マルチメディア通信と分散処理 92－12
（１９９９．２．２６）

連続メディア通信のための動的プロトコルスタック構成モデルの実装と評価†

戸辺 義人[1]　田村 陽介[2]　徳田 英幸[3]

[1] 慶應義塾大学 SFC 研究所　[2] 慶應義塾大学大学院政策・メディア研究科　[3] 慶應義塾大学環境情報学部

あらまし

連続メディアの配送を行うサーバとの間で通信を行うクライアントにおいて，経由するネットワークの状況に応じて通信品質に差が生じる．本稿では，ネットワークに接続されたクライアントが，サーバとの通信時のデータ配送の信頼性，スループットに応じて，通信開始時および通信持続中の両方の場合に，フロー毎にプロトコルスタックを動的に変更する DPC (Dynamic Protocol stack Configuration) モデルを提案する．さらに，サーバ，クライアント間で協調してプロトコルスタックを変更するためのプロトコルとして，DPCP (Dynamic Protocol stack Configuration Protocol)，および DPC モデルに対応して socket を拡張した dsocket を定義する．本モデルを FreeBSD に実装した計算機間を無線 LAN で接続したシステムにおいて，ネットワークの負荷の変化に対して TCP と UDP との間の動的変更を適用し，DPC モデル適用の有効性が示された．

キーワード　プロトコル，QoS，TCP/IP，フロー，自動構成

Design and Implementation of Dynamic Protocol Stack Configuration Model for Continuous Media Communications

Yoshito Tobe[1]　Yosuke Tamura[2]　Hideyuki Tokuda[3]

[1]tobe@sfc.keio.ac.jp Keio University, 5322, Endo, Fujisawa-shi, Kanagawa, 252 Japan,
[2]tamura@mag.keio.ac.jp Keio University, 5322, Endo, Fujisawa-shi, Kanagawa, 252 Japan,
[3] hxt@sfc.keio.ac.jp, Keio University, 5322, Endo, Fujisawa-shi, Kanagawa, 252 Japan.

Abstract.

This paper presents a new approach towards QoS adaptation of continuous media flows. We propose that adaptation be performed by selecting a more suitable protocol for the flow depending on a communication environment. We call a model to accommodate such selection Dynamic Protocol Configuration (DPC) model. Additionaly, notions of DPC Protocol (DPCP) and dsocket are introduced. We demonstrate that the DPC model is useful in changing transport protocol between TCP and UDP over a wireless LAN.

Keywords: protocol, QoS, TCP/IP, flow, dynamic configuration

†この研究は，情報処理振興事業協会 (IPA) が実施している高度情報化支援ソフトウェア育成事業「ネットワーク指向 R3 (Reliable, Real-time, and Reconfigurable) システムのための共通基盤ソフトウェアの開発」プロジェクトのもとに行われました．

1 はじめに

連続メディアデータをネットワークに接続された計算機間で授受するアプリケーションが増えてきた [4, 5, 9]. こうした連続メディアデータをネットワーク上で転送するために, データを送り出すサーバと受け取るクライアントのアプリケーションタスク間のフロー毎に通信品質 (QoS: Quality of Service) を考慮することが重要である.

従来, 計算機通信は OSI (Open Systems Interconnection) 7 階層モデルに代表されるようにプロトコルの階層を明確に区別し, 上位層は下位層の詳細を把握することなく通信が可能となる設計がされてきた. アプリケーションは, フローを確立するときに直下のプロトコルを指定し, フローが持続する期間, 同一のプロトコルを使用し続ける. しかし, クライアントにとっては同一の連続メディアストリームを受け取るにしても, サーバとの間で形成されるフローが通過するネットワークの状況, (例えば, サーバとクライアント間は同一 LAN セグメントにありトラフィック量も少ない, 輻輳するルータを通過する, パケット欠損の高い無線ネットワークを経由する等) に応じて動的にプロトコル構成を変更したり, フロー開設時に決定した方がよい場合が生じてきた. したがってプロトコルは, ユーザの要求とネットワークの状況から選択されるのが望ましい.

以上の課題を解決するために, 我々は, 通信環境に適応して処理プロトコルを組み合わせる DPC (Dynamic Protocol stack Configuration) モデルを提案する. フローは, 2 つの計算機ノード間で成立するので, 1 つの計算機内での処理プロトコルを変更するには, 他方の計算機も同様の変更が必要となる. そのため, 両者間の協調をとるために, 同モデルを支援するプロトコル, DPCP (Dynamic Protocol stack Configuration Protocol) を定義する.

本稿では, 第 2 章で関連研究, 第 3 章で DPCP の概要, 第 4 章で本方式の FreeBSD への実装方法, 第 5 章でテストシステムでの評価結果を述べ, 第 6 章で考察を行う.

2 関連研究

動的ネットワークアーキテクチャ [8] では, ネットワークソフトウェアを構成するプロトコル群を参照関係から順序付けしてプロトコルグラフを生成する. プロトコルグラフを動的に構成するアーキテクチャを提案しており, そのための OS (Operating System) として x-kernel [3] を用いる. SCOUT [6] では, 1 ホスト内でプロトコル層を通過するパケットの流れをパスとしてモデル化し, プロトコルモジュール間を接続することによりパスを生成して送信, 受信, あるいはノード内でのタスク間通信を行えるようにする. Tschudin 等 [10] は, 同様にプロトコルスタックを固定せずに動的に生成することを提案したが, 特に言語レベルでのサポートに重点を置いている. Zitterbart 等 [11] は, プロトコルを階層化せずに, プロトコルの機能を動的に生成する方法を提案している. 最近では, Huard 等 [2] が, QoS 要求を考慮してフロー毎のデータ配送トランスポート層を決定できる, プログラマブル・トランスポート・アーキテクチャを提唱している. これらの研究は, 各層のプロトコルを動的に組み合わせたり, プロトコルそのものを動的に生成する設計方法に重点を置いており, フロー両端でのプロトコル組合せの決定方法, および信頼性を保証するトランスポートプロトコルと信頼性を保証しないトランスポートプロトコル間の切り替え等異質のプロトコルへの動的な変更に関する具体的な手順まで踏み込んでいない. 我々は, 以上の研究で解決できなかった, フローの両端で協調動作によるプロトコルの動的変更を提供する点に特徴がある. 本論文では, 送信側と受信側とでプロトコル層の組合せを変更するための情報交換とプロトコルの制御を行う DPCP の実装を元に評価を行う.

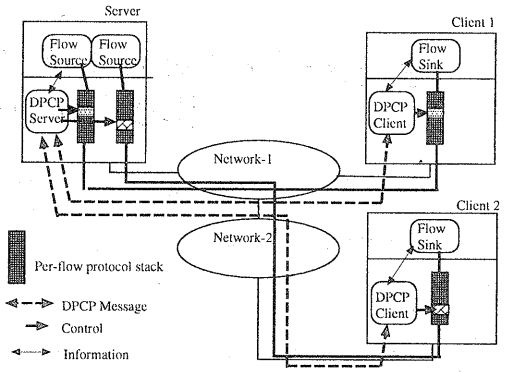

図 1: DPC モデル

3 DPCP

本章では, 送信側と受信側とでプロトコル層の組合せを変更するための情報交換とプロトコルの制御を行うプロトコル, DPCP の仕様について述べる.

3.1 基本エンティティおよび構成要素

DPCP を, DPCP サーバおよび DPCP クライアントの 2 つのエンティティ間で規定する. DPCP クライアントは DPCP サーバに対して要求を出し, DPCP サーバはその要求に対する処理結果を DPCP クライアントに返答する形式とし, 両者の間のメッセージの交換を確実に行うために, DPCP は TCP 上で動作するものとする. DPCP サーバとフローの送信アプリケーション, DPCP クライアントとフローの受信アプリケーションを同一とすることも可能である. しかし, 1 つのホストに DPCP サーバと DPCP クライアントが各々1 つだけあり, これらがホストに出入りするすべてのフローの制御をするという実装が汎用性の点で望ましい. そのため, DPCP サーバ, DPCP クライアントと, フローの送信アプリケーション, 受信アプリケーションを分離する. フローの送信アプリケーション, 受信アプリケーションを各々, フローソース, フローシンクと呼ぶ. また, DPCP サーバと DPCP クライアントを DPCP エンティティと総称する. 図 1に, DPC モデルを示す.

図 2に DPCP メッセージの形式を示す. DPCP ヘッダは 32 ビットとし, ヘッダ内にバージョン番号 (ver), 要求応答の区別 (req), コマンドの種類 (command) を含む. DPCP メッセージのコマンドの種類として, GET_PROTOCOL_LIST, GET_PROTOCOL, SET_PROTOCOL の 3 種類を設ける.

- GET_PROTOCOL_LIST: フローソースが全プロトコル層 (データリンク層, ネットワーク層, トランスポート層) でサービスとして提供可能なプロトコルおよびデフォルトで提供されるプロトコルの種類のリストを取得する.
- GET_PROTOCOL: 特定のフローソースの特定のプロトコル層 (データリンク層, ネットワーク層, トランスポート層のいずれか) で現在使用しているプロトコルの種類を取得する.
- SET_PROTOCOL: 特定のフローソースの特定のプロトコル層で使用するプロトコルを設定する. これにより, フローソース側のプロトコルスタックが変更

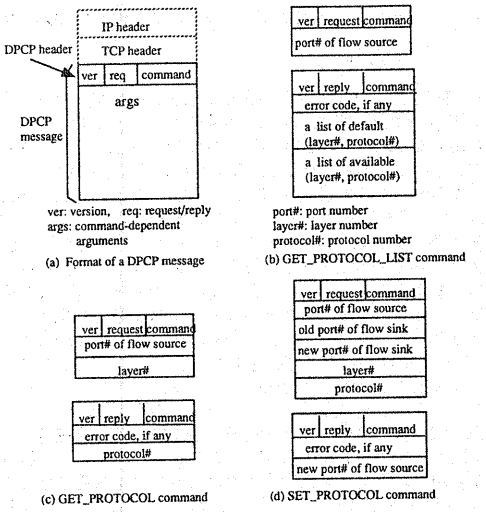

図 2: DPCP プリミティブ

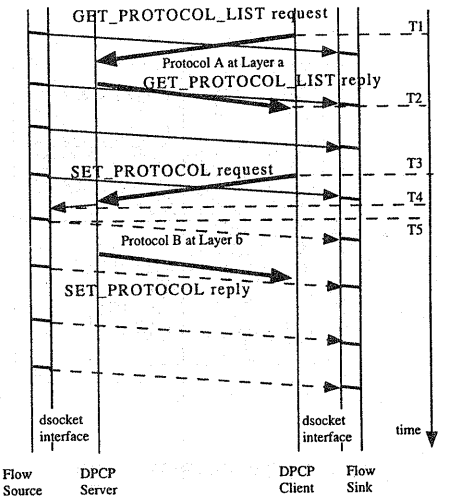

図 3: DPCP メッセージのシーケンス

される．DPCP クライアントから DPCP サーバに対してフローを指定して要求が出される．プロトコルスタックを変更したときに，フローソース側，フローシンク側のポート番号が変更になる実装が可能なように，SET_PROTOCOL 要求には，フローシンク側の変更前のポート番号の他に変更後のポート番号を含み，SET_PROTOCOL 応答には，フローソース側の変更後のポート番号を含む．

本論文の実装および評価の範囲においては，変更対象となるプロトコル層はトランスポート層だけであるが，将来への拡張に備えて，DPCP メッセージの中にプロトコル層の識別子を含めた．

3.2 DPCP エンティティの動作

前節で示した DPCP メッセージが DPCP エンティティ間でいかに処理されるかを述べる．

[フロー開設時] フローシンクは，フロー開設時にフローソースが提供するプロトコルを把握しているときには，明示的にプロトコルを指定する．この場合，DPCP エンティティ間のメッセージの交換はない．しかし，フローシンクは，利用可能なプロトコルのリストを保有しないときには，DPCP クライアントを通して利用可能なプロトコルのリストを取得し，選択する．DPCP クライアントは，DPCP サーバに対して，GET_PROTOCOL_LIST 要求を出す．DPCP サーバは指定されたフローソースに関する利用可能なプロトコルのリストを DPCP クライアントに返答する．DPCP クライアントは返答結果をフローシンクに渡す．

[フロー持続中] DPCP メッセージのシーケンスを図 3 に示す．同図においてはプロトコルスタック a を変更するものとする．DPCP クライアントは，選択したいプロトコルが，フローソースの利用可能なプロトコルリストに含まれていることを確認する．GET_PROTOCOL_LIST 応答の結果が DPCP クライアントにキャッシュされていれば，キャッシュされた情報を調べ，なければ GET_PROTOCOL_LIST 要求を DPCP サーバへ出す (時刻 T1)．DPCP クライアントはプロトコル変更が可能であることを確認した場合 (時刻 T2)，クライアント側で変更後プロトコルが動作可能となるための設定を行う (時刻 T2〜T3)．設定終了後，DPCP サーバへ SET_PROTOCOL 要求 (時刻 T3) を出す．DPCP サーバは SET_PROTOCOL 要求を受けとると，フローソースに対して，プロトコル変更指示を行ない (時刻 T4)，変更が済むと，DPCP クライアントへ SET_PROTOCOL 応答を返す．時刻 T5 においてフローのプロトコルが変更される．同図において，dsocket インタフェース (第 4 章で説明) は，アプリケーションにプロトコルの変更を意識させないためのインタフェースである．

3.3 アプリケーションとのインタフェース

DPC モデルの前提として，部分的なロスが許容されるアプリケーションのフローに関してプロトコルの変更を考慮することとした．こうしたアプリケーションを考えるときに，アプリケーション上で扱える単位， ADU (Application Data Unit) [1] を元に，受信成功，スループットといった性能を考えなければならない．そのため，アプリケーションは，送信に際しては ADU 境界を明示し，受信側で ADU 単位での性能を観測できるようにする．本論文では，ADU が複数のパケットに分割されて送信される場合，ADU を構成するすべてのパケットが正常に受信されて ADU 単位での受信に成功することを ADU が再生されると言い，ADU を構成するパケットの一部が受信されずに欠損することを ADU の部分的破壊と称する．

4 FreeBSD への実装

本章では，DPC モデルをトランスポート層の動的変更部分に限定して，FreeBSD (バージョン 2.2.6R) への実装方法について述べる．

4.1 Dsocket インタフェース

DPC モデルを FreeBSD へ実装するに当たり，カーネルへの変更を最小限にとどめるために，DPCP サーバおよび

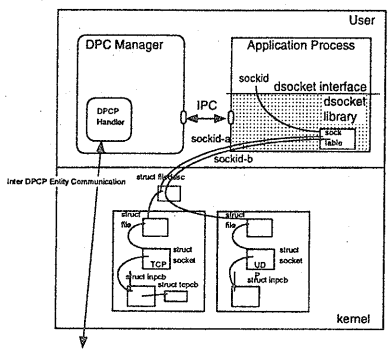

図 4: DPC の FreeBSD への実装

```
(1) int dsocket(int family, int type, int protocol,
                struct sockaddr *localaddr, int addrlen);
(2) int dbind_dest(int dsockfd, struct sockaddr *foreignaddr,
                int addrlen);
(3) int drecv(int dsockfd, caddr_t msg, size_t len, int flags);
(4) int dsend(int dsockfd, caddr_t msg, size_t len, int flags
                int adu_ind);
(5) int dclose(int dsockfd);
(6) int dpcp_request(caddr_t dpcp_server, int command,
                int *arg, int arglen);
(7) int dsocket_get(int dsockfd, int flavor, int *arg, int *arglen);
(8) int dsocket_set(int dsockfd, int flavor, int *arg, int arglen);
(9) int dgetsockopt(int dsockfd, int level, int optname, char *optval,
                int *optlen);
(10) int dsetsockopt(int dsockfd, int level, int optname, char *optval,
                int optlen);
```

図 5: dsocket の API

DPCP クライアントを DPC Manager というユーザプロセスとして実装する．アプリケーションに対して DPCP 動作が可能なインタフェースを提供するライブラリを dsocket (DPCP-capable socket) ライブラリと呼ぶ．実装形態を図 4 に示す．フローの送信プロセスおよび受信プロセスとの間でプロセス間通信 (IPC: Inter-Process Communication) を行う．

BSD UNIX で 定められる socket インタフェースは，CO 型と CL 型とを区別した API (Application Programming Interface) として定義される．例えば，CL 型の sendto() に対して，CO 型の send() があり，CL 型の recvfrom() に対して，CO 型の recv() がある．ここで，両者の違いとして，CO 型の場合には，既に通信相手が定まっているので，引数に相手に関する情報が含まれない．また，CL 型では，生成した socket に対して，IP アドレスとポート番号とをバインドする bind() を必ずしも呼ばなくてもよい．さらに大きな違いとして，CO 型の場合には，コネクションを生成するという動作が必須となるので，コネクション生成を受け付ける側で listen(), accept()，コネクション生成要求を出す側に connect() を呼び出すのに対し，CL 型では呼び出さない．このような両者の差を吸収するために，CO 型と CL 型を統一した dsocket インタフェースを定める．図 5 に dsocket API を列挙する．

4.2 パケット損失率および ADU スループットの観測

DPCP クライアントのある受信側において，パケット損失率 ρ を観測するためには，到着したパケットの「抜け」を検出しなければならない．それにはパケットにシーケンス番号がある必要がある．そこで，送信側ではパケットのペイロードとなるデータの先頭に RTP (Real-time Transport Protocol) [9] ヘッダを付け加えてシーケンス番号が含まれるようにする．受信側では，RTP ヘッダからシーケンス番号を抽出して，RTP ヘッダを除去してアプリケーションへ渡す．dsocket() を用いて送られるデータの先頭に元々RTP ヘッダがある場合には，ペイロードそのものからシーケンス番号を抽出する．アプリケーションデータの RTP ヘッダ有無は，dsocket 独自の RTP ペイロードタイプ番号を割り当てることにより判断する．以上のようにして dsocket ライブラリの中でフロー毎のシーケンス番号検査を行い，パケット損失率 ρ を記録する．TCP を用いる場合には，送信側で正常送信完了を確認している限り，ソケット層でのシーケンス番号やパケットの抜けはないが，正常動作を確認するためにシーケンス番号検査を行う．(TCP 内部でのパケットの抜けに関しては，第 7 章に述べるように送信側での観測を必要とし，現在実装していない．)

パケット損失率 ρ を計算するに当たり，計算方法を定義しておく必要がある．ここでは，以下のように定義する．

$$\rho = \frac{T_{mon}\text{中のパケット損失数}}{T_{mon}\text{中に期待されたパケット受信数}}.$$

ここに T_{mon} は，観測時間とし，実装では 100 ms とした．T_{mon} 毎に，ρ の計算を更新していく．期待されたパケット受信数は，T_{mon} の期間最初に受信したパケット，期間最後に受信したパケット，各々のシーケンス番号から計算する．

同様に，ADU スループット τ も定義する．ADU スループットは，1 個の ADU が複数のパケットに分割された場合に，ADU を構成するパケットがすべて受信に成功し，完全に再生できる ADU のスループットである．

$$\tau = (T_{mon}\text{中に再生できた ADU のバイト数})/T_{mon},$$

と定義する．ADU スループットを計算するためには，受信側で，シーケンスの抜け以外に ADU の境界を把握する必要がある．そのため，dsocket ライブラリ内部で RTP ヘッダを付加する際に，dsend() の引数として，adu_ind に 1 が設定してある場合，RTP ヘッダのマーカビットを 1 とする．これにより受信側は ADU 境界を知ることができる．

実際の実装上は，ρ および τ は，drecv() の中で計算を行うので，分母の値は正確には，T_{mon} を下回らない ADU 境界受信の間隔となる．したがって，パケット欠損がバースト的に発生し受信されない期間が長くなると，これらの分母の値も大きくなる．

5 評価

本章では，DPC モデルを実装した評価実験システムを組み，プロトコルの動的変更の評価を行った結果を記す．

5.1 評価方法

無線 LAN におけるプロトコル動的変更の効果を評価するための実験の構成を図 6に示す．同図の無線 LAN として AT&T 社の 帯域 2 Mbps の 2.4 GHz 帯 WaveLAN を用いた．Host 3 から Host 4 へはトラフィック負荷を与える目的で，両ホスト上で，nttcp [7] ソフトウェアツールを用いて周期的 UDP データを発生させた．他にトラフィックがないときに

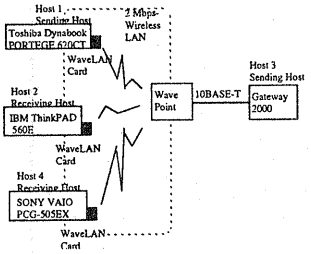

図 6: 無線 LAN 実験の構成

Host 4 上で観測されるスループット T_{back} が 0, 0.6 Mbps, 1.2 Mbps となる 3 種類の場合について比較した. nttcp に同一のパラメータを与えて同一トラフィック環境を設定したが, Host 2 上で観測されるスループットは有効数字 2 桁の範囲でしか安定しなかったので, データの処理は有効数字 2 桁で行った. また, Host 1 から Host 2 へ送るトラフィックとして, 表 1に示す TR1, TR2, TR3 の 3 種類を設けた.

表 1: トラフィックの種類

	Message length per packet	Number of packets per ADU
TR1	64 byte	1
TR2	512 byte	10
TR3	512 byte	20

5.2 プロトコルを固定したときの性能の比較

まず UDP, TCP 単体での特性を調べた. P_{rt} は 13 ms と観測された. UDP, TCP 各々で, T_{back} を 0, 0.6 Mbps, 1.2 Mbps として, TR1, TR2, TR3 の 3 種類のトラフィックを Host 1 から Host 2 へ送信する 18 種類の試行を, 各々 10 s 間 10 回行った.

Host 1 において, UDP データを送り出す時間間隔をある程度大きくしないと, カーネル内部のソケットバッファからデータが出力されるスループットよりも dsocket を使うアプリケーションから入力されるスループットの方が高くなりバッファオーバフローとなる. そこで, UDP データを送り出す時間間隔は, T_{back}が 0 のときに Host 2 で最大のスループットが観測できる値とした. ADU スループットτ は, 新たな T_{mon} サイクルに入って最初に ADU の最後のパケットを受信した時刻に記録することとした.

表 2 に, ADU スループット τ, ADU 遅延時間 D_{adu}, ADU 到着間隔 I_{adu}, パケット損失率 ρ の最大値 ρ_m を示す. ave, max, σ は各々, 平均値, 最大値, 標準偏差である. ADU スループットに関しては全通信期間を通しての平均値, ADU 遅延時間, ADU 到着間隔に関しては平均値と最大値を求め, さらに ADU 到着間隔については標準偏差を求めた. 表 2の a, b, c は, 各々, T_{back} = 0, 0.6 Mbps, 1.2 Mbps の場合に相当する. 同表において, TCP を用いて通信した場合ソケットレベルではパケット損失は生じないので, $\rho_m = 0$ となっている. TR1 送信時に, UDP と TCP とでスループットと遅延時間に大きな差が生じるのは, 送信ホストにおけるバッファリングの違いが起因している. (UDP では短いパケットを即座に送信するのに対して, TCP では即座に送信せずにソケット内部のバッファに格納し, TCP のウィンドウ制御にしたがって送信を行う.) $T_{back} = 0$ のとき, UDP による通信は TCP による通信に比較して, ADU スループットでは低くて劣るが, ADU 遅延時間では小さくて勝る. そのため, TR1, TR2, TR3 のいずれを送信する場合も, UDP, TCP のどちらかがすぐれているとは言えない.

T_{back} が 0.6 Mbps, 1.2 Mbps と大きくなるにつれ, TCP による通信の ADU スループットは低下し, ADU 遅延時間, ADU 到着間隔時間の揺らぎが大きくなる. ADU 遅延時間が極端に大きくなるのは, TCP レベルでのパケット損失をトリガとする再送タイマのタイムアウトが原因であると考えられる. それに対して, UDP による通信においては, $T_{back} = 0.6$ Mbps のときには ADU スループット, ADU 遅延時間, ADU 到着間隔時間の揺らぎ, いずれの劣化もさほどない. しかし, $T_{back} = 1.2$ Mbps までトラフィック負荷を大きくすると劣化が顕著になる. TR2, TR3 送信時のパケット損失が大きくなり, 特に TR3 送信時には最初の 2~3 s の間に ADU を再生できることもできるが, その後はパケットを受信してもパケット欠損が大きく ADU 再生が不可能となった. 表 2における, $\tau = 10 \times 10^3$ byte/s というのは, ADU 再生が可能である最初の期間での平均値であり, ほとんど通信が不可能であるため, D_{adu}, I_{adu}, ρ_m は表 2に記していない. $T_{back} = 1.2$ Mbps 時に UDP で TR3 を送信したときに, 受信側で観測されるパケット損失率 ρ の変化のスナップショットを 図 7 に示す.

5.3 プロトコル動的変更の評価

次に dsocket によるプロトコル動的変更の効果を調べた. フロー開始時のプロトコルを UDP とし, TR3 のトラフィックを Host 1 から Host 2 へ 10 s 間送信した. Host 2 側におけるプロトコル変更条件として, T_{mon} 毎に ρ を検査し, $\rho \geq 0.2$ となった時に, TCP に変更させることとした. dsocket を用いて観測された ADU スループット τ の時間変化を示したスナップショットを, 図 8 に示す. プロットされた時刻は, 各観測時間の終了時である. T_{back} は 1.2 Mbps に固定してある. 時刻 1.1 s の観測時刻において, $\rho \geq 0.2$ の条件を満たし, プロトコル変更動作に入り, Host 2 から Host 1 に対して SET_PROTOCOL 要求が渡り, プロトコルが TCP に切り替わった後, 時刻 1.8 s には最初の ADU が安定して再生できるようになっている. プロトコルが切り替わらずに UDP のままであると, UDP で固定して通信したときの結果から ADU スループットが 0 に落ち込むことが予想され, TCP に切り替わったことにより, ADU 再生を保つことが可能である. 同様の試行を 10 回繰り返し, 同様に切り替わりが観測され, dsocket を用いたプロトコル変更後の τ の平均値は 30×10^3 byte/s となり, 高トラフィック下での ADU 再生が可能となった. 同様の試行を TR1 に対して行ったが, プロトコル切り替え条件を満たさず, プロトコルは変更されなかった.

6 考察および今後の課題

評価実験においては, UDP から TCP へ変更させるための条件を与えたが, 同様に TCP から UDP へ変更させるための条件も考えなくてはならない. TCP においては, 受信側においては TCP ヘッダに含まれるシーケンス番号からだけではパケットの損失を正確に把握することができない. 送信側から出したパケットが途中で紛失しても, 送信ウィンドウが小さくてそれ以上パケットが送出されなかったり, それに続くパケットもバースト的に紛失すると送信側で再送タイマがタイムアウトし, パケットが再送され, 受信側から見ると何ごともなく正しくシーケンス通りのパケットが到着したように見えるからである. そのため, 受信側だけでは End-to-end

表 2: 無線 LAN における ADU スループット (τ[10^3byte/s]), ADU 遅延時間, ADU 到着間隔, パケット損失率の最大値

		τ	D_{adu}[ms]		I_{adu}[ms]			ρ_m
			ave	max	ave	max	σ	
UDP (TR1)	a	8.7	14	21	14	25	5	0
	b	8.7	43	61	14	25	12	0
	c	8.5	47	120	14	63	5	0.077
UDP (TR2)	a	63	110	120	80	88	6	0
	b	63	140	160	80	180	14	0.10
	c	20	680	960	180	380	110	0.83
UDP (TR3)	a	63	200	220	76	170	5	0
	b	58	240	250	170	480	53	0.83
	c	10	-	-	-	-	-	-
TCP (TR1)	a	130	120	280	1	150	3	0
	b	100	120	260	1	150	5	0
	c	49	310	1600	2	1200	20	0
TCP (TR2)	a	140	190	210	37	50	8	0
	b	100	200	350	49	180	25	0
	c	37	560	3600	140	3100	300	0
TCP (TR3)	a	140	240	350	76	180	13	0
	b	60	490	2800	190	2000	330	0
	c	30	860	6100	350	4500	560	0

の挙動を把握することが一般には難しく，送信側，受信側での情報交換が必要となる．そうした複雑さを避けるためには，TCP で通信中にも並行して UDP での通信も行ない，統計をとり希望する性能を超えていれば UDP に変更するといったことも考えられる．DPCP そのものは送信，受信側でのプロトコルスタックを一致させるためのメカニズムにしかすぎないが，そのメカニズムをどのような条件を与えて利用するかという点は，検討を重ねなくてはならない．

また，DPC および DPCP では，既存のプロトコルの組合せを変えるという手法を用いた．しかし，より細かく通信環境に適応させるには，プロトコルを変更せずにプロトコル処理で用いるパラメータ (再送タイムアウト値等) を変更したり，プロトコル処理実行間隔を動的に変更することも考えられる．RTP [9] に含まれる RTCP を用いると UDP を用いつつ送信帯域制御を行うことができる．しかしながら，パケット欠損が発生する状況においては，送信帯域を下げるだけではなく適宜再送も包含させることにより ADU スループットの向上が狙えると期待できる．

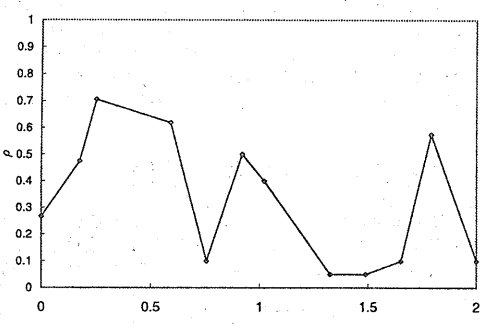

図 7: UDP による TR3 送信時のパケット損失率 (T_{back} = 0.12 Mbps)

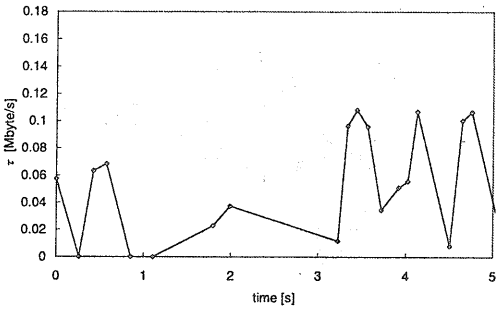

図 8: 無線 LAN における TR3 送信時の ADU スループットの変化 (T_{back} = 0.12 Mbps)

評価実験では，送信トラフィックを固定長データとしたが，MPEG エンコードされた画像データ等の実際の連続メディアデータは可変長であり，可変長データ送信による評価も必要である．今後の課題とする．

7 おわりに

従来のプロトコルスタックを固定して通信を行なう場合に欠けていた柔軟性を提供するために，本稿では，サーバとクライアントとがネットワークに接続された状況に応じてプロトコルスタックを動的に変更する DPC モデル，そのためのプロトコル，DPCP とを提案した．我々が提案するモデルを FreeBSD に実装し，無線 LAN で負荷を可変にしたネットワークにおいて有効性を確認した．

今後の課題として，上位アプリケーションにプロトコル変更をスムーズに提供するためのユーザの要求とプロトコル変更の対応の詳細な検討，プロトコルを固定してパラメータのみ変更する方法との組合せ，現実の連続メディアデータトラフィックによる検証，さらにネットワーク層，データリンク層への拡張が挙げられる．

参考文献

[1] D. D. Clark and D. L. Tannenhouse, "Architectural Consideration for a New Generation of Protocols," Proc. of ACM SIGCOMM'90, pp. 200-208 Sept. 1990.

[2] J.-F. Huard and A. A. Lazar, "A Programmable Transport Architecture with QoS Guarantee," IEEE Commmunications, pp. 54-62, Oct. 1998.

[3] N. C. Hutchinson and L. L. Peterson, "The x-Kernel: An architecture for implementing network protocols," IEEE Trans. on Software Eng., 17(1) pp. 64-76, 1991.

[4] H. Kanakia, P. P. Mishra, and A. Reibman, "An Adaptive Congestion Control Scheme for Real-Tim Packet Video Transport," Proc. of ACM SIGCOMM'93, pp. 20-31, Sept. 1993.

[5] S. McCanne and V. Jacobson, "vic: A Flexible Framework for Packet Video," Proc. of ACM Multimedia'95, Nov. 1995.

[6] D. Mosberger, and L. L. Peterson, "Making Paths Explicit in the Scout Operating System," Proc. of Symp. on Operating Systems Design and Implementation, pp. 153-167, 1996.

[7] URL:http://www.leo.org/ bartel/nttcp/

[8] E. W. O'malley, and L. L. Peterson, "A Dynamic Network Architecture," ACM Trans. on Computer Systems, 10(2) pp. 110-143, 1992.

[9] H. Schulzrinne et al., "RTP: A Transport Protocol for Real-Time Applications," RFC 1889, January 1996.

[10] C. Tschudin, "Flexible Protocol Stacks," Proc. of ACM SIGCOMM'91, pp. 197-204, Sept. 1991.

[11] M. Zitterbart, B. Stiller, and A. N. Tantawy, "A Model for Flexible High-Performance Communication Subsystems," IEEE JSAC vol. 11, no. 4, pp. 507-518, May 1993.