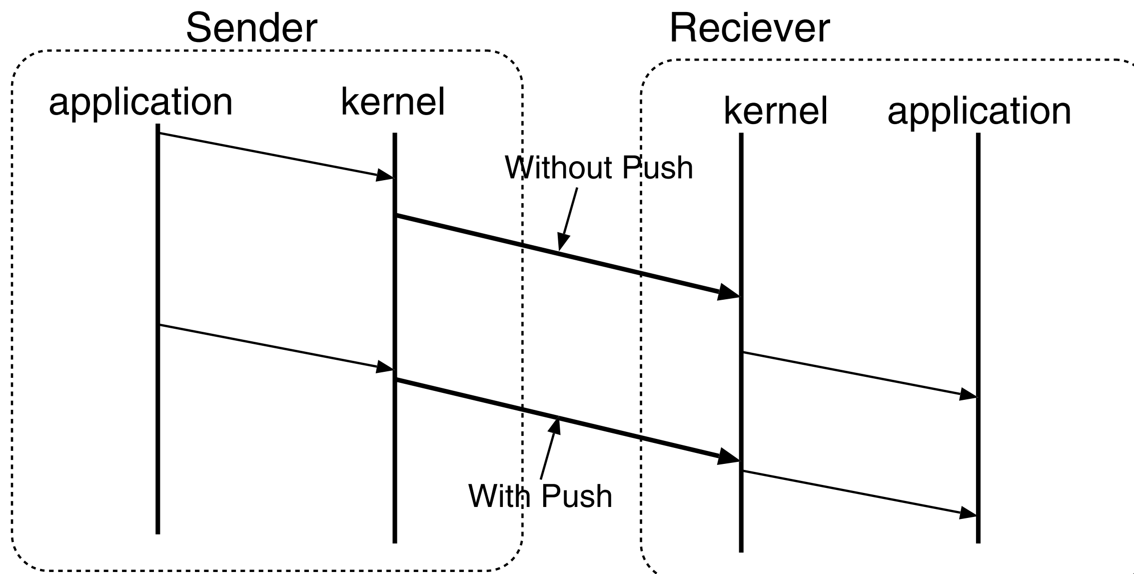
- 受信側が小さなウインドウサイズを通達することにより、小さなパケットがやりとりされる
  - 通信の効率が低下する
  - 回避方法(RFC813)
    - 受信側
      - ウインドウサイズを通達する条件
        - ウインドウサイズがフルサイズセグメント分空いている
        - ウインドウサイズの1/2が空いている
    - 送信側
      - パケットを送出できる条件
        - フルサイズセグメントを送出できる
        - 通達されたウインドウサイズの1/2を送出できる
        - 待ちパケットがない

TCPの通信フラグ

- TCPヘッダ中で指定
  - ６つの指定が可能
    - SYNフラグ:　コネクション確立時に使用
    - ACKフラグ:　パケットがACK情報を含んでいることを示す
    - FINフラグ:　コネクション終了時に使用
    - Pushフラグ
    - RSTフラグ
    - 緊急フラグ

Pushフラグ

- 受信者にデータを遅延なく転送したい場合に利用
  - 送信側のカーネルでのバッファリングの抑制
  - 受信側のカーネルでのバッファリングの抑制
    - 現在ではあまり意味がない
    - 受信したパケットは直ちにプロセスに転送される



RSTフラグ

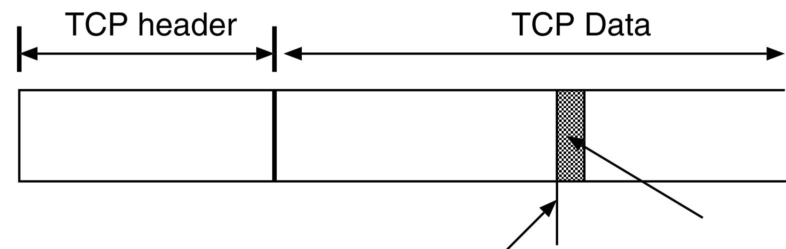
- TCPコネクションをリセットする
  - listenしていないportに対する接続要求を拒否する
  - コネクションを中断する
  - ハーフオープンのコネクションを終了する
    - RSTに対する確認応答は生成されない
      - RSTを送信すると同時にClose
      - RSTを受信すると同時にCloseまたはListen

緊急フラグ (1)

- 帯域外データを転送するために利用する
- 帯域外データとは？
  - アプリケーションに緊急にデータを渡す手段
    - データ転送を中断したい場合などに利用
  - 実際には通常のデータストリームと同様に配送される
    - 正確には帯域外ではない

緊急フラグ (2)

- 帯域外データの転送処理
  - 送信側の処理
    - 緊急フラグをセット
    - 送出データの中で緊急データの最後のバイトのシーケンス番号を緊急ポインタにセット
  - 受信側の処理
    - アプリケーションプロセスに緊急データの到着を通知
    - アプリケーションが緊急データまでのストリームを読みだすまで緊急モードと見做す



TCPと輻輳制御

- □輻輳制御の重要性
- □輻輳制御の困難さ
- □TCPによる輻輳制御アプローチ

輻輳制御の重要性

- 輻輳崩壊の危険性
  - 輻輳は悪化していく傾向がある
  - 適切な制御技術の必要性

- 輻輳制御の歴史
  - 初期のインターネット: 輻輳制御技術なし
  - 1980年初: 輻輳崩壊の発生が指摘される
  - 1980年後半: エンドノード主体による輻輳制御技術の出現
  - 1990年後半: 中継ノード主体による輻輳制御技術の出現

輻輳制御の難しさ (1)

- インターネットの状態はわかりにくい
  - IPの特徴
    - 様々な通信媒体の性質を抽象化
      - 上位プロトコルから通信媒体の性質が見えにくい
    - 中継システムの簡素化
      - 中継システムの機能が少ない
  - ネットワークの許容量がわからない
  - ネットワークの混雑度がわからない

輻輳制御の難しさ (2)

- インターネットは自律的
  - インターネット全体を制御する機構がない
    - ユーザの振るまいを統括的に制御できない

輻輳制御の難しさ (3)

- インターネットはモデル化しにくい
  - スケールが大きすぎる
  - 構成要素が多様、構成形態が多様
  - インターネットは変化する
    - 昨日のインターネットは今日のインターネットではない

TCPによる輻輳制御

- 終端ノードによる自律的な制御
- 簡易な通信経路の推測機構
  - 通信経路に適した転送レートの選択
    - 輻輳の発生を防ぐ
  - 輻輳の検出
    - 輻輳崩壊を防ぐ

TCP輻輳制御アルゴリズムの歴史

- 1988頃
  - Tahoe
    - スロースタート、輻輳回避アルゴリズムの採用
    - Fast Retransmitアルゴリズムの採用
- 1990頃
  - Reno
    - Fast Recoveryアルゴリズムの採用
      - 輻輳の度合が少ない場合、転送速度を大きく落さない
- 1996頃
  - NewReno
    - Fast Recoveryアルゴリズムの修正
      - パケットの喪失率がやや大きい場合に対するアルゴリズムの不具合の修正

TCPの輻輳制御アルゴリズム (1)

- □TCPの割り切り
  - ○ネットワークの状態はよくわからない
- □単純なアルゴリズムによる転送制御
  - ○パケットが喪失しない
    - ▹ネットワークは空いている→転送速度を上げる
  - ○パケットが喪失する
    - ▹ネットワークは混んでいる→転送速度を上げる

↓

  - ○パケット喪失が起きるまで転送速度を上げ続ける
    - ▹通信経路の限界をパケット喪失で調べる

TCPの輻輳制御アルゴリズム (2)

- 転送速度の制御
  - 輻輳ウインドウ(cwnd)のサイズを変える
    - min(輻輳ウインドウ、受信側が通知したウインドウ)で通信
- ２つの通信段階
  - 輻輳ウインドウの増加量が異なる
  - スロースタート
    - ACKを受け取る毎に１つづウインドウサイズを増加
    - 指数的にウインドウサイズが増加
  - 輻輳回避
    - ACKを受け取る毎に１つづウインドウサイズを増加
    - 線形にウインドウサイズが増加

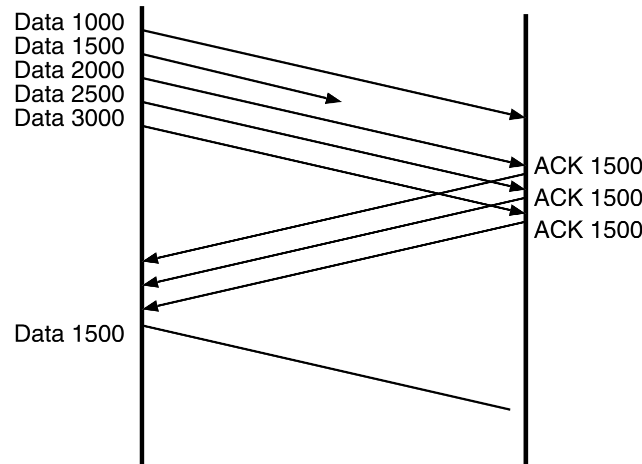
TCPの輻輳制御アルゴリズム (3)

- データの喪失が起きれば
  - 輻輳ウインドウサイズの1/2をssthreshとして登録
  - 輻輳ウインドウサイズを減らす(転送速度を下げる)
    - 重複確認応答による喪失の検出
      - 現在の仕様ではcwndを1/2にする
    - 再送タイムアウトによる喪失の検出
      - cwndを最小にする

- データの喪失が起きない場合
  - 輻輳ウインドウサイズを増やす(転送速度を上げる)
    - cwnd < ssthresh
      - スロースタート
    - cwnd > ssthresh

Fast Retransmitアルゴリズム

- 再送タイムアウトを待たずに再送パケットを転送する
  - 重複する確認応答が３つ来た場合、パケット喪失の可能性が高い
    - パケットは高い確率で順序通りに転送される
  - 迅速な再送による転送効率の向上
  - パケット再送後は、転送速度を減少させる
    - 輻輳崩壊を避ける
      - Tahoe → スロースタート
      - Reno → Fast Recovery



Fast Recoveryアルゴリズム(1)

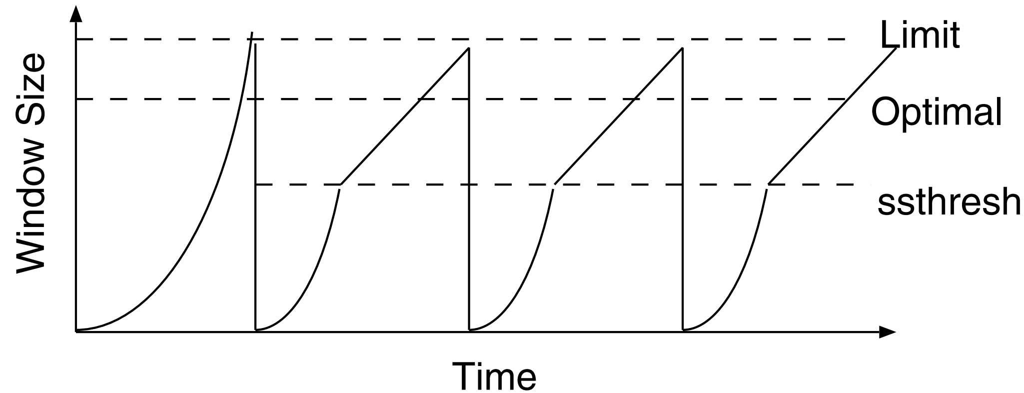
- Tahoeの輻輳回避アルゴリズムの問題
  - Fast Retransmit終了後に転送速度を落しすぎる
    - ウインドウサイズ=1
- Fast Recoveryの目的
  - Fast Retransmitが成功すれば輻輳は軽微だったと見做す
    - 転送速度をパケット喪失検出前の50%に落す

Fast Recoveryアルゴリズム (2)

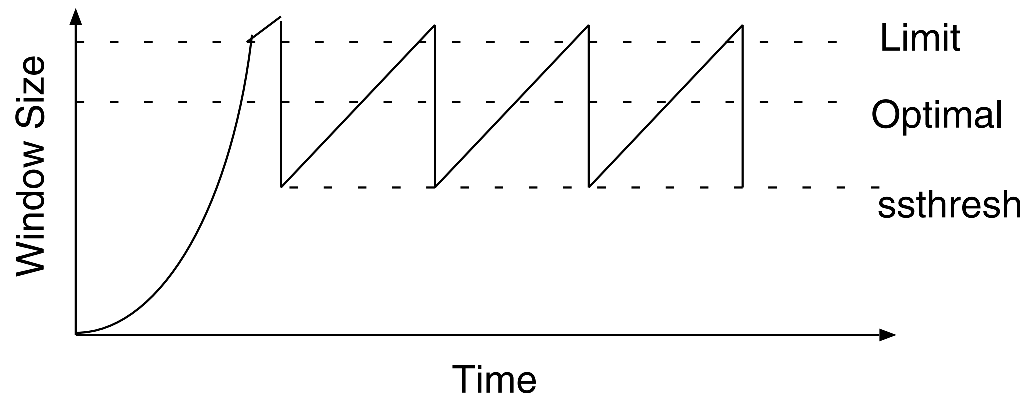
- パケット再送後の挙動
  - さらに重複再送パケットが到着する
    - さらにパケットが到着しているので、cwndを一時的に増加
      - 新しいセグメントを送出する
  - 新しいACKが到着する
    - cwndをssthreshにセットする(パケット喪失前の1/2)
    - スロースタートには入らず、いきなり輻輳回避段階へ

輻輳ウインドウの変化の例

□Tahoeによる輻輳ウインドウの変化



□Reno,NewRenoによる輻輳ウインドウの変化



TCPの輻輳制御アルゴリズムのねらい

- ネットワークの状態の変化に対応する
  - 輻輳が起これば転送速度を下げる
    - 輻輳崩壊を避ける
  - 輻輳が起こらない限り転送速度を増加させ続ける
    - ネットワークが空いた時にも適応
- 輻輳が起きないぎりぎりの転送速度で長く通信する
  - ssthresh < windowsize < limitの間で長く通信する
- 通信経路に適したウインドウサイズで通信するとセルフクロッキングが起きる

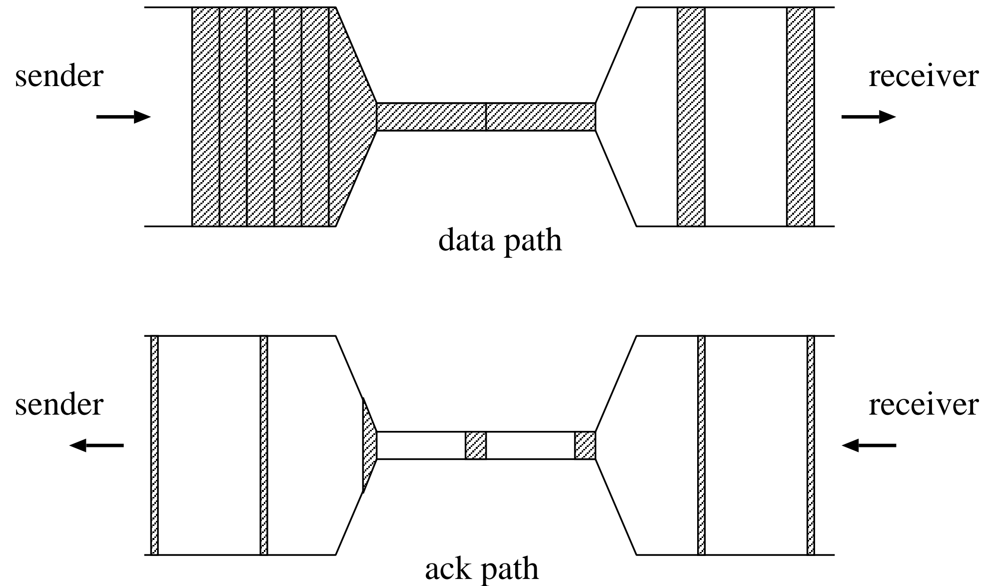
セルフクロッキング

□確認応答をデータパケットの送信のトリガにする

○送信パケットの間隔が経路中の一番細いリンクに合わされる

▹転送のバースト性が低くなる

▹複雑な転送レート制御機構がいらない

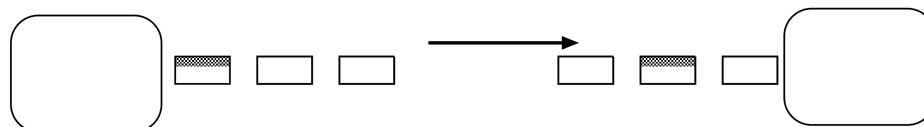


採用されつつある機能

- 広帯域、高遅延ネットワークへの対応
- Path MTU discovery
- SACK(選択確認応答)
- 現在の実装状況

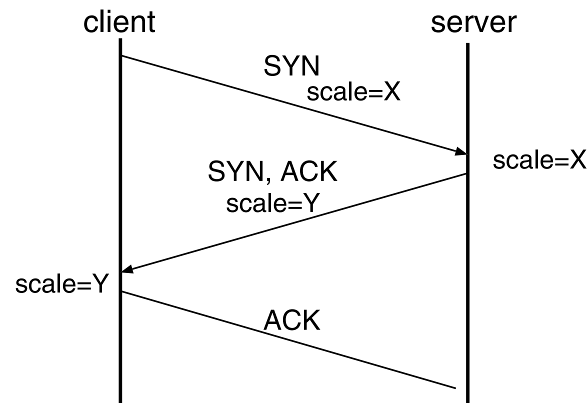
広帯域、高遅延ネットワークへの対応

- 広帯域、高遅延ネットワークにおけるTCPの問題
  - ウインドウサイズが足りなくなる
    - TCPの転送性能
      - ウインドウサイズ / RTT
    - RFC793では最大ウインドウサイズは 65535バイト
      - 帯域 2Mbps, RTT 0.5秒の回線では 512000バイト必要
        - 最大でも帯域の12%しか利用できない
  - RTTの計測が不正確になる
    - ウインドウサイズが増加 → 計測頻度の減少
  - シーケンス番号の周回
    - ネットワーク中にパケットが滞留している間に、同じシーケンス番号を持つパケットが送信されてしまう



ウインドウスケールオプション

- 大きなウインドウサイズを指定できる
  - ウインドウサイズの値をビットシフトする値を指定
    - スケール値= X
      - → TCPヘッダ中のウインドウサイズの値を X bitシフトして使用
      - → ２のX乗倍のウインドウサイズ
    - 最大シフト値 14
    - 最大のウインドウサイズ: 65535 × 2^14 = 1,073,725,440
  - 3way handshake中に指定する
    - コネクションの途中でスケール値は変更できない



タイムスタンプオプション

- RTTの計測頻度の向上
  - RTTの精度を向上
- シーケンス番号周回の防止
  - タイムスタンプ値とシーケンス番号のペアで周回を検出

  - 送信側
    - データパケットにタイムスタンプをつけて送信する
  - 受信側
    - 確認応答パケットにタイムスタンプの値を送り返す
  - 送信側で現在の時刻とタイムスタンプ値の差を計算する

Path MTU discovery

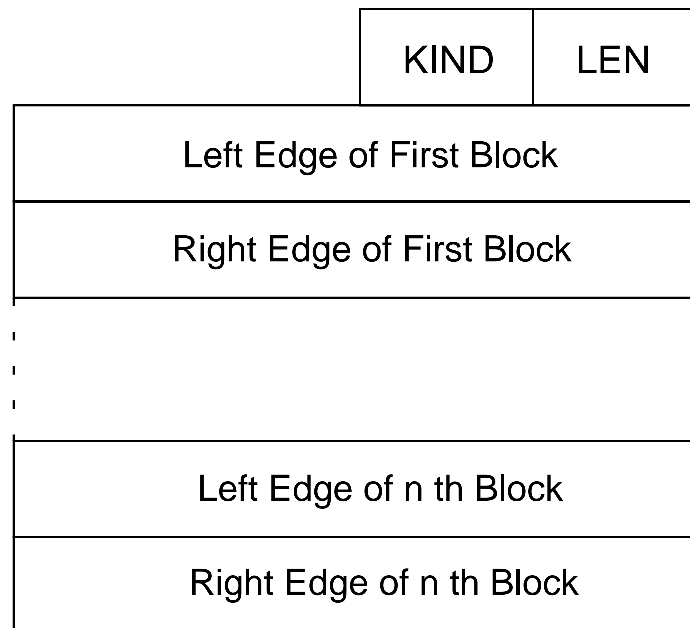
- すべてのデータパケットにDF(Don't Fragment)ビットをセット
- ICMPによるDF通知が返って来たら、MSSを再設定する
  - 再設定後はスロースタート
- MSSの有効期間
  - 推奨 10分 (RFC1191)
  - 相手側の通知する MSSとインターフェースのMTUを試す

SACK(Selective Acknowledgement)

- 選択確認応答オプション
  - どのパケットが到着したかを詳細に通知する
- RFC2018で規定
  - ２つのTCPオプションを追加する
  - SACK Permitted Option
    - SACK optionに対応していることを通知
    - 3way handshake時にnegotiate
      - SYN以外のパケットでは利用できない
  - SACK Option
    - 到着したパケットの情報を格納する

SACK option format

- 最大で4ブロック設定できる
  - 到着したシーケンス番号のブロックの左端と右端を通知



SACKの使用例

- 最初のブロックには最新の受信シーケンス番号を格納する
  - 受信側に到着した最大のシーケンス番号がわかりやすい
    - 例:
      - シーケンス番号5000〜8500のパケットをMSS 500バイトで送信
      - シーケンス番号5500,6500,7500のパケットがlost

Trigger Segemnt	ACK	1st block Left	1st block Right	2nd block Left	2nd block Right	3rd block Left	3rd block Right
5000	5500						
5500 (lost)							
6000	5500	6000	6500				
6500 (lost)							
7000	5500	7000	7500	6000	6500		
7500 (lost)							
8000	5500	8000	8500	7000	7500	6000	6500

現在の実装状況

- Pittuburgh Supercomuting centerの調査
  - http://www.psc.edu/networking/perf_tune.htmlより抜粋

OS	PMTU	RFC1323	SACK
Win95	YES	NO	NO
Win98	YES	YES	YES
WinNT3.5/4.0	YES	NO	NO
FreeBSD3.3	YES	YES	NO
SunOS4.1	NO	NO	NO
Solaris2.6	YES	YES	YES
Solaris7	YES	YES	YES

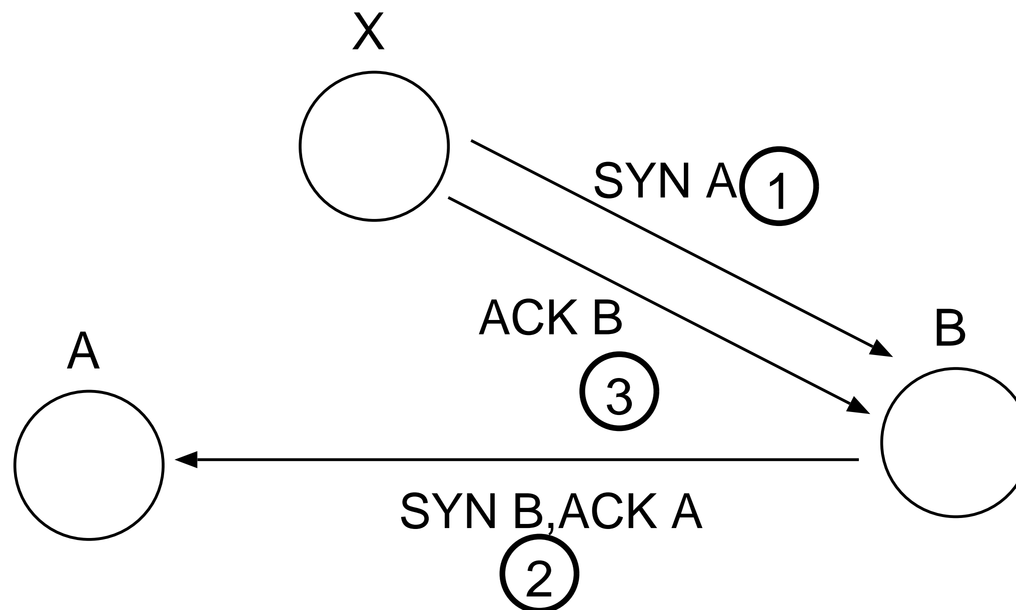
TCPとセキュリティ

- 過去のTCPに対する攻撃
  - Sequence number attack
  - SYN flood Attack

- 適切な処置
  - IPsec
  - 適切なfiltering

Sequence number attack (1)

- TCPコネクションに使われるSequence番号を推測する
- 偽造パケットで偽のTCPコネクションを確立する
  - コマンドなどを送り込むことが可能

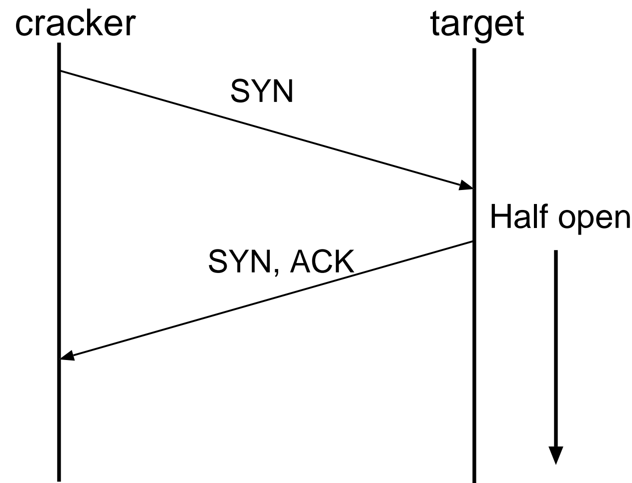


Sequence number attack (2)

- Sequence番号生成アルゴリズムの修正
  - 古い実装
    - 次の２つの変数からSequence番号を生成する
      - 1秒毎にカウンタを1増加
      - コネクション毎に固定値を加算
  - 推奨される実装
    - 次の２つの変数からSequence番号を生成する
      - 4マイクロ秒毎にカウンタを増加
      - src adr, src port, dst adr, dst portなどの情報をハッシュしたもの

SYN flood attack (1)

- サービス妨害(DoS)の一種
- targetに偽のSYNを送る
  - half openのTCP connectionを大量に作らせる
  - 新しいコネクションを受け付けられなくなる
    - コネクション要求キューの溢れ
    - メモリの消費



SYN flood attack (2)

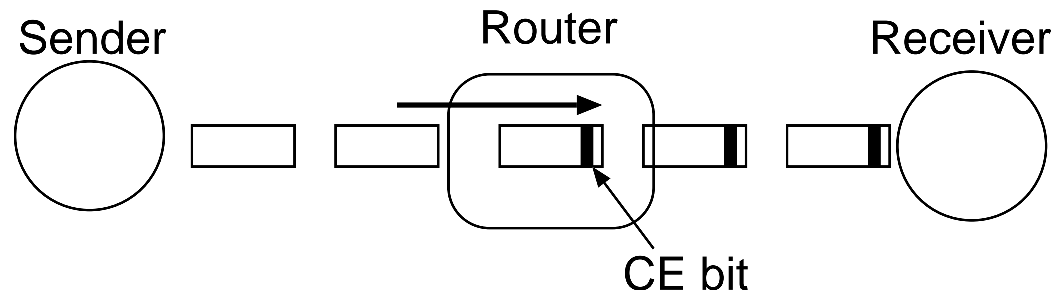
- 対策
  - 終端システムでの対策
    - 3way handshake時のtimeoutを短くする
    - コネクション要求を受け付けるqueueを大きくする
    - half openのconnectionに割り当てるメモリを減らす
  - 中継システムでの対策
    - 信頼できるアドレス以外からのアクセスを拒絶する
    - SYNパケットの転送レートを制限する

これからの技術動向

- □Explicit Congestion Notification
- □Initial Large Window
- □TCPVegas
- □NewReno
- □Rate-Halving
- □TCPfriendly

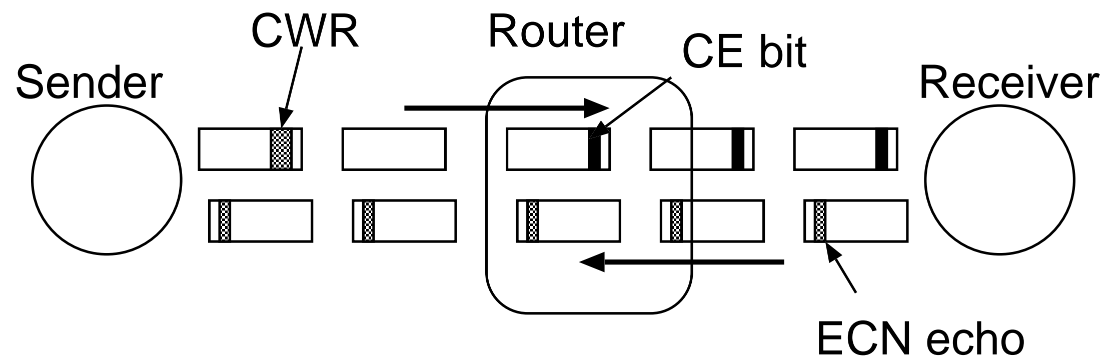
ECN (Explicit Congestion Notification)

- ECNとは？
  - 中継ルータが明示的に congestionの発生を知らせる
  - IPのTOS field中の2bitを利用する
    - RFC2481で規定
    - ECT(ECN Capable Transport)ビット
      - Transport層プロトコルがECNに対応していることを示す
    - CE(Congestion Experience)ビット
      - 輻輳が起きていることを示すビット



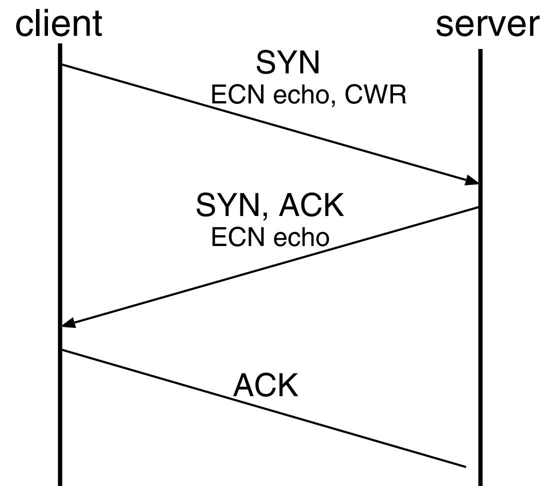
ECNとTCP (1)

- □TCP headerに2つのbitを追加する
  - ○RFC2481で規定
    - ▹Reserved field中の2bitを利用
    - ▹ECN echoフラグ
      - ▵輻輳が起こっていることを相手側に通知
    - ▹Congestion Window Reduceフラグ
      - ▵ECN echoに従って輻輳ウインドウを減少させたことを通知する



ECNとTCP (2)

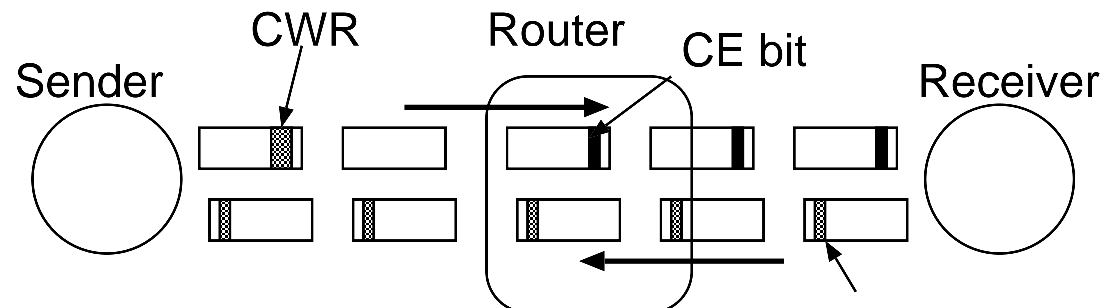
- Handshake時
  - 通信する両者がECNの使用について合意する
  - Sender:
    - ECN echoとCWRをセットし、SYNを送信
  - Receiver:
    - ECN echoビットだけセットし、SYN + ACKを送信
      - TCPの実装の中には senderの reserved fieldをそのままecho backするものがある



ECNとTCP (3)

□輻輳時の挙動

- ▹Receiver:
  - ▵CEビットのセットされたパケットを受信すると以後 ACKを転送する際に必ず ECN echoビットをセットして送信する
- ▹Sender:
  - ▵Packet Lossの場合と同様にウインドウを減少させる
  - ▵最初の再送パケットに CWRビットをセットして転送
- ▹Receiver:
  - ▵CWRビットのセットされたパケットを受信した後は、CEビットのセットされていないパケットのACKに ECN echoビットをセットしない



Large Initial Window (1)

- RFC2414で規定
- 輻輳ウインドウの最小値を大きくする
  - 従来は1 MSS
- 利点
  - 最初の転送に遅延確認応答が適用されない
  - 輻輳ウインドウの立上りが早い
    - 遅延の大きな回線で性能が向上する
  - 小さなファイルなら1 RTTで送れる
    - HTTPは細かいデータ転送が多い
- 欠点
  - 転送のバースト性が増加する可能性がある

Large Initial Window (2)

- 新しい初期値
  - min(4 * MSS, max(2 * MSS, 4380)
    - MSSが1095バイト以下なら: 4 * MSS
    - MSSが1095バイト以上2190バイト以下なら: 4380バイト
    - MSSが2190バイト以上なら: 2 * MSS

TCPVegas (1)

- TCPVegas
  - アリゾナ大のBrakmoらにより開発
- 従来のTCPの輻輳制御アルゴリズム
  - 単純な割り切り
    - パケット喪失が起こる→ネットワークが混んでいる
    - パケット喪失が起きない→ネットワークが空いている
  - パケットロスが起こるまで転送速度を増加させる
    - 輻輳を起こすことによって通信経路の限界を推測
- TCPVegasのねらい
  - より詳細な通信経路の状態の推測
    - 混み具合に応じた転送速度の調節

TCPVegas (2)

- TCPVegasの輻輳制御アルゴリズム
  - データ転送しながら２つのスループットを計測する
  - Actual Throughput
    - 実際の転送性能
  - Expected Throughput
    - これまでの転送の結果から推測した転送性能
  - 通信経路の状態の推測
    - Actual < expected
      - ネットワークが混んできた→転送速度を落す
    - Actual > expected
      - ネットワークが混んできた→転送速度をあげる
    - Actual = expected
      - ネットワークが安定している→転送速度を維持

NewReno

- RFC2582で規定
  - MITのHoeらのアイデアがベース
- TCPの新しい輻輳制御アルゴリズム
  - Fast Retransmit,Fast Recoveryアルゴリズムの変更
    - 1RTT内に複数のパケット喪失がある場合の転送性能の改善
    - Reno
      - Fast Retransmitの誤作動
      - 再送タイムアウト待ちによる転送性能の劣化
    - NewReno
      - Fast Retransmitの誤作動の防止
      - 再送タイムアウトを待たずに複数パケットを再送する
        - Renoよりややaggressiveなポリシー

Rate Halving

- MITのHoe, PSCのMathisらにより提案
- パケット喪失検出後の処理
  - 従来のFast recovery
    - 輻輳ウインドウを1/2にする
  - Rate halving
    - 再送段階
    - adjust段階
      - 2ACK毎に1データパケットを転送する
        - self-clockingを崩さずに転送速度を50%に
        - 転送のburst性を抑える

TCPfriendly

- □ACIRIのS.Floydらにより提案
- □TCPには輻輳制御があるがUDPにはない
  - ○回線が混むとUDPが帯域を占領してしまう
- □UDPのための輻輳制御の指標
  - ○TCPの通信モデルに合わせる
  - ○TCPの転送速度とパケットロス率の関係

$$bandwidth = \frac{1.3MTU}{RTT\sqrt{Loss}}$$

- □ルータなどでUDP flowがこの式に従っているかを判別する
  - ○輻輳崩壊の原因となるflowを強制的に排除

TCPの今後

- 標準化、実装の普及
  - SACK, ECN, (NewReno)
- 輻輳制御アルゴリズムの改善
  - Rate halving, Vegas
- 新たな技術との組合せ
  - CBQ, Diffserv
- エンドシステム全体の輻輳制御技術への発展
  - TCP friendly
  - Congestion Manager
    - TCPの輻輳制御機構の分離

For More Information

- TCPに関連する主なRFC
  - RFC793 .. 基本仕様
  - RFC813 .. Silly Window Syndrome
  - RFC1122 .. Host Requirement
  - RFC1323 .. Extention for high performance
  - RFC2414 .. Large Initial Window
  - RFC2418 .. ECN
  - RFC2581 .. Congestion Control
  - RFC2582 .. NewReno algorithm
- IETF WG
  - TCP Implementation (tcpimpl)
  - TCP Over Satellite (tcpsat)
  - Performance Implications of Link Characteristics (pilc)