■3群（コンピュータネットワーク）－4編（トランスポートサービス）

3章 TCP (Transmission Control Protocol) 性能評価

■概要■

■3群 － 4編 － 3章

3-1 TCP のモデル化と解析

（執筆者：鶴　正人）［2013 年 7 月 受領］

TCP は，エンド端末間での信頼性のあるバイトストリーム（以下，TCP フローと呼ぶ）を提供するために，誤り制御，輻輳制御，受信フロー制御などの機能をもっている（1 章参照）．輻輳制御は，ネットワークの輻輳や TCP フロー間の不公平性を回避しながら，通過するネットワーク経路上のリンクの可用帯域を最大限利用して高速にデータを転送するために，各 TCP フローの送信者が送信レートを自律的に調整する技術であり，ネットワーク資源をお互いに無関係な複数の TCP フロー間で効率良く共有し，個々の TCP フローのスループットを向上することを目指している．

本節では，この輻輳制御に関する TCP フローのモデル化や解析の研究例を紹介する．TCP の輻輳制御は様々に進化してきた（2 章参照）が，ここでは，受信者からの Acknowledgment（ACK）パケットに基づいて，送信者がウィンドウ長を制御し，それによって送信レートを調整することを，最低限の共通モデルとする．このとき，このフィードバックを特徴づけるのは，送信者側のウィンドウ長の制御方式，受信者側の ACK の返信制御方式，そして途中のネットワーク（ルータ，スイッチ）でのリンクごとのキューイングバッファの制御方式（Droptail やその他の Active Queue Management（AQM））である．キューイングバッファ制御と ACK 返信制御に依存して，送信者に対して暗黙的にネットワーク輻輳状態がフィードバックされ，そこから得られるパケットロスまたは遅延の情報に基づき，ウィンドウ制御が行われる．フィードバックに必要な送受信者間パケット往復分の遅延時間（Round-Trip-Time：RTT）も挙動に影響する．

- まず単一 TCP フローのウィンドウ長の時系列挙動のモデル化とそれに基づくスループット予測が研究され，輻輳状態通知の発生を外部入力として，基本的な Additive Increase Multiplicative Decrease（AIMD）型のウィンドウ制御を忠実にモデル化した離散パケットモデルによる方法 [1), 2)] が提案された．パケット長を無限小と仮定して送信レートやウィンドウ長を連続量として扱う流体近似を用いた，多様なロスモデルに対応する研究 [3)] や，制御理論を用いてフィードバック遅延のある閉ループをモデル化する研究 [4)] もある．
- また，複数 TCP フローがネットワーク資源を共有する場合の平衡（定常）状態を調べるために，複数 TCP フローのフィードバックに基づく大域的閉ループ系をモデル化し，大域的な平衡状態における，個々のフローのスループットやネットワーク状態に関する性質，更に平衡状態への収束・安定性に関する性質の解析手法が研究された [5)-7)]．
- 更に，それらの解析から得られた，平衡状態での複数 TCP フロー間の帯域共有・配分原理に基づき，有限長ファイルを転送する個々の TCP フローがランダムに開始・終了する動的な状況を確率過程（特に待ち行列理論）を用いてマクロにモデル化し，ファイル転送の集団レベルでの帯域共有の統計的性質を解析する研究があり，実用的にも重要である [8), 10)]．ただし，これに関しては，分かり易い日本語の解説記事があるのでそちらを参照されたい [9)]．

これらのモデル解析によって，提案・実装されている制御方式の性質・性能を様々な条件の下で予測したり，逆に，望ましい性質・性能をもつ新しい制御方式を設計したりすること

が可能になる．もちろん，簡単化・理想化がされているので，パケットレベルのシミュレーション実験や実機実験も必要である．特に，制御方式が複雑になるほど，また利用環境が多様になるほど，解析可能なモデルと現実との乖離は大きくなる．しかしながら，環境条件によらない性質と強く依存する性質とを区別したり，方式間の関係を定性的に予測したりするためには，モデル解析は重要である．また，フロー本数やリンク本数が膨大でパケットレベルのシミュレーションが計算量・記憶容量的に困難な場合にも流体近似は有効である．

3-1-1 単一フローの時系列挙動のモデル化

古典的な TCP では，3 重複 ACK とタイムアウトによってパケットロスを検知し，それを輻輳の暗黙的通知としてウィンドウ長を減少させ，パケットを再送する．そこで，パケットロス発生モデルを与え，ウィンドウ長の時系列挙動をモデル化し，TCP フローの時間平均スループットを予測することは，自然なアプローチである．ここでは，最も有名で実用的にも広く使われている離散パケットモデルによる研究 [1] を紹介する．近年の改良型も存在する [2]．

定常状態での TCP フローの送信レート（送信パケット数÷時間）を予測する式を導出しよう．簡単のため，パケットロスに関して 3 重複 ACK による通知（「TD（Triple Duplicated ACK）ロス通知」と呼ぶ）だけが発生し，タイムアウトは発生しない状況で説明する．**図 3･1**(左)のように，輻輳回避モードにおいてウィンドウ長が線形に増加し，TD ロス通知が発生し，ウィンドウ長が半分になる場合を考える．時間軸は 1 RTT 時間を単位（Round と呼ぶ）として進行し，各 Round において，そのときのウィンドウ長分のパケットを送信する．以下の仮定を置く．

(i) ウィンドウ長は他の制約（例えば受信フロー制御による広告ウィンドウ長）を受けず，また，いつでも送るべき送信パケットが存在する．

(ii) 各 Round 内では，一旦パケットロスが発生すると，「その Round 内のそれ以降の全パケットも連続的にロスする」．任意のパケットがこのバーストロスの先頭パケットになる確率を p と置く．

(iii) 各 Round の RTT は独立で期待値は一定．この値を r と置く．

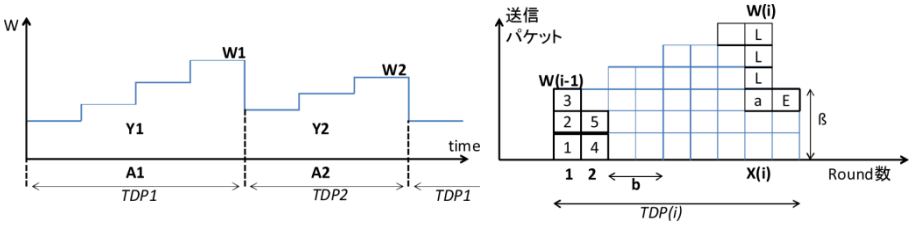


図 3・1　3 重複 ACK の発生とウィンドウ長変化の時系列（左：全体，右：1 回の 3 重複 ACK 発生）

二つの TD ロス通知間の期間を *TDP*（TD Period）と呼び，i 番目の TDP_i の間に送信されるパケット総数（途中でロスするものも含む）を Y_i，経過時間を A_i とすると，定常状態での TCP フローの送信レート B は $\dfrac{E[Y_i]}{E[A_i]}$ で近似できるので，これを求める．なお，Y_i を修正する

ことで，再送パケットを含まない厳密なスループット（グッドプット）の予測も可能である．

そのため，図 3･1(右)のように一つの TDP_i 内を詳しく見る．i 番目の TD ロス通知発生時，すなわち TDP_i の終了直前，のウィンドウ長を W_i とすると，TDP_i の開始時（最初の Round）のウィンドウ長は $W_{i-1}/2$ である．1 個の ACK で b 個のデータパケットの確認応答を兼ねる（Delayed ACK，古典的には，b = 2)．よって，Round ごとに（ウィンドウ長÷b)個の ACK が発生し，b Round ごとにウィンドウ長が 1 増える．ここで，TDP_i 内で先頭から $(\alpha_i + 1)$番目のパケットがロスし，その時点は X_i 番目の Round に含まれ，その Round 内では $(\beta_i + 1)$番目のパケットだとする．Round 内のそれ以降のパケットはすべてロスする（図の L 印)．Round 内でβ_i 番目までのパケットに対しては次の Round で ACK が返ってくる．よって，次の Round でβ_i 個のパケットを送信した時点で最初のロスに気付き（TD ロス通知が発生，図の E 印)，TDP_i が終了する．よって，$\{W_i\}$, $\{X_i\}$, $\{Y_i\}$の関係として，

$$W_i = \frac{W_{i-1}}{2} + \frac{X_i}{b},\quad Y_i = \alpha_i + (W_i - \beta_i) + \beta_i = \alpha_i + W_i,\quad Y_i = \frac{1}{2}X_i\left(\frac{W_{i-1}}{2} + W_i - 1\right) + \beta_i$$

が成り立つ．ここで，$\Pr[\alpha_i = k] = (1-p)^k p$ より，$E[\alpha] = 1/p - 1$．更に，β_i の一様分布を仮定すると，$E[\beta] = E[W]/2$.

これらの関係式を利用し，$\{W_i\}$と$\{X_i\}$を互いに独立と仮定して期待値をとると，未知数 $E[W]$, $E[X]$, $E[Y]$ が，p, b の式として推定できる．一方，TDP_i の j 番目の Round の RTT を r_{ij} と置くと，$E[r_{ij}] = r$ であり，

$$A_i = \sum_{j=1}^{X_i+1} r_{ij}, \qquad \text{より} \qquad E[A] = r\,(E[X] + 1)$$

以上より計算すると，

$$B(p, r) \approx \frac{E[Y]}{E[A]} = \frac{\dfrac{1-p}{p} + \dfrac{2+b}{3b} + \sqrt{\dfrac{8(1-p)}{3bp} + \left(\dfrac{2+b}{3b}\right)^2}}{r\left(\dfrac{2+b}{6} + \sqrt{\dfrac{2b(1-p)}{3p} + \left(\dfrac{2+b}{6}\right)^2} + 1\right)} \approx \frac{1}{r}\sqrt{\frac{3}{2bp}} \quad (p \ll 1) \qquad (3・1)$$

なお，タイムアウトの発生も考慮したより細かいモデル化に基づく定常状態送信レートの予測式（ただし高速再送機構は考慮しない）は，タイムアウト時間を T_0 と置くと，

$$B(p, r) \approx \left(r\sqrt{\frac{2bp}{3}} + T_0 \min\left(1, 3\sqrt{\frac{3bp}{8}}\right) p(1 + 32p^2)\right)^{-1} \quad (p \ll 1) \qquad (3・2)$$

3-1-2 フローレベルの大域的平衡点のモデル化

継続時間の長い複数の TCP フローが共存する状況では，その大域的な平衡状態での個々のフローの時間平均スループット，すなわちフロー間での帯域共有・配分を知ることが重要になる．そこで，TCP フローのフィードバックによる閉ループ系（の一部）を，流体近似を用いてモデル化し，与えられた多数の（固定本数の）フローが共存する閉ループ系の平衡状態における，個々のフローのスループット（または送信レート)，キュー遅延・キュー長，パケッ

トロス，更にフロー間の公平性，あるいは平衡状態への収束・安定性などを解析する研究が発展した．以下で 2 種類の研究を紹介するために，まず共通の前提と記号を定義する．

ネットワークを構成する M 本のリンクの集合 $\mathbf{M} = \{1, 2, \cdots, M\}$，ネットワーク上を流れる N 本の TCP フローの集合 $\mathbf{N} = \{1, 2, \cdots, N\}$，$(\mathbf{N}, \mathbf{M})$上のフロー経路行列 $\mathbf{A}$：$A_{ij} = 1$（フローi がリンク j を通る場合）；$= 0$（それ以外）は固定で変化しない．

- $d_i > 0$：フローi の往復伝搬遅延時間．$d = (d_1, d_2, \cdots, d_M)$.
- $x_i(t) \geq 0$ ：時刻 t でのフローi の送信レート．$x = (x_1, x_2, \cdots, x_N)$.
- $w_i(t) \geq 0$ ：時刻 t でのフローi のウィンドウ長．$w = (w_1, w_2, \cdots, w_N)$.
- $c_j > 0$：リンク j の帯域レート（容量）．$c = (c_1, c_2, \cdots, c_M)$.
- $y_i(t) \geq 0$ ：時刻 t でのリンク j を通過する集約フローレート．$y = (y_1, y_2, \cdots, y_M)$．$y_j(t) = (\mathbf{A}^{\mathrm{T}} x(t))_j$．$y_i(t) \leq c_j$ が前提条件.
- $q_i(t) \geq 0$ ：時刻 t でのリンク j 上での待ち時間（キューイング遅延）．$q = (q_1, q_2, \cdots, q_M)$.
- $\bar{d}_i(t)$：時刻 t でのフローi の往復遅延時間（RTT）．$\bar{d}_i(t) = d_i + (\mathrm{A}q(t))_i$（$d_i$ と経路上の各リンクの q_j の和）．

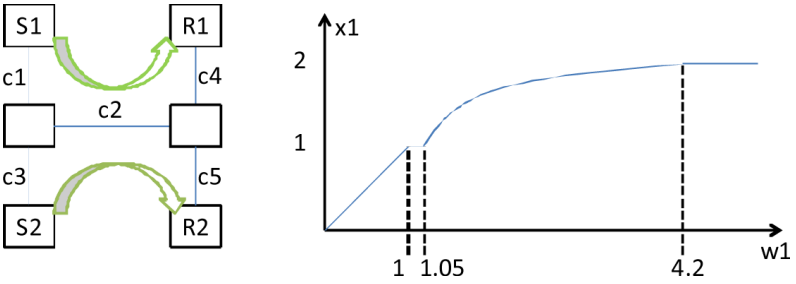
(1) 比例公平な送信レート配分とそれを実現するウィンドウ制御 [5)]

ウィンドウ長の制御に基づく任意の送信レート調整アルゴリズムを用いる複数フローがネットワーク上で共存する場合に，キューイングバッファは十分長く，パケットロスは無視できるとして，大域的な平衡状態（もし存在するなら）では以下が成立する（$i \in \mathbf{N}$）$j \in \mathbf{M}$）．

$$y_j - c_j \leq 0 \qquad (3・3)$$

$$q_j(y_j - c_j) = 0 \qquad (3・4)$$

$$x_i \bar{d}_i = w_i \qquad (3・5)$$



(a) トポロジーとフロー　　(b) ウィンドウ長 w_1 と送信レート x_1 の関係

図 3・2　2 フロー競合時の平衡状態

式(3･4)は，輻輳があるリンクではその帯域を使い切っていることを意味する．式(3･5)は，ウィンドウ長と送信レートの関係を規定する．$\bar{d}_i(t) = d_i + (\mathbf{A}q(t))_i$ なので，輻輳がない範囲（$q = \mathbf{0}$）では比例関係 $x_i = w_i / d_i$ にあるが，輻輳があると必ずしもそうではない．$\{\mathbf{A}, d, c\}$を固定

して，w, x, q を変数ベクトルとして扱うと，実は w を決めると x が一意に決まることが示される（q は必ずしも一意ではない）．**図 3･2** の例では，図(a)のトポロジー上のリンク帯域（$c_1 = c_3 = 2, c_2 = c_4 = c_5 = 3$），フロー1（$S_1 \rightarrow R_1$），フロー2（$S_2 \rightarrow R_2$）を与えたときに（$d_1 = d_2 = 1$），フロー2 のウィンドウ長を $w_2 = 2.1$ に固定すると，平衡状態でのフロー1 のウィンドウ長 w_1 と送信レート x_1 の関係は(b)のグラフになる．さて，

- 送信レートベクトル x（ただし $\mathbf{A}^{\mathrm{T}}x \le c$ ）はフローi へ送信レート x_i を分配する．複数フロー間の平衡状態における理想の送信レート分配として，比例公平（Propotional Fair）を定義する．「分配 x^*が比例公平」とは，任意の可能な分配 x に対して，

$$\sum_{i=1}^{N} \frac{x_i - x_i^*}{x_i^*} \le 0$$

  が成立すること．定義より，比例公平分配 x^*は，もし存在するなら一意である．
- なお，平衡状態での可能な分配 x の全体集合（ $\mathbf{A}^{\mathrm{T}}x \le c$ ）の凸性から，比例公平分配 x^* は存在し，以下の最大化問題の唯一の解に等しいことが示される．

$$\max_{x \ge \mathbf{0}} \sum_{i=1}^{N} \log x_i \qquad \text{subject to } \mathbf{A}^{\mathrm{T}}x \le c$$

- 一方， $x_i(t)q_j(t) = c_j q_j(t) \cdot \frac{x_i(t)}{c_j}$ は輻輳しているリンク j 上のフローi の滞留パケット量を意味する（通過するフローごとの滞留量がそのフローの送信レートに比例すると仮定して）ので，$w_i(t) - x_i(t)\, d_i = x_i\, (\mathbf{A}q(t))_i$ をフローi の経路上滞留量と定義し，平衡状態における全フローで共通の経路上滞留量の目標値$\sigma > 0$ を定めると，実は，すべてのフローにおいてそれを達成する，つまり，$w_i - x_i d_i - \sigma = 0$ となるようなウィンドウ長割り当て $w = w^*$が一意に存在し，そのときの送信レート配分 $x = x^*$が比例公平になる．

これに基づき，次の微分方程式が与える $w(t)$は，適切な初期値の下で，w^* に収束することが示される．κは正の定数．

$$\frac{d}{dt} w_i(t) = \kappa \left(\frac{\sigma}{w_i(t)} + \frac{d_i}{\bar{d}_i(t)} - 1 \right) \frac{d_i}{\bar{d}_i(t)} \qquad (3 \cdot 6)$$

上の微分方程式に基づく擬似ウィンドウ制御方式では，定数σとκを事前に定めて，各フローi の送信者が，観測した RTT（$\bar{d}_i(t)$）の増減に基づいてウィンドウ長 $w_i(t)$を自律的に制御する．d_iは観測された最小の RTT で代用する．全フローがこの方式に従うならば，送信レートが比例公平配分 x^*である平衡状態に近づく．ただし，モデル上では送信者は$\bar{d}_i(t)$を瞬時に得るが，実際には受信者からの ACK の観測によるので，その時間分遅れ，収束性・安定性に影響する可能性がある．比例公平を拡張した $(\mathbf{p}, \alpha)$-比例公平と呼ばれるものに関しても，類似の議論が展開できる．

(2) 送信レート制御と AQM を含む平衡状態モデルの一般化と適用 [6), 7)]

TCP の輻輳制御は，送信者のウィンドウ長制御方式とネットワークのキューイングバッファ制御方式の組合せで挙動が決まり，前項の擬似方式では，フローi が通過する各リンク j

上での待ち q_j の和 $(\mathbf{A}q(t))_i = \bar{d}_i(t) - d_i$ が輻輳状態通知に相当した．

以下で紹介するものは，逆に，先に具体的な制御方式が与えられたときに，その性質を調べるためのモデル化であり，特にキューイングバッファ制御が AQM を用いる場合にも適用できる．一旦ウィンドウ制御を離れて，一般の送信レート制御とキューイングバッファ制御の各々のアルゴリズムによる状態変数の更新を流体近似モデル上の関数で記述する．その関数系よる状態変数更新に平衡点（不動点）が存在すると仮定すると，

- 各フロー i の送信者とネットワーク内の各リンク j が各々の関数を逐次的・分散的に実行して状態変数を更新すると，それらの値は平衡点に収束し，
- その平衡点が唯一の解になるような主問題・双対問題（送信レートを主変数，輻輳状態通知を双対変数とする）の利得関数が導出できる．

つまり，与えられた制御方式が適切な条件を満たせば，それに対して，適切な利得関数を構成し，その主問題・双対問題を調べることで，平衡状態での送信レート配分を予測できる．そのため，キューイングバッファ制御に関する一般化した変数を追加する．

- $p_j(t) \geq 0$ ：時刻 t でのリンク j 上での輻輳状態通知．大きな値は高い輻輳を意味する．$p = (p_1, p_2, \cdots, p_M)$.
- $s_i(t) \geq 0$ ：フロー j の送信者が利用可能な輻輳状態通知．$s_j(t) = (\mathbf{A}p(t))_i$. $s = (s_1, s_2, \cdots, s_N)$. つまり，通過する各リンクで発生した輻輳状態通知の「和」が送信者に伝わる，と仮定している．「キューイング遅延」や「値が十分小さい範囲でのパケットロス率」は適合する．
- $v_j(t)$：リンク j での AQM 制御のための内部状態変数（一般にはベクトル変数）．

さて，送信レート制御とキューイングバッファ制御の一般形を，

$$x_i(t+1) = F_i(x_i(t), s_i(t)) \qquad (3・7)$$

$$p_j(t+1) = G_j(y_j(t), p_j(t), v_j(t)), \qquad v_j(t+1) = H_j(y_j(t), p_j(t), v_j(t)) \qquad (3・8)$$

と書く（$i \in \mathbf{N}, j \in \mathbf{M}$）．$y_j(t)$ は $\{x_i(t) \mid i = 1, \cdots, N\}$ で，$s_i(t)$ は $\{p_j(t) \mid i = 1, \cdots, M\}$ で記述できる．F_i, G_j は非負関数とし，式(3･7), (3･8)の系の平衡点の存在を仮定し，(x^*, p^*) と置く．F_i, G_j, H_j に以下の性質を仮定する．

- x_i^* の近傍で，$x_i = F_i(x_i, f_i(x_i))$ が成り立つような 1 変数関数 $s_i = f_i(x_i)$ が存在し，かつその $f_i(x_i)$ は真に減少．つまり，F_i は，平衡状態の近くでは輻輳が高いほど送信レートを下げる制御になっている．
- $p_j = G_j(y_j, p_j, v_j), v_j = H_j(y_j, p_j, v_j), p_j > 0$ なる点 (y_j, p_j, v_j) があるとき，$y_j = c_j$．つまり，G_j, H_j は，平衡状態では輻輳状態が発生しているリンクの帯域は使い切る制御になっている．

このとき，フロー i に対応する利得関数を $U_i(x_i) = \int f_i(x_i)dx_i$ で定義し，主問題・双対問題

$$\max_{x \geq 0} \sum_{i=1}^{N} U_i(x_i) \quad \text{subject to } \mathbf{A}^{\mathrm{T}}x \leq c, \qquad \text{及び} \qquad \min_{p \geq 0} \left(\sum_{i=1}^{N} \max_{x_i \geq 0} (U_i(x_i) - x_i(\mathbf{A}p)_i) + \sum_{j=1}^{M} p_j c_j \right)$$

を考えると，実は左の主問題の唯一解が x^*，右の双対問題の唯一解が p^* になる．

例えば，FAST TCP でパケットロスがない場合の時間挙動（スロースタートは含まない）を考える．$\sigma > 0$, $\gamma \in (0, 1]$ を与え，また，輻輳通知 $p_j(t)$ としてリンク j での待ち時間 $q_j(t)$ を

用い，$x_i(t) = w_i(t)/(d_i + (\mathbf{A}q(t))_i)$ で略記すると，挙動を規定する関数系は以下になる．

$$w_i(t+1) = \gamma(x_i(t)\, d_i + \sigma) + (1-\gamma)w_i(t) \qquad (3・9)$$

$$\mathbf{A}^{\mathbf{T}}x(t) \le c\,, \quad \text{かつ} \quad q_j(t)((\mathbf{A}^{\mathbf{T}}x(t))_j - c_j) = 0 \qquad (3・10)$$

その平衡点を(w^*, q^*)とし，$x^*_i = w^*_i/(d_i + (\mathrm{A}q^*)_i)$ と置くと，対応する主問題・双対問題は，

$$\max_{x\ge\mathbf{0}} \sum_{i=1}^{N} \log x_i \quad \text{subject to } \mathbf{A}^{\mathbf{T}}x \le c, \quad \text{及び} \quad \min_{q\ge\mathbf{0}} \left(-\sigma \sum_{i=1}^{N} \log(\mathbf{A}p)_i + \sum_{j=1}^{M} q_j c_j \right)$$

となり，左の解が x^*，右の解が q^*と一致し，比例公平な送信レートが実現できる．

■参考文献

1) J. Padhye, V. Firoiu, D. Towsley, and J. Kurose, “Modeling TCP Reno performance: A simple model and its empirical validation,” IEEE/ACM Trans. Netw., 8(2):133-145, 2000.
2) N. Parvez, A. Mahanti, and C. Williamson, “An analytic throughput model for TCP NewReno,” IEEE/ACM Trans. Netw., 18(2):448-461, 2010.
3) E. Altman, K. Avrachenkov, and C. Barakat, “A Stochastic Model of TCP/IP With Stationary Random Losses,” IEEE/ACM Trans. Netw., 13(2), 2005.
4) C. V. Hollot, V. Misra, D. Towsley, and W. Gong, “Analysis and Design of Controllers for AQM Routers Supporting TCP Flows,” IEEE Trans. Auto. Control, 47(6):945-959, 2002.
5) J. Mo and J. Walrand, “Fair end-to-end window-based congestion control,” IEEE/ACM Trans. Netw., 8(5):556-567, 2000.
6) S. H. Low, “A Duality Model of TCP and Queue Management Algorithms,” IEEE/ACM Trans. Netw., 11(4):525-536, 2003.
7) D. X. Wei, C. Jin, S. H. Low, and S. Hegde, “FAST TCP: Motivation, Architecture, Algorithms, Performance,” IEEE/ACM Trans. Netw., 14(6):1246-1259, 2006.
8) S. Ben Fredj, T. Bonald, A. Proutiere, G. Regnie, and J.W. Roberts, “Statistical Bandwidth Sharing: A Study of Congestion at Flow Level,” In Proc. ACM SIGCOMM 2001, pp.111-122, 2001.
9) 石橋圭介, 川原亮一, “TCP フロー制御における帯域共有のモデル,” オペレーションズ・リサーチ(経営の科学), 49(7):438-442, 2004.
10) A. Lakshmikantha, C. Beck, and R. Srikant, “Impact of File Arrivals and Departures on Buffer Sizing in Core Routers,” IEEE/ACM Trans. Netw., 19(2):347-358, 2011.

■3 群 － 4 編 － 3 章

3-2 シミュレーションソフトウェアによる TCP 性能評価

（執筆者：長谷川　剛）［2013 年 6 月 受領］

TCP などのトランスポート層プロトコルの性能評価にあたっては，その実装コストの大きさなどにより，実機での性能評価は比較的困難である．また，その挙動が複雑であるため，数学的解析手法などを用いた性能評価においては，ネットワーク規模の大きさや TCP の挙動の複雑さと解析の複雑さがトレードオフとなるため，実ネットワークに近い環境における解析は極めて困難である．そのため，シミュレーション技術による性能評価が盛んに行われている．

本節では，フリーのネットワークシミュレータ ns-2 を用いてトランスポート層プロトコルのシミュレーションを行う方法について解説する．具体的には，ソフトウェアのインストール，シナリオファイルの記述，単純なネットワークモデルを用いたシミュレーションの実行，ログの取得方法などの基本的事項，更に TCP の輻輳制御方式の改造方法について説明する．

3-2-1 ネットワークシミュレータ ns-2

ns-2 [1] は，TCP/IP ネットワークのソフトウェアシミュレーションソフトウェアとして最も著名なものであり，トランスポート層プロトコル評価のためのソフトウェアとしても，盛んに用いられているものである．ns-2 はフリーソフトウェアであることもあり，全世界で用いているユーザ数が多く，メイリングリストなどのコミュニティが充実している．ns-2 は Linux などの UNIX（あるいは UNIX ライクな）OS だけではなく，Windows や MacOS X 環境でも用いることが可能である．

ns2 及びシミュレーション結果のアニメーション表示ツールである nam の実行には，Tcl/Tk，otcl，TclCl，xgraph などの外部パッケージが必要となる．これらは個別にインストールすることも可能だが，別途用意されている All-in-one パッケージを利用することも可能である．また，ns-2 はプロトコルの動作を変更すると，シミュレーションプログラムのバイナリーそのものが書き変えられるため，計算サーバ上などのマルチユーザ環境においては，コンパイルツリーをユーザごとにもつなどの工夫が必要となる．

次節以降では，具体的なネットワークモデルを想定し，シミュレーションのシナリオ設定，シミュレーション実行，シミュレーション結果の出力，及びグラフ化といった一連の流れについて説明する．

3-2-2 シミュレーション方法

(1) シミュレーション対象

本節では，**図 3･3** に示す簡単なネットワークを用いたシミュレーションを行い，TCP コネクション及びボトルネックルータの状態をログとして取得し，グラフ表示することを想定し，シミュレーション設定方法を解説する．

対象とするネットワークはボトルネックリンクが 1 本存在するダンベル型のネットワークであり，ボトルネックリンクの帯域は 10 Mbps，片道伝播遅延時間は 50 msec とする．また，4 台の端末（TCP Sender 1 及び 2，TCP Receiver 1 及び 2）が接続されており，アクセス回線

の帯域及び片道伝播遅延時間はすべて 100 Mbps 及び 5 msec とする．Router 1 に接続されたボトルネックリンクの出力バッファサイズは 100 パケットとする．TCP コネクションは TCP Sender 1 と TCP Receiver 1 の間（TCP コネクション 1），及び TCP Sender 2 と TCP Receiver 2 の間（TCP コネクション 2）に設定され，2 本の TCP コネクションがボトルネックリンクを共有する. TCP コネクション 1 はシミュレーション開始直後に TCP Sender 1 から TCP Receiver 1 へ，TCP コネクション 2 はシミュレーション開始 30 秒後に TCP Sender 2 から TCP Receiver 2 へそれぞれデータ転送を開始する．データ転送は FTP によって行い，転送データは無限に存在する．シミュレーション時間は 100 秒とする．

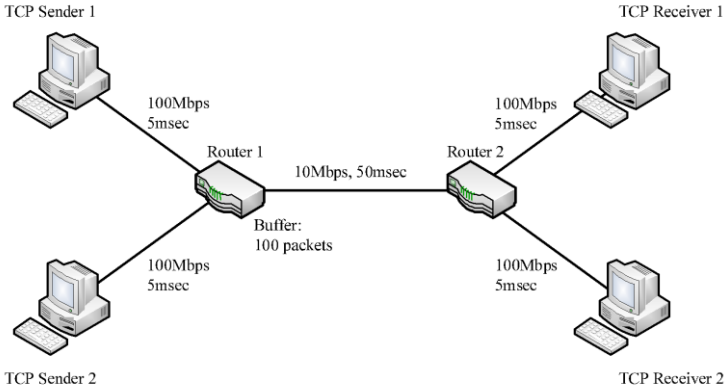


図 3・3　ネットワークモデル

(2)　シナリオファイル

図 3･4 に，図 3･3 のネットワークを ns-2 を用いてシミュレートする際のネットワークトポロジーを示す．ns-2 においては通常，端末及びルータはすべてノードとして扱われる．ノード n0，n1，n2，n3，n5，n6 は，TCP Sender 1，TCP Sender 2，TCP Receiver 1，TCP Receiver 2，Router 1，Router 2 にそれぞれ対応する．

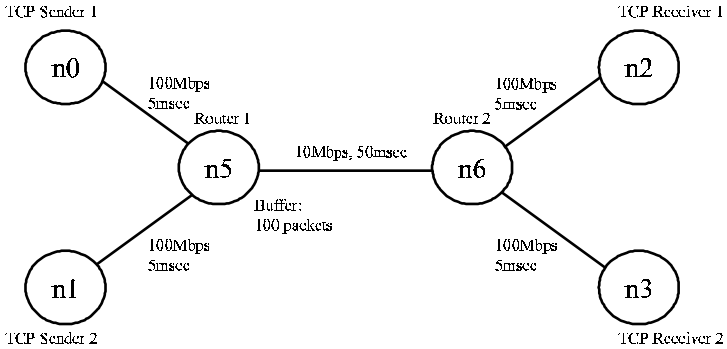


図 3・4　シミュレータにおけるトポロジー

ns-2 によるシミュレーションを行うためには, シナリオを記述したファイルが必要となる. シナリオファイルは Tcl スクリプトとして記述する. 以下に, 前節で説明したシミュレーションを実行するシナリオファイル 2tcp.tcl を示す.

```
1: # create simulator
2: set ns [new Simulator]

3: # for TCP
4: Agent/TCP set packetsize_ 1000
5: Agent/TCP set window_ 1000000

6: # create nodes
7: set n0 [$ns node]
8: set n1 [$ns node]
9: set n2 [$ns node]
10: set n3 [$ns node]
11: set n5 [$ns node]
12: set n6 [$ns node]

13: # create links
14: ## bottleneck link between n5 and n6
15: $ns duplex-link $n5 $n6 10Mb 5ms DropTail
16: $ns queue-limit $n5 $n6 100
17: ## access link between n0 and n5
18: $ns duplex-link $n0 $n5 100Mb 5ms DropTail
19: $ns queue-limit $n0 $n5 10000
20: ## access link between n1 and n5
21: $ns duplex-link $n1 $n5 100Mb 5ms DropTail
22: $ns queue-limit $n1 $n5 100000
23: ## access link between n6 and n2
24: $ns duplex-link $n2 $n6 100Mb 5ms DropTail
25: $ns queue-limit $n2 $n6 100000
26: ## access link between n6 and n3
27: $ns duplex-link $n3 $n6 100Mb 5ms DropTail
28: $ns queue-limit $n3 $n6 100000

29: # create TCP sender agents
30: set tcp_s1 [new Agent/TCP/Reno]
31: set tcp_s2 [new Agent/TCP/Reno]
32: # create TCP receiver agents
33: set tcp_r1 [new Agent/TCPSink]
34: set tcp_r2 [new Agent/TCPSink]

35: # attach TCP agents to nodes
36: $ns attach-agent $n0 $tcp_s1
```

```
37: $ns attach-agent $n1 $tcp_s2
38: $ns attach-agent $n2 $tcp_r1
39: $ns attach-agent $n3 $tcp_r2

40: # set TCP connections
41: $ns connect $tcp_s1 $tcp_r1
42: $ns connect $tcp_s2 $tcp_r2

43: # create FTP applications
44: set ftp1 [new Application/FTP]
45: set ftp2 [new Application/FTP]

46: # attach FTP agents to TCP connections
47: $ftp1 attach-agent $tcp_s1
48: $ftp2 attach-agent $tcp_s2

49: # set start time of FTP transmission
50: $ns at 0 "$ftp1 start"
51: $ns at 30 "$ftp2 start"

52: # finish setting
53: $ns at 100 "exit 0"

54: $ns run
55: exit
```

以下，各部について説明する．

1-2 行 : シミュレーション全体を統轄するための Simulator クラスのインスタンスの生成
3-5 行：TCP に関連する初期設定
6-12 行：シミュレーションで用いるノードの生成
13-28 行：ノード間リンクのパラメータ設定（リンク帯域，伝播遅延時間，及びリンクへの出力バッファサイズ(パケット単位)）
29-34 行：TCP 送信側と受信側のための Agent の生成
35-39 行：生成した Agent のノードへの設置
40-42 行：TCP コネクションの設定
43-45 行：FTP アプリケーションの生成
46-48 行：FTP アプリケーションの TCP コネクションへの関連付け
49-51 行：FTP によるデータ転送時刻開始の設定
52-53 行：シミュレーション停止（100 秒後）の設定
54-55 行：シミュレーションの実行及び終了の設定

なお，シナリオ内で使用しているエージェントである Agent/TCP/Reno は TCP Reno を用いる場合に指定するものである．TCP Tahoe を用いる場合には Agent/TCP を用いる．また，特

に指定しない場合には，Delayed ACK 及び SACK オプションは無効になっている．

本シナリオでは，簡易的な動作をする TCP のエージェントを利用しており，3-way handshake や ACK パケットによる広告ウィンドウサイズの通知などが行われない．これらの動作をシミュレートするためには，full TCP エージェントを用いる必要がある．

(3) シミュレーションの実行

シミュレーションは下記コマンドにより実行する．

```
% ./ns 2tcp.tcl
```

2tcp.tcl はログ出力を行わないため，シナリオファイルに誤りがなければ何も出力されずにシミュレーションが終了する．

(4) ログの出力

本節では，シミュレーション結果を確認するためのログ出力の方法として，ns-2 が用意しているトレースを用いる方法，及び Tcl を用いる方法を紹介する．ここでは，TCP コネクション 1 の輻輳ウィンドウサイズ（Congestion Window）の変化を出力することを想定する．

トレースを用いる方法を用いるためには，下記をシナリオファイルへ追記する．

```
# Filename for trace output
set tcpf [open tcp.tr w]
# Trace setting
Agent/TCP set trace_all_oneline_ true
# Trace congestion window for TCP connection 1
$tcp_s1 trace cwnd_
# Set trace
$tcp_s1 attach-trace $tcpf
```

これにより，シミュレーション終了後に tcp.tr という名前のファイルが生成される．その内容を下記に示す．

```
2.34049   0   0   2   0   cwnd_ 87.500
2.34132   0   0   2   0   cwnd_ 87.511
2.34216   0   0   2   0   cwnd_ 87.523
2.34299   0   0   2   0   cwnd_ 87.534
2.34382   0   0   2   0   cwnd_ 87.546
```

詳細は省略するが，各行の 1 列目がシミュレーション時刻，最終列が輻輳ウィンドウサイズであり，トレースしている変数の値に変化があるたびに記録される．そのため，変数値のすべての変化を知ることができる反面，シミュレーション時間が長い場合には，ログサイズが膨大になる．

Tcl を利用したログ出力を行う場合には，下記をシナリオファイルへ追記する．これは，シミュレーション開始後 0.1 秒から 100 秒までの間，0.1 秒間隔で TCP コネクション 1 の輻

輻輳ウィンドウサイズを出力する．

```
# logging scheduling
for {set i 0.1} {$i <= 100} {set i [expr $i + 0.1]} {
        $ns at $i "logging [expr $i]"
}

# logging function
proc logging { now } {
    global tcp_s1
    puts -nonewline $now
    puts -nonewline " "
    set temp [$tcp_s1 set cwnd_]
    puts $temp
}
```

これにより，下記のような出力が標準出力に得られる．

```
42.000000000000327 60.3817 51.1093
42.100000000000328 61.514 52.0978
42.20000000000033 62.6258 53.0868
42.300000000000331 63.6868 54.0761
42.400000000000333 64.7149 55.0658
```

この方法は，一定間隔でログを出力するため，変数値のすべての変化を知ることはできない反面，シミュレーション時間や注目したい時間の粒度に応じて出力間隔を設定することができる．

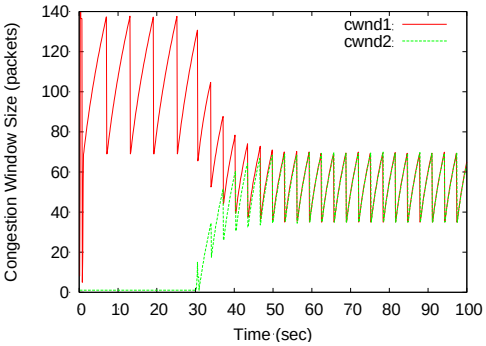


図 3・5　ウィンドウサイズの変化

図 3･5 に，本シミュレーションによって得られた，2 本の TCP コネクションの輻輳ウィンドウサイズの変化を示す．図から，TCP コネクションがウィンドウサイズをのこぎり状に変

動させていることがわかる．

(5) キューイベントの出力

本シミュレーションにおいては，Router 1 に接続されたボトルネックリンクの出力バッファにパケットが蓄積され，パケット廃棄が発生する．そこで，下記をシナリオファイルへ追記することで，キューへのパケットの出入りのイベントのログを取得する．

```
# Queue Trace
set queue [open queue.tr w]
$ns trace-queue $n5 $n6 $queue
```

これにより，下記のようなログが queue.tr という名前のファイルに得られる．

```
+ 0.283864 4 5 tcp 1040 ------- 0 0.0 2.0 278 435
d 0.283864 4 5 tcp 1040 ------- 0 0.0 2.0 278 435
- 0.28436 4 5 tcp 1040 ------- 0 0.0 2.0 176 282
r 0.284368 4 5 tcp 1040 ------- 0 0.0 2.0 169 272
+ 0.284613 4 5 tcp 1040 ------- 0 0.0 2.0 279 437
```

行頭の + はパケットのキューへの侵入，- はパケットのキューからの離脱，r はパケットの受信，d はパケットの廃棄を意味する．**図 3･6** は，このログを利用し，ボトルネックリンクのキュー長の変化をプロットしたものである．図 3･5 及び図 3･6 から，TCP コネクションはパケット廃棄が発生するまでウィンドウサイズを増加させ続け，パケット廃棄発生時にウィンドウサイズを半減させていることがわかる．

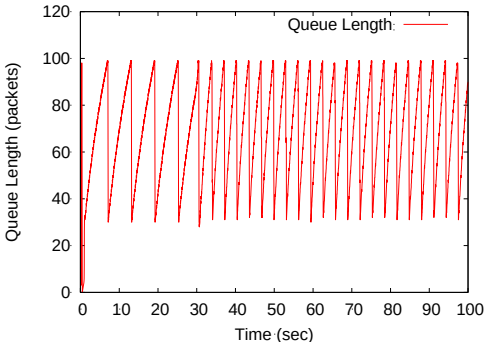


図 3・6　キュー長変化

3-2-3 TCP 輻輳制御方式の改変

ns-2 の TCP Reno の実装コードは，ns-2 のソースコード内に含まれる tcp.cc 及び tcp-reno.cc である．詳細は割愛するが，コードは C++で書かれているため，TCP Reno の動作に精通していればコードの理解は容易である．例えば，一つの ACK パケットを受信した際のウィンド

ウサイズの増加速度を変化させる場合は, tcp.cc 内の void TcpAgent::opencwnd() を改変する．改変後は，ns のバイナリーを再度作成する必要があるため，再コンパイルする必要がある．

また，Linux カーネルの TCP コードを ns-2 のコードに変換し，シミュレーションを行う手法が提案されている [2]．これを用いることで，様々な研究者が公開している TCP の改良方式の Linux コードを使って, ns-2 によるシミュレーションを行うことができる. ns-2 には, Linux に実装されている TCP に関する多くのコードが含まれている．これらを参考に，様々な TCP 輻輳制御方式や，自身による改良方式の性能評価を行うことができる．

3-2-4 シミュレーションにおける注意点

(1) Tcl スクリプトの改善

ns-2 のシナリオファイルは Tcl スクリプトによって記述されているため，スクリプト自体の改善や拡張によって，シミュレーションの手間の削減などを行うことができる．例えば配列を利用することで，ノード数やコネクション数を変化させたシミュレーションを効率良く行ったり，コマンドライン引数を利用することなどがあげられる．

また，3-2-2 項(4)において，シミュレーションログを出力する二つの方法を紹介したが，二つの方法はそれぞれ利点と欠点があるため，所望するログの精度，時間スケールなどに応じて使い分けることが求められる．特に，トレース出力や，キューイベントの出力は，非常に多くのログを出力するため，シミュレーション実行にかかる時間を大きく増加させる要因となる．一例としては，あるマシン上における 2tcp.tcl の実行に必要な時間は，ログを一切出力しない場合は約 2.7 秒だが，トレースログを出力させた場合には約 7.1 秒となる．特に，TCP コネクション数だけではなく，リンク帯域が大きい場合にも発生するイベント数が増大する．そのため，シミュレーションの円滑な実行のためには，ログのディスクへの出力の回避や，不必要なログ出力の停止などの工夫が必要である．

(2) シミュレーションを実行するハードウェア環境

ns-2 はメモリ管理のコードにバグが多く存在するため，特にシミュレートするネットワーク規模が大きい場合や，リンク帯域が大きく，単位時間当たりに処理するイベント数が多い場合に，メモリ使用量が増大し，Segmentation Fault が発生しシミュレーションの実行が停止することがある．そのため，そのような大規模なシミュレーションを行う場合には，シミュレーションを実行する計算機のメモリを潤沢に搭載することが推奨される．

また，ns-2 はバージョン間の互換性が確保されていない場合があり，研究者が公開しているパッチファイルが特定のバージョンにのみ適用可能であることが頻繁にある．また，シミュレータの実行には Tcl/Tk などの外部パッケージが複雑に関係するため，それらのバージョンの違いが動作に影響を与える場合がある．そのため，シミュレーション環境の引き継ぎを行う場合には，単に ns のバージョンを合わせるだけではなく，実行していた計算機の環境も含めることを推奨したい．特に，シミュレーション規模が小さい場合には，本解説論文が前提としたように，シミュレーション環境を仮想 PC 上に構築することで，引き継ぎが容易に行えると考えられる．

現在開発が進められている，ns の新バージョンである ns-3 においては，その実装方針が一新されている．そのため，これらの問題は解決されることが期待される．

(3) シミュレーション結果の取り扱い

商用のシミュレータである Qualnet [3] や OPNET [4] と異なり，ns-2 にはトポロジーやシナリオの設定を行うための GUI（Graphical User Interface）が存在しない．ns-2 には nam と呼ばれるシミュレーション結果をアニメーション表示するツールは存在するが，シミュレーションの設定や実行を制御することはできない．そのため，得られたシミュレーション結果が正しいか否かの判断は，ログファイルから読み取る必要がある．その際には，シミュレートしているトランスポートプロトコルに関する知識が必要となるため，プロトコルの詳細が未知のままシミュレーションを行うことは避けるべきである．特に，評価しているプロトコルの性能が，自分の期待する結果となった場合には，それをそのまま信用せず，正しいシミュレーションが行われているか否かを慎重に検討すべきである．

■参考文献

1) Network simulator-ns (version 2). available from http://www.isi.edu/nsnam/ns/
2) A Linux TCP implementation for NS2. available from http://netlab.caltech.edu/projects/ns2tcplinux/ns2linux/index.html
3) Creators of Qualnet Network Simulator Software. available from http://www.qualnet.com/
4) OPNET. available from http://www.opnet.com/