平成 19 年度　修士論文

惑星間インターネットにおける
バッファ管理方式

A Buffer Management System
for Interplanetary Internet

提出日 : 2008 年 1 月 29 日
指導 : 若原 恭 教授

東京大学大学院 新領域創成科学研究科 基盤情報学専攻
学籍番号 : 47-66318

久保　淳

概 要

地上系のインターネットを拡張し，惑星間に広げるという惑星間インターネットの研究が行われおり，宇宙環境に適したファイル転送プロトコルとしてCFDPが存在する．しかしCFDPでは中継ノードでのバッファ溢れへの対応が規定されていないため，本研究ではCFDPを前提としたバッファ管理方式を提案する．惑星間インターネットを「快適に使う」ためには，File Delivery Timeの短縮が不可欠であり，再送制御にかかる時間を最小化する方式が望ましい．本研究では，Sent Drop方式，RTT Aware方式，Concentrated Drop方式を提案する．これらの方式を用いることで，File Delivery Timeを大幅に短縮できることをシミュレーションによって確認した．

目次

1 序論 **1**
1.1 研究の背景 1
1.2 研究の目的 3
1.3 論文の構成 4

2 関連研究 **6**
2.1 TCP の性能劣化問題 6
2.2 TP-Planet とその問題点 8
2.3 CFDP とその問題点 11

3 提案手法 **14**
3.1 Immediate Send +ACK 方式 15
3.2 Sent Drop 方式 16
3.3 RTT Aware 方式 17
3.4 Concentrated Drop 方式 18

4 シミュレーションによる提案手法の評価 **19**
4.1 NS-2 上への CFDP シミュレータの実装 19
4.2 提案手法の評価のためのシミュレーション 21
4.2.1 シミュレーションの条件 21
4.2.2 定常状態におけるバッファ使用量 24
4.2.3 シミュレーション結果と考察 25

5 結論と将来課題 **59**

付録 **61**

謝辞 **99**

図 目 次

1	Interplanetary Internet Architecture	2
2	地球と月・火星・小惑星帯間の伝播遅延	3
3	輻輳ウインドウサイズの挙動	7
4	TP-Planet: Initial State（ [11] から引用）	9
5	TP-Planet: Steady State（ [11] から引用）	10
6	Core procedures of CFDP	11
7	Extended procedures of CFDP	12
8	CFDP の Drop-Tail 方式によるデッドロック状態	13
9	バッファ内でのファイルの状態	14
10	バッファ内でのファイルの状態（Immediate Send +ACK 方式）	15
11	Sent Drop 方式の破棄例	16
12	RTT Aware 方式の破棄例	17
13	Concentrated Drop 方式の破棄例	18
14	シミュレーション１のトポロジー	21
15	シミュレーション２のトポロジー	22
16	シミュレーション１の結果,（BER=0，Node0）	26
17	シミュレーション１の結果,（BER=1e-07，Node0）	27
18	シミュレーション１の結果,（BER=5e-07，Node0）	28
19	シミュレーション１の結果（拡大）,（BER=5e-07，Node0）	29
20	シミュレーション１の結果,（BER=1e-06，Node0）	30
21	シミュレーション１の結果（拡大）,（BER=1e-06，Node0）	31
22	シミュレーション１の結果,（BER=5e-06，Node0）	32
23	シミュレーション１の結果（拡大）,（BER=5e-06，Node0）	33
24	シミュレーション１の結果,（BER=1e-05，Node0）	34
25	シミュレーション１の結果（拡大）,（BER=1e-05，Node0）	35
26	シミュレーション１の結果,（BER=0，Node1）	36
27	シミュレーション１の結果,（BER=1e-07，Node1）	37
28	シミュレーション１の結果,（BER=5e-07，Node1）	38
29	シミュレーション１の結果,（BER=1e-06，Node1）	39
30	シミュレーション１の結果,（BER=5e-06，Node1）	40
31	シミュレーション１の結果,（BER=1e-05，Node1）	41
32	File Delivery Time の分布（Node0, BER=0, Buffer=0.6）	42
33	File Delivery Time の分布（Node0, BER=1e-07, Buffer=0.6）	43
34	File Delivery Time の分布（Node0, BER=5e-07, Buffer=0.6）	44

35　File Delivery Time の分布（Node0, BER=1e-06, Buffer=0.6） . . 45
36　File Delivery Time の分布（Node0, BER=5e-06, Buffer=0.6） . . 46
37　File Delivery Time の分布（Node0, BER=1e-05, Buffer=0.6） . . 47
38　File Delivery Time の分布（Node0, BER=0, Buffer=0.9） 48
39　File Delivery Time の分布（Node0, BER=1e-07, Buffer=0.9） . . 49
40　File Delivery Time の分布（Node0, BER=5e-07, Buffer=0.9） . . 50
41　File Delivery Time の分布（Node0, BER=1e-06, Buffer=0.9） . . 51
42　File Delivery Time の分布（Node0, BER=5e-06, Buffer=0.9） . . 52
43　File Delivery Time の分布（Node0, BER=1e-05, Buffer=0.9） . . 53
44　シミュレーション２の結果,（Node1, 1200s-10s） 54
45　シミュレーション２の結果,（Node1, 1200s-100s） 55
46　シミュレーション２の結果（拡大）,（Node1, 1200s-100s） . . . 56
47　シミュレーション２の結果,（Node1, 1200s-1000s） 57
48　シミュレーション２の結果（拡大）,（Node1, 1200s-1000s） . . 58

表目次

1　シミュレーション１の各種パラメータ 22
2　シミュレーション２の各種パラメータ 23
3　シミュレーション１における最大バッファ使用量 24
4　シミュレーション２における最大バッファ使用量 24

1　序論

1.1　研究の背景

コンピュータネットワークの普及に伴い，データ転送の大部分がコンピュータを通して行われてきている．また，無線技術や人工衛星技術の発展に伴い，コンピュータネットワークの利用範囲は、ユビキタスという言葉が示す通り，ありとあらゆる場所に広がりつつある．コンピュータネットワークを様々な環境下で効率良く動作させるために多くの研究がなされている．その中の一つとして，惑星間ネットワークの研究が進められている．

人類は，その活動範囲を地球から宇宙へと広げつつある．これに伴って，アメリカの NASA による火星探査機 “Mars Science Laboratory [1]” や日本の JAXA による月周回衛星 “SELENE [2]” など、多くのサイエンスミッション・プロジェクトが進行中である．また，これは遠い将来のことになるが，スペースコロニーやテラフォーミング [3] といった技術によって，地球以外の惑星に居住が可能になることも考えられる．こうした状況の中では，惑星内でのネットワークに加えて，惑星同士を結ぶネットワークが必要となる．実際，惑星探査機や科学調査衛星と地球局とのデータ転送は惑星間に広がるネットワークを通じてやり取りがされている．

現状で惑星間ネットワークが活躍するのはサイエンス・ミッションがほとんどであり，ネットワーク資源は比較的高価でプロプリエタリに作られている．汎用的に使える通信インフラストラクチャとしての惑星間ネットワークを構築すれば，通信にかかる金銭的なコストを軽減できる．このような動機を基にして，宇宙空間にまで張り巡らさせたネットワーク，惑星間インターネット（Interplanetary Internet, IPN）のプロジェクト [4] が NASA や Internet Society（ISOC）を中心に進められている．

惑星間ネットワークのアーキテクチャ

惑星間ネットワークとして，次のようなアーキテクチャ [5] が提案されている．

- 惑星間バックボーンネットワーク
  惑星間バックボーンネットワーク（図 1）は，惑星や人工惑星の間で通信するためのインフラストラクチャである．非常に長い距離のリンクで構成される．

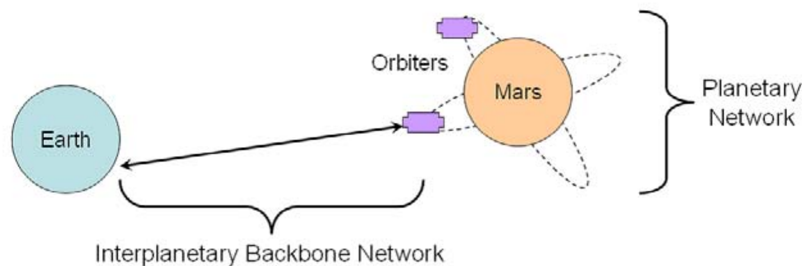

図 1: Interplanetary Internet Architecture

- 惑星ネットワーク
  惑星ネットワーク（図 1）は，惑星の表面と衛星軌道に存在するネットワークである．地球上ではインターネットがこれに当たる．

宇宙環境下でのネットワーク構築を議論するとき，ネットワーク全体を単一のプロトコルでカバーするのではなく，小さな「エリア」に分割し，エリアごとに異なるプロトコルを使用することが多い．後の 2.1 節で述べるように，惑星間インターネットの諸研究においては，惑星ネットワークでは従来のプロトコルスイート（IP や TCP）を用い，惑星間バックボーンネットワークでは宇宙環境に特化したプロトコル（CFDP など）を使うことが一般的である [5] [6]．本研究では，惑星間バックボーンネットワークにおけるプロトコルを研究対象とした．

宇宙環境の特徴

地上環境との違いとして，次のような宇宙環境の特徴 [5] [7] が存在する．

- 非常に長い伝播遅延
  地球上では大きな問題にならない光の伝播遅延も，宇宙のスケールで考えるとネットワークの構築に非常に大きな影響を与える．光速で地球火星間を往復したとしても，240 秒〜1200 秒（地球と火星の位置関係によって値が大きく異なる）という長い伝播遅延が発生する．

- 高いエラー率
  宇宙における通信は無線を使う．そもそも無線は有線に比べ高いエラー率があるが，宇宙では電磁波や太陽風といった外部からの妨害により，さらに高いエラー率となる．

- 断続的な接続性
  宇宙環境下においては，一度確立されたリンクが断絶することがしばしば発生する．この原因には，高いエラー率のほかに，惑星や飛翔体の移動や電力不足が挙げられる．

以上の特徴は，現在のインターネットでは想定されていないため，現在使われているプロトコル（TCP や IP）をそのまま宇宙に持ち込んでも動作しないことや動作しても極端な性能劣化が起こることがある．そこで，宇宙環境下で用いることを前提とした新たなアーキテクチャ [6] やプロトコル [8] の開発が進んでいる．

1.2 研究の目的

宇宙環境下では伝播遅延が人間にとって「待つ」ことが意識されるオーダーの時間になる．図 2 のように，地上系のネットワークでは伝播遅延が多くとも 1 秒のオーダーであるが，宇宙では 100 秒オーダーの伝播遅延がかかることが珍しくない．加えて，送信元によって伝播遅延に 2 〜 3 桁の違いがでてくることもある．

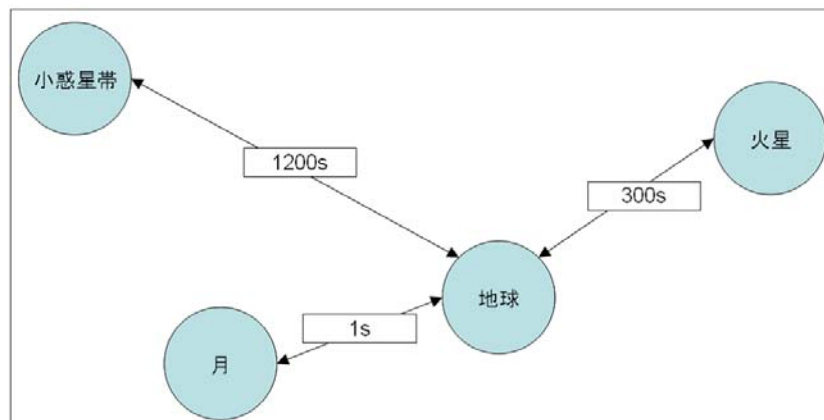

図 2: 地球と月・火星・小惑星帯間の伝播遅延

宇宙環境下でのデータ転送では，伝播遅延の影響が大きいため，送信者がファイルを送り始めてから受信者が正常に受け取り終えるまでの時間 (File Delivery Time) が重要な指標となる．以下では，宇宙環境下において File Delivery Time を最小化することを目的とする．

本研究では，TCP ようなの輻輳制御（2.1 節に述べる）をせず，CFDP のようにパケットを送信し続ける（2.3 節に述べる）プロトコルを前提とし，File Delivery Time は次の式で定義する．

$$FDT = \sum_{l \in Link} T_{wait}(l) + \frac{S}{B(l)} + T_{prop}(l) + T_{ret}(l) \times max(N_{ret}(i)) \quad (1)$$

ただし，$Link$ は Source-Destination 間のリンクの集合で，リンク $l \in Link$ において，$T_{wait}(l)$ は l の送信バッファでの待ち時間，S はファイルサイズ，$B(l)$ は l の帯域，$T_{prop}(l)$ は l の伝播時間，$T_{ret}(l)$ は l で再送にかかる時間（再送パケットの $T_{wait}(l)$ や $T_{prop}(l)$ などを含む），$N_{ret}(i, l)$ は l での i 番目のパケットの再送回数である．

この式において，注目すべきは $T_{ret}(l) \times max(N_{ret}(i))$ の項である．この $T_{ret}(l)$ には少なくとも $2 \times T_{prop}(l)$ が必要なため，この項の影響を極力少なくすることが，File Delivery Time の最小化に繋がる．本研究では，N_{ret} を減らし，T_{ret} を小さくするような方式を目指す.

再送を引きこす原因として，エラーによるパケットの破棄と中継ノードのバッファ溢れによる破棄が考えられる．エラーによる破棄への対処については, CFDP のような再送制御やターボ符号 [9] に代表される誤り訂正符号が挙げられる．バッファ溢れによる破棄への対処は，バッファ管理方式に依存する．

地上のネットワークと同様に宇宙ネットワークにおいても，バッファは溢れないように設計するのが通常である．しかし，バッファの交換・増設には非常に大きなコストがかかるため，例え故障したとしても地上のようにすぐに交換するということは不可能である．また，サイエンスミッションのようにデータフローがスケジューリング可能であれば別であるが，地上のインターネットのようにフローの性質や量が確率的にしか分からない場合は，溢れが発生しないほど十分大きなバッファが常に用意されているとは限らない．送信前に帯域確保を行いバッファのあふれを無くす研究もあるが，このためには他ノードとの交渉に少なくとも 1RTT（Round Trip Time ＝往復伝播遅延）がかかり，宇宙環境下では無視できない大きさとなるため現実的ではない.

本研究では，バッファ溢れによる破棄への対処に焦点を当て，宇宙環境に適するバッファ管理方式を提案することとした.

1.3 論文の構成

まず第 2 章で関連研究を紹介する．次に第 3 章で，惑星間インターネットにおけるバッファ管理方式を提案し，第 4 章において，提案手法の評価のための

シミュレーション方法を述べ，結果と考察を記載する．最後に第 5 章で結論と将来課題を述べる．付録として，NS-2 上に実装したソースコードとシナリオファイルを記載する．

2 関連研究

惑星間インターネットに用いることを前提としたトランスポート層プロトコルに TP-Planet [11]，トランスポート層からアプリケーション層に位置するファイル転送プロトコルに CFDP [13] がある．この章では，TP-Planet の基本となっている TCP を長遅延環境下で使用した場合に発生する性能劣化について議論した後，その性能劣化を改善したプロトコル TP-Planet を紹介する．その次に，宇宙でのファイル転送用プロトコル CFDP を紹介する．

2.1 TCP の性能劣化問題

ネットワークが混雑すると，中継ノードでバッファが溢れてパケットの損失が発生することがある．これを防ぐため，TCP ではリンクの混雑度によって転送レートを変化させる輻輳制御を行う．輻輳制御は輻輳ウインドウサイズ（cwnd）を変化させることで実現する．ここで輻輳ウインドウサイズとは，ACK を待たずに送信することの出来るデータのサイズである．現在一般に使われている輻輳制御は，最小の cwnd から始めて，ACK が正常に戻ってきたら徐々に cwnd を増加させ，戻ってこない ACK があると輻輳が発生したとして cwnd を半減させる（図 3）．これをスロースタートアルゴリズムという．

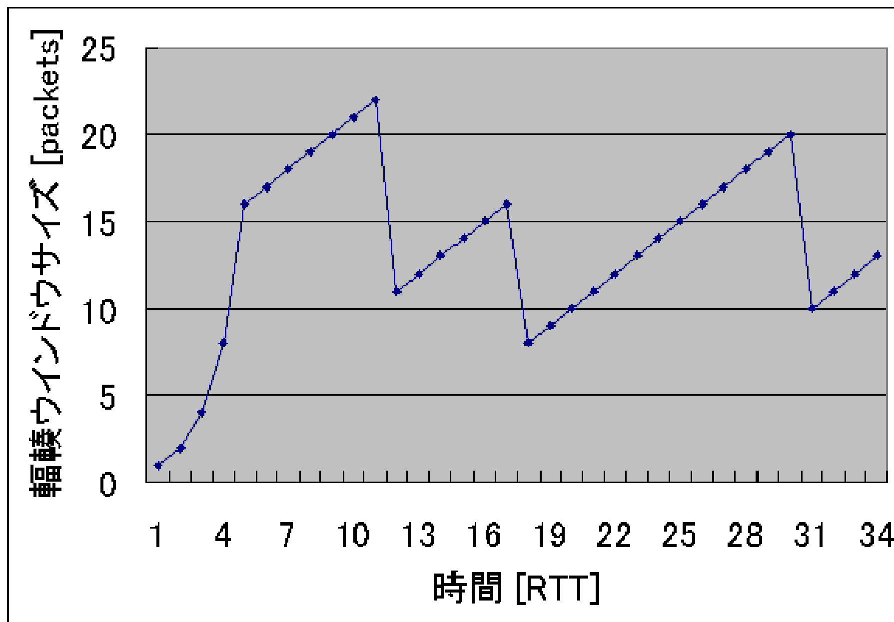

図 3: 輻輳ウインドウサイズの挙動

長遅延環境下では ACK が送信先から戻ってくるまでの時間が非常に長くなる．スロースタートアルゴリズムでは，ACK が戻るまでの間は cwnd を増加させないので，帯域にはまだ空きが有るのにもかかわらず使うことができない．加えて，cwnd の増加は「ゆっくりと」行われるのに対して，減少は「すばやく」行われる．そのため，長遅延下でスロースタートアルゴリズムを使うとスループットの低下を招く．

Long Fat Pipe 問題

スロースタートアルゴリズムのために帯域を効率よく使うことが出来ないという問題は，地上系のネットワークを構築する際に “Long Fat Pipe 問題” として指摘されている．Long Fat Pipe 問題への対応策として，FAST TCP や XCP が知られている [10]．これらは，帯域遅延積 $B \times RTT$（RTT：往復伝播遅延，B：帯域幅）が大きい場合について，スループットの低下を改善する技術である．

惑星間ネットワークも高帯域幅遅延積のシステムと言えるが，地上系の高帯域幅遅延積のシステムとは少し違う部分がある．例えば，地上系の高帯域幅遅

延積システムに $B = 10$[Gbps]，$RTT = 60$[ms] というシステムがあるが，同じ帯域幅遅延積のシステムであっても，宇宙系では $B = 1$[Mbps]，$RTT = 600$[s] となる．このように，同じ高帯域幅遅延積のシステムといっても，宇宙と地上では遅延と帯域幅の割合が違うため，地上系の Long Fat Pipe 問題の対応策をそのまま宇宙ネットワークに適応させても効率的ではないことがある．

例えば，宇宙ネットワークでは RTT を基準にする時間（パケットの再送や中継ノードの状態通知にかかる時間など）が File Delivery Time に与える影響は，RTT によって変化しない時間（バッファでの待ち時間や計算処理など）が File Delivery Time に与える影響に比べて非常に大きい．すなわち，宇宙ネットワークにおけるデータ転送では，例え後者が増えたとしても前者を減らすような方式が良いということになる．

また，地上系のシステムではネットワークの情報を収集した上で，輻輳制御などを行う方式があるが，これを長遅延環境に適用させようとすると不都合が生じる可能性がある．なぜならば，隣接ノードの情報を収集するのにかかる時間が非常に長いために，その間にネットワークの状況が変化して情報がすでに古いものになっており，現状を反映していない可能性が大きいからである。長遅延環境下では「1RTT を待つ」というのは極力避けるべきであり，隣接ノードの情報を元に制御を行う方式は宇宙環境にはあまり向かないと言える．

2.2　TP-Planet とその問題点

惑星間インターネットに用いることを前提としたトランスポート層プロトコルに，TP-Planet [11] がある．TP-Planet は，TCP を長遅延環境下で使用する際に発生する極端な性能劣化の改善を目的としている．

TP-Planet では，TCP のように転送レートの調整を RTT ごとに行うのではなく，間隔

$$T = \sqrt{\frac{RTT}{B}} \tag{2}$$

ごとに行う．ここで B(packet/s) は送信側が指定する Target Throughput である．

$t < RTT$ の間を Initial State の Immediate Start Phase という．この間，転送レートは閾値

$$ssthresh_e = \sqrt{RTT \times B} \tag{3}$$

まで T ごとに増加させる（図 4）．このように TP-Planet では転送レートを素早く増加させ，Target Throughput を早い時点で実現する仕組みになっている．またこの間，データパケットの他に優先度の低い NIL セグメントを送信する．

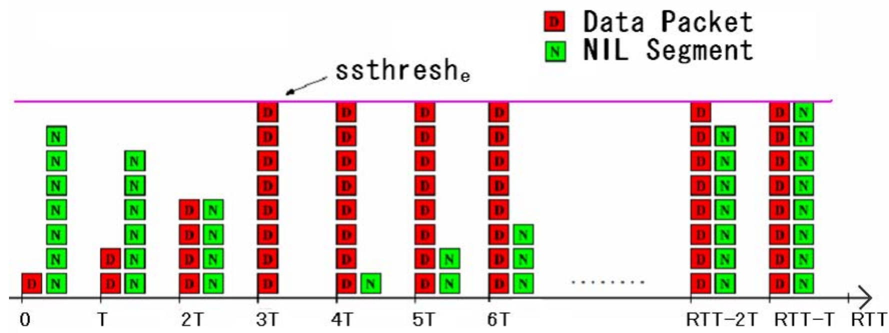

図 4: TP-Planet: Initial State（ [11] から引用）

$RTT \leqq t < 2 \times RTT$ の間は Initial State の Follow-up Phase という．この間は NIL セグメントに対する ACK（NIL ACK）を元に転送レートを決める．それと同時に，送信側は NIX セグメントを受信側に一定間隔で送り続ける．NIX セグメントは，データを持たないセグメントで，高優先度のものと低優先度のものの 2 種類あり，同じ転送レートで送り出される．

$2 \times RTT \leqq t$ は，Steady State である（図 5）．到着した高優先度のセグメント数（N_{high}）と低優先度のセグメント数（N_{low}）の割合から転送レートの制御を行う．

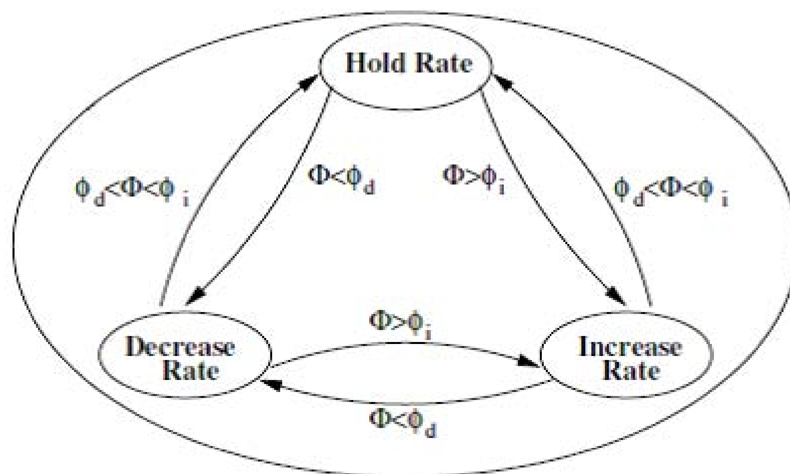

図 5: TP-Planet: Steady State（ [11] から引用）

$t < RTT$ の Immediate Start Phase においては，TP-Planet は 2.3 節で述べるように CFDP と同じように「送れるだけ送る」という方針を採る．高優先度と低優先度のセグメントを設定し，輻輳によりバッファが溢れることがあれば低優先度のセグメントが破棄される．そのため，他のフローが定常状態にあるときに新たなフローが発生すると，低優先度のセグメントの破棄は避けられない．したがって，低優先度のセグメントは破棄されても良いデータでなければならず，到着保証が必要なデータのフローだけがシステム上に存在するときは，うまく機能しない．

また，Follow-up Phase や Steady State において，高優先度と低優先度のセグメント数の割合から転送レートを決定するが，転送レートを送信ノードへ知らせるために少なくとも 1RTT かかってしまう．この間，フローの状況が変化し輻輳状態が変わってしまうことがあり,「最善の」転送レートではない可能性がある．さらに，TP-Planet では End-to-End で制御を行うため，より影響が大きくなる．

2.3 CFDPとその問題点

宇宙における通信規格を策定する国際機関 Consultative Committee for Space Data Systems [12]（CCSDS）で提案されているプロトコルとして，CCSDS File Delivery Protocol [13]（CFDP[1]）がある．CFDP によるファイル転送は，Core procedures と Extended procedures という二つの手順が用意されている．

Core procedures は，送信者（Sender）と受信者（Receiver）の間で行われるファイルのコピー（Copy File）の手順である（図 6）．転送はメタデータ，データ本体，EOF の順に行われる．ACK モードのときは，さらに NAK，EOF ACK，FIN，FIN ACK が追加される．NAK 方式で送達確認が行われるのは，貴重な帯域をなるべく消費しないように，という設計思想に基づくものである．

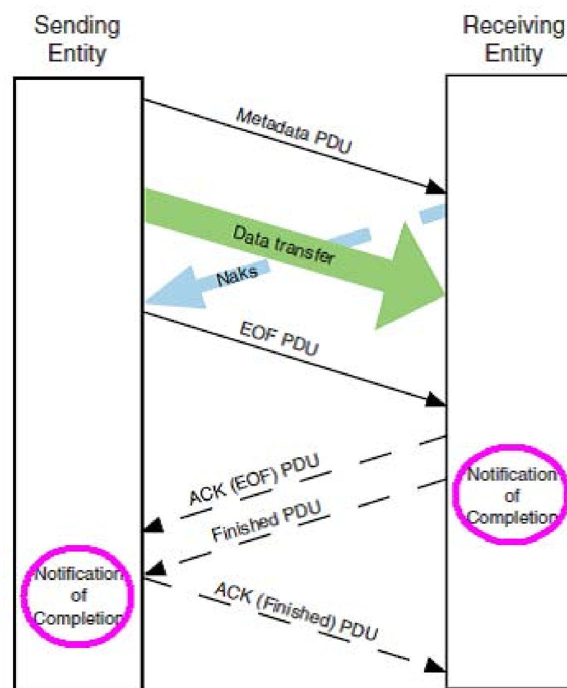

図 6: Core procedures of CFDP

出発地（Source）と目的地（Destination）の間で直接通信が出来ない場合，Extended procedures が行われる．この手順は，ストレージを持つ中継ノード間

[1] RFC 1235 の The Coherent File Distribution Protocol との混同に注意

でファイルをストア＆フォワードして最終的に Destination まで転送するものである（図 7）．End-to-End の通信路が確保できなくとも転送を行うことが可能なため，断続的な接続性に対応していると言える．

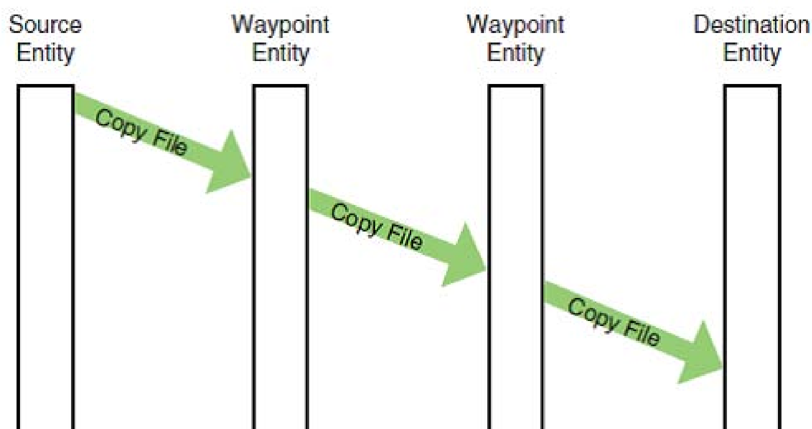

図 7: Extended procedures of CFDP

CFDP は単一の Source と Destination の間のファイル転送を想定しているため，中継ノードでの輻輳は発生しないことが前提となっており，TCP のようなウインドウ制御を行わない．このため，CFDP を用いて惑星間ネットワークを複数の Source で共有することを考慮すると，中継ノードにおいて輻輳が発生しバッファが溢れる可能性がある．

バッファの溢れは File Delivery Time に深刻な影響を与える．なぜならば，Reliability を保証すべきファイル転送の場合，バッファの溢れによってパケットの損失が発生し，再送せざるを得ないからである．再送のためには少なくとも 1RTT かかることを考慮すると，バッファの溢れへの対応が重要であるが，CFDP ではその規定が無い．

地上でのバッファ管理の多くは「バッファが溢れたとき，最後に来たパケットを破棄する」という Drop Tail 方式である．単純明快なポリシーで実装も簡単なので広く使われている．しかし，CFDP でパケット単位の Drop Tail 方式を採用すると，いわゆるデッドロック状態に陥る危険性があることが本研究の過程で明らかになった．

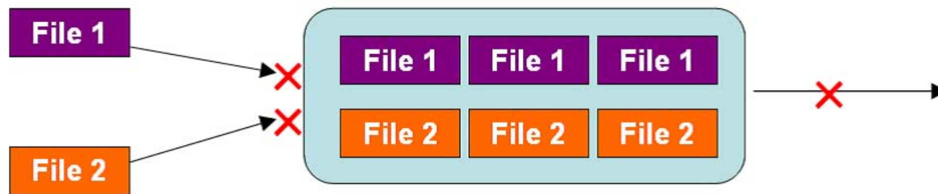

図 8: CFDP の Drop-Tail 方式によるデッドロック状態

例として，図 8 のような状況を考える．１ファイルが４つのパケットに分割され，File1, File2 いずれも３パケットがこのノードに届いているとする．この状況でバッファが満杯であったとすると，Drop Tail 方式では次に来たパケットを破棄するため，File1, File2 のどちらのパケットが来ても破棄されてしまう．CFDP では，全てのパケットがそろうまでパケットを保持しておき，全てのパケットがそろってから次のノードにファイルを送信し，次ノードからファイルがそろったことを伝える FIN パケットが届いてからパケットを消去する．このため，図 8 の状況ではパケットを消去することもできず，パケットを受信するためのバッファの空きが確保できない．

このように，パケット単位で Drop Tail 方式を用いると CFDP はデッドロック状態に陥る可能性がある．ここで，Meta パケット受信時に，ファイル容量分のバッファを確保するファイル単位での Drop Tail であれば，前述のようなデッドロックは発生しない．しかし，バッファ管理方式によって，File Delivery Time が大きく左右される．本研究では，CFDP を前提としたファイル転送におけるバッファ管理方式を提案し，File Delivery Time を短縮する．

3　提案手法

この章では，本研究の提案方式である Immediate Send +ACK 方式，Sent Drop 方式，RTT Aware 方式，Concentrated Drop 方式を説明する．以下，全てのパケットがそろっていないファイルを Incomplete File，全てのパケットがそろったが次のノードへ送信していないファイルを Complete File，次ノードに送信したがまだ FIN が帰ってきていないファイルを Sent File と呼ぶことにする（図 9）．前ノードから届いたファイルは，まず Incomplete File とされる．Incomplete File のパケットが全てそろえば Complete File となり，Complete File が次ノードへ送信されれば Sent File となる．Sent File は次ノードから FIN パケットが届くとバッファから消去される．

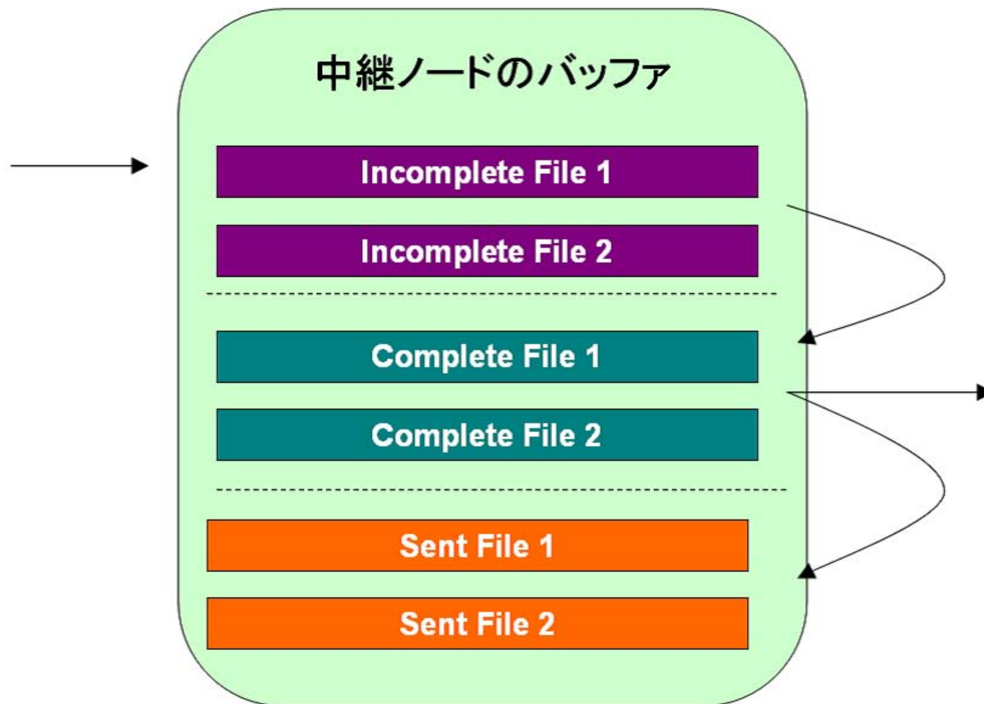

図 9: バッファ内でのファイルの状態

3.1 Immediate Send +ACK 方式

CFDP は，次ノードから送信された FIN パケットを受信するまでバッファから File を消去しない．バッファが溢れることが無ければ問題ないが，バッファの溢れを考慮する場合，ファイル単位での消去は効率的ではない．確かに，NAK が届いたパケットは再送制御で次ノードへ送信した後も，再び再送が必要になる可能性があり，Sent File としてバッファに保持しておく必要がある．しかし，NAK が届かなかったパケットは次ノードへの到着が確認できるため，FIN パケットを受信するまで消去しないのは無駄である．

そこで本研究では，Receiver ではパケットを受け取ったら ACK を Sender に返し，Sender では ACK の受信をもってバッファからパケットを消去してよい，という方式に CFDP を修正することを提案する．また，中継ノードで早く ACK を返すため，パケットを受け取ったらすぐに次のノードへ送るという方式を提案する．この方式を適応すると,「パケットがそろうのを待つ」ことが無くなるため，Incomplete File と Complete File の区別が無くなる（図 10）.

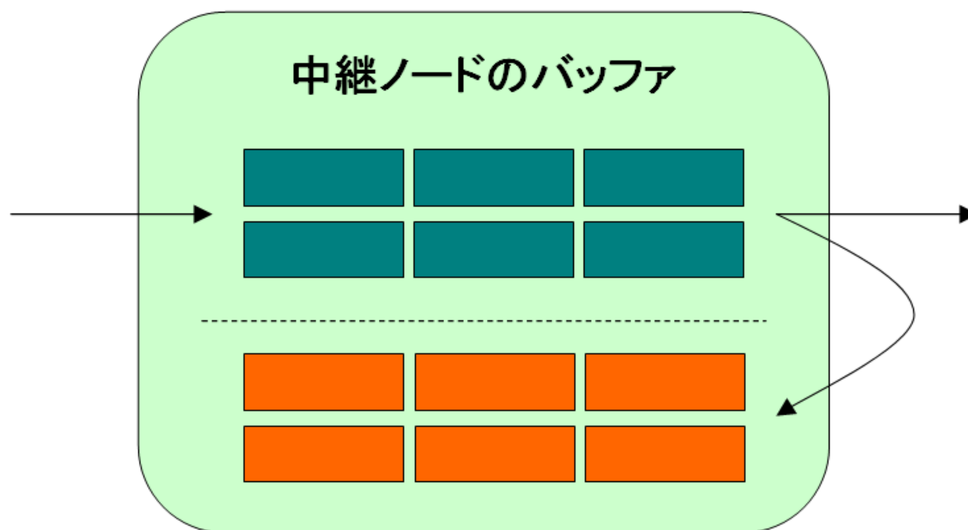

図 10: バッファ内でのファイルの状態（Immediate Send +ACK 方式）

3.2　Sent Drop 方式

Immediate Send +ACK 方式を取り入れたとしても，バッファからパケットを消去できるのは ACK の受信後であるから，次ノードへパケットを送信した後も中継ノードのバッファで保持しておく必要がある．現在の CFDP では Sent File パケットを破棄できないため，Incomplete File パケットをから破棄せざるを得ない．Incomplete File パケットを破棄した場合，再送が必ず必要になる．

そこで本研究では，Sent File パケットの破棄を許可する方式を提案する．Sent File パケットは，NAK が帰ってこなければ，破棄しても再送する必要が無い．Incomplete File パケットは破棄すると必ず再送が必要であるため，Sent File パケットを破棄することで再送回数を減少させることができる．図 11 でパケットを破棄しなければならないとき，パケット８～パケット１３のいずれかを破棄するようにする．

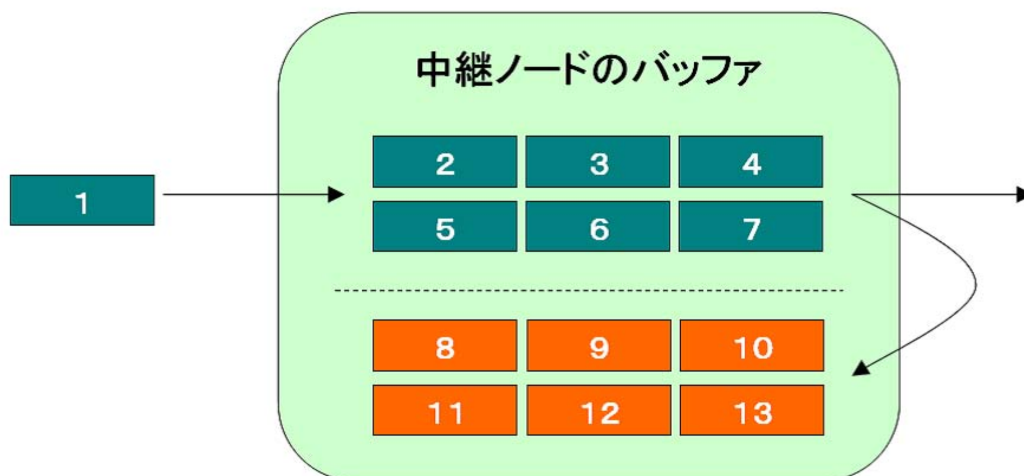

図 11: Sent Drop 方式の破棄例

ただし，Sent Drop 方式を用いる場合，NAK が中継ノードに届いた時に対応するパケットがバッファの溢れにより破棄されおり，再送ができない場合が考えられる．そこで，このような場合には「再送要求を Source に対して行う」（以下，Source Retransmission と言う）制御をするよう CFDP に修正を加えるようにする．エラーやバッファ溢れによるパケットの破棄率が高く，Source Retransmission の発生頻度が高い場合にこの方式を用いると，用いない場合に

比べて File Delivery Time が長くなることもあるが，破棄率が低い場合はこの方式は有効である．

3.3 RTT Aware 方式

Sent Drop 方式を採用した場合，Source Retransmission が発生する可能性があるため，破棄するパケットは慎重に選ぶ必要がある．なぜならば，パケットによって再送にかかる時間が大きく異なるからである．例として，地球で火星と月からのパケットを受け取ったとする．火星からのパケットを破棄した場合，再送要求を出してから再送パケットが届くまで 600 秒かかるが，月からのパケットを破棄した場合は 2 秒で済む．惑星間インターネットではこのように伝播遅延が 3 桁違ってくることがあるため，再送回数は同じ一回でも再送にかかる時間が大きく異なることがある．

そこで本研究では，破棄するパケットを決定する際，Source までの伝播遅延が最も短いものから破棄する方式を提案する．図 12 でパケットを破棄しなければならない場合は，RTT が最小のパケット８〜パケット１０のいずれかを破棄する．

図 12: RTT Aware 方式の破棄例

RTT Aware 方式を用いることによって，長い伝播遅延を経験するパケットは再送する回数が減少し，File Delivery Time の合計を短縮することができる．

3.4 Concentrated Drop 方式

再送制御の際，同じファイルのパケットで再送が必要なものが他にある場合，そのパケットとまとめて再送制御を行うことができる．

例えば，RTT が 600 秒のリンクがあり，そこで再送が起こることを考える．バッファには File1 と File2 のパケットが存在し，時刻 0 秒と 50 秒の 2 回バッファが溢れて破棄しなければならないとする．このとき，時刻 0 秒で File1 のパケットを破棄し，時刻 50 秒で File2 を破棄したとすると，File1, File2 ともに File Delivery Time が 600 秒延長することになる．しかし，時刻 0 秒と時刻 50 秒の両方で File1 のパケットを破棄した場合，時刻 50 秒で破棄したパケットが戻ってくる時刻 650 秒で再送制御が終わる．このとき，File1 の File Delivery Time は 650 秒延長し，File2 は延長しない．このように，システム全体では File1 のパケットを２つ破棄した方が File Delivery Time の延長が少なくて済む.

そこで本研究では，破棄されるパケットが特定のファイルに集中する方式を提案する．図 12 でパケットを破棄しなければならない場合は，前にパケット１０が破棄されているのでパケット８かパケット９を破棄する．

図 13: Concentrated Drop 方式の破棄例

この方式を用いれば，公平性は損なわれるが，全体の File Delivery Time は向上する．

4　シミュレーションによる提案手法の評価

この章では，提案手法の実効性を確認するためのシミュレーションを行い，提案手法の評価について述べる．

4.1　NS-2 上への CFDP シミュレータの実装

ここでは，ns-2 [14] 上に実装した CFDP シミュレータについて述べる．CFDP のシミュレーションに関しては非公開のシミュレータによる実験報告 [15] があるものの，ソースコードが公開されているものは存在しないため，ns-2 上に実装することにした．

ns-2 の実装方針 [16] に基づき，C++言語によってプロトコルの挙動を，Tcl 言語によってシミュレーションのシナリオファイルを作成した．次に実装した CFDP シミュレータの主な仕様を示す.

- エラーの発生
  ビット単位でランダムに発生する．1bit でもエラーが発生すればパケット全体を破棄する．

- 制御パケット
  制御パケット（Meta, ACK, NAK, FIN, Final FIN）にはエラーが発生しない．

- 再送パケット
  他のパケットに優先して送り出すことはしない．

- バッファ使用量
  バッファ使用量は，Incomplete File，Complete File，Sent File の総合計とし，この総合計がバッファ量を超えたら溢れたものとする（Incomplete File，Complete File，Sent File ごとの制限を設けない）．

- Source のバッファ量
  Source においては，バッファ量は無限にあり，溢れることはないとする．ファイルの発生順に送信バッファに入り，その順番で送信される．

- ルーティング
  ルーティングテーブルは外部から与えられているものとする．

- Deferred NAK
  EOF パケットが届いてから，NAK パケットを返す.

- FIN
  全てのパケットがそろったら，FIN パケットを返す.

- 複数 Source への対応
  パケットをフロー別に区別して取り扱う.

- ACK（独自仕様，オプション）
  パケットを受信したら ACK を返す．ACK を受信したらパケットを消去してよい.

- Source Retransmission（独自仕様，オプション）
  Source までパケットを取りに帰る制御を行う．Final FIN と同時に用いる.

- Final FIN（独自仕様，オプション）
  Source Retransmission のために，Source では Destination からの Final FIN を受信するまでパケットを消去しない.

実装したシミュレータの信頼性を確かめるため，CFDP の評価に関する実験報告 [15] のパラメータでシミュレーションを行い，結果が一致することを確認した.

4.2　提案手法の評価のためのシミュレーション

提案手法の評価を行うにあたって，二種類のシミュレーションを行った．シミュレーション１ではビットエラー率の違いによる影響，シミュレーション２ではトポロジーの違いによる影響を調査することを目的とした．

4.2.1　シミュレーションの条件

シミュレーション１のトポロジーを図 14，各種パラメータを表 1 に，シミュレーション２のトポロジーを図 15，パラメータを表 2 に示す．

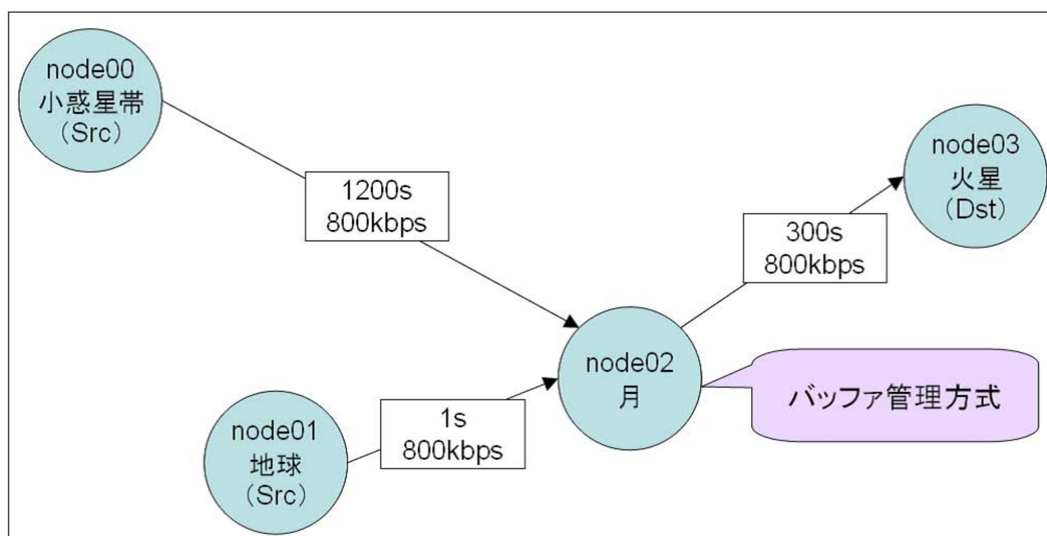

図 14: シミュレーション１のトポロジー

表 1: シミュレーション１の各種パラメータ

パケットサイズ	1,000 [B]
ファイルサイズ	10,000 [B]
Node0 でのファイル平均発生間隔	0.5[s/file]（ポワソン分布）
Node0 でのファイル発生数	25,000
Node1 でのファイル平均発生間隔	0.5[s/file]（ポワソン分布）
Node1 でのファイル発生数	25,000
ビットエラー率	$0, 10^{-7}$, 5×10^{-7}, 10^{-6}, 5×10^{-6}, 10^{-5}
実験回数	Random Seed を変えて 10 回ずつ

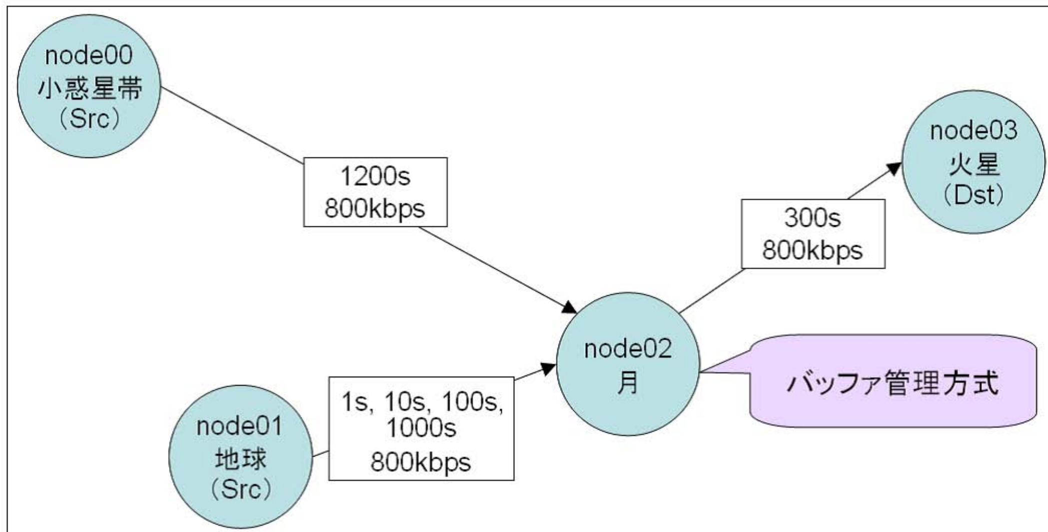

図 15: シミュレーション２のトポロジー

表 2: シミュレーション 2 の各種パラメータ

パケットサイズ	1,000 [B]
ファイルサイズ	10,000 [B]
Node0 でのファイル平均発生間隔	0.5[s/file]（ポワソン分布）
Node0 でのファイル発生数	25,000
Node1 でのファイル平均発生間隔	0.5[s/file]（ポワソン分布）
Node1 でのファイル発生数	25,000
ビットエラー率	10^{-5}
実験回数	Random Seed を変えて 10 回ずつ

4.2.2　定常状態におけるバッファ使用量

中継ノード (Node2) のバッファ使用量を見積もるため，評価のためのシミュレーションに先立ち予備実験を行った．予備実験では，中継ノード (Node02) のバッファ量を制限せず，溢れが発生しない状況におけるバッファ使用量の挙動を調べた．シミュレーション１における定常状態の最大バッファ使用量を表 3 に，シミュレーション２における定常状態の最大バッファ使用量を表 4 に示す．

表 3: シミュレーション１における最大バッファ使用量

ビットエラー率	最大バッファ使用量
0	25.0 [MB]
10^{-7}	25.0 [MB]
5×10^{-7}	25.1 [MB]
10^{-6}	25.5 [MB]
5×10^{-6}	26.0 [MB]
10^{-5}	28.5 [MB]

表 4: シミュレーション２における最大バッファ使用量

Node1-Node2 の伝播遅延	最大バッファ使用量
1 [s]	28.6 [MB]
10 [s]	28.6 [MB]
100 [s]	28.6 [MB]
1,000 [s]	28.6 [MB]

4.2.3　シミュレーション結果と考察

提案手法の有効性を確認するため，2.3 節で述べたファイル単位での Drop Tail 方式（File Drop Tail），3.2 節で述べた Sent File パケットを破棄する方式（Sent Drop），3.3 節で述べた RTT の考慮を Sent Drop 方式に加えた方式（RTT Aware），3.4 節で述べた破棄するパケットをなるべく同じファイルから選択することを RTT Aware 方式に加えた方式（Concentrated Drop）の 4 方式についてシミュレーションを行った．いずれの方式についても，3.1 節で述べた Immediate Send +ACK 方式を取り入れた．以下に結果と考察を記述する．

なお以下で，定常状態における最大バッファ使用量とは，4.2.2 節で求めた値である．また，全てのシミュレーションは random Seed を変え 10 回ずつ行い，結果はその平均である．

シミュレーション１

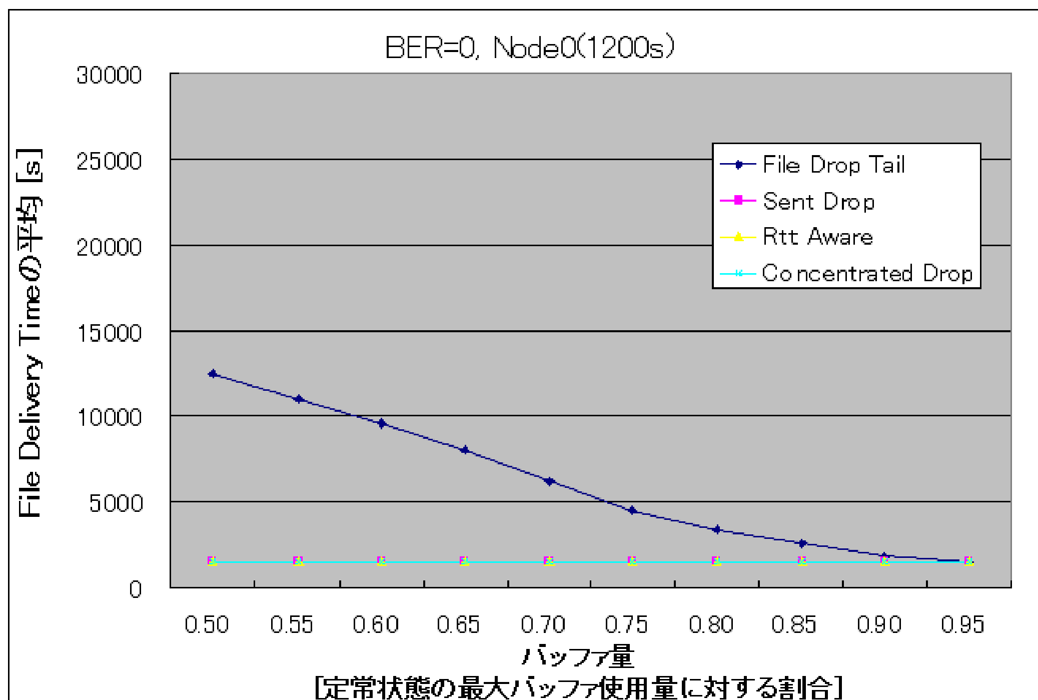

図 16: シミュレーション１の結果,（BER=0, Node0）

図16で，File Drop Tail方式ではバッファ量が小さくなるにつれてFile Delivery Timeの平均が長くなっているのに対して，Sent Drop，RTT Aware，Concentrated Dropの各方式は，平均1500.3[s]で転送が完了している．

バッファの溢れが無いとき，伝播遅延1200[s]+300[s]，送信時間0.1[s]，平均送信待ち時間0.2[s]を合計した1500.3[s]になるはずである．

したがって，Sent Drop，RTT Aware，Concentrated Dropの各方式では溢れが起こらなかったときとFile Delivery Timeが同一になると言える．これは，バッファが溢れたとき，Sent Fileパケットから破棄することでバッファの空きを確保することができるためと考えられる．

このことから，Sent Drop方式を採用すると，バッファが溢れても再送する必要が無くなることがあり，再送回数が減少する．

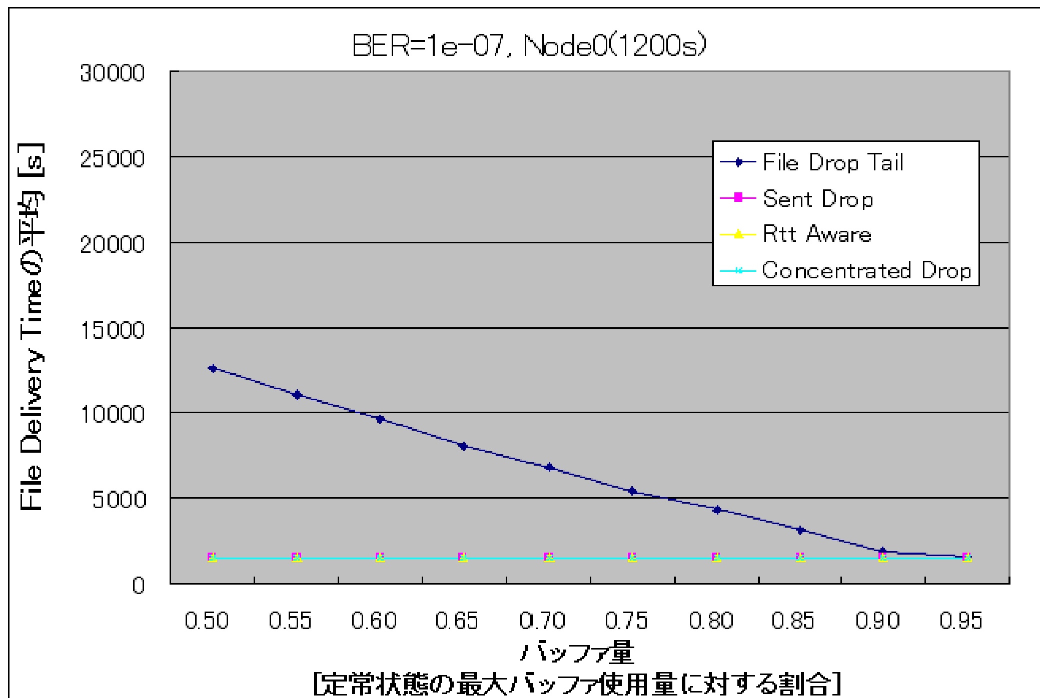

図 17: シミュレーション 1 の結果,（BER=1e-07，Node0）

図 16 に比べて，図 17 はエラーによる影響があるため，File Delivery Time の平均は全体的に長くなる．例として，Buffer=0.5 を見ると，Sent Drop，RTT Aware，Concentrated Drop の各方式は，平均 1525[s] で転送が完了している．

ビットエラー率が 10^{-7} のとき，パケットエラー率は $P_{per} = 8 \times 10^{-4}$ 程度であるため，P_{per} の 2 乗以上の項（2 回目以降の再送）を無視する．1 ファイル 10 パケットのうち 1 つでも再送が発生すれば，File Delivery Time は 1RTT だけ長くなり，その確率は，$(1 - P_{per})^{10} = 8 \times 10^{-3}$ である．Node0 からのファイルは 1200[s] と 300[s] のリンクを通るので，File Delivery Time の平均はエラーによって $2400 \times 8 \times 10^{-3} + 600 \times 8 \times 10^{-3} = 24[s]$ 長くなるはずであり，期待される File Delivery Time は 1500[s]+24[s]=1524[s] である．

以上のように，Sent Drop，RTT Aware，Concentrated Drop の各方式では溢れが起こらなかったときと File Delivery Time がほぼ同一になると言える．これは図 16 と同様に，バッファが溢れたとき，Sent File パケットから破棄することでバッファの空きを確保することができるためと考えられる．

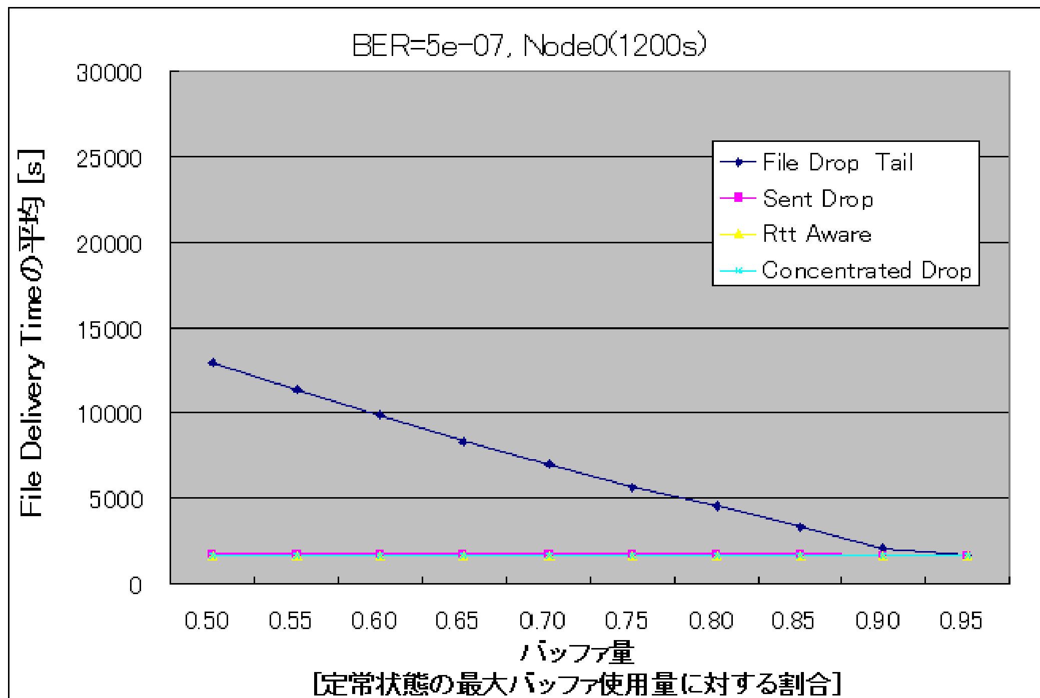

図 18: シミュレーション 1 の結果,（BER=5e-07，Node0）

図 18 では図 17 と同様に，エラーによる影響があるため，File Delivery Time の平均は全体的に長くなる．例として，Buffer=0.5 を見ると，RTT Aware 方式・Concentrated Drop 方式は，平均 1616[s] で転送が完了している．

ビットエラー率が 5×10^{-7} のとき，パケットエラー率は $P_{per} = 4 \times 10^{-3}$ 程度であるため，P_{per} の 2 乗以上の項（2 回目以降の再送）を無視する．1 ファイル 10 パケットのうち 1 つでも再送が発生すれば，File Delivery Time は 1RTT だけ長くなり，その確率は，$(1 - P_{per})^{10} = 4 \times 10^{-2}$ である．Node0 からのファイルは 1200[s] と 300[s] のリンクを通るので，File Delivery Time の平均はエラーによって $2400 \times 4 \times 10^{-2} + 600 \times 4 \times 10^{-2} = 118[s]$ 長くなるはずであり，期待される File Delivery Time は 1500[s]+118[s]=1618[s] である．

したがって，RTT Aware，Concentrated Drop の各方式では溢れが起こらなかったときと File Delivery Time がほぼ同一になると言える．これは図 16 と同様に，バッファが溢れたとき，Sent File パケットから破棄することでバッファの空きを確保することができるためと考えられる．また，Node0 は Node1 に対して RTT が長いため優先される．したがって，ほとんどの場合 Node1 から

のパケットから破棄されるため，Node0 からのパケットは溢れが無いときとほぼ同一になると考えられる．

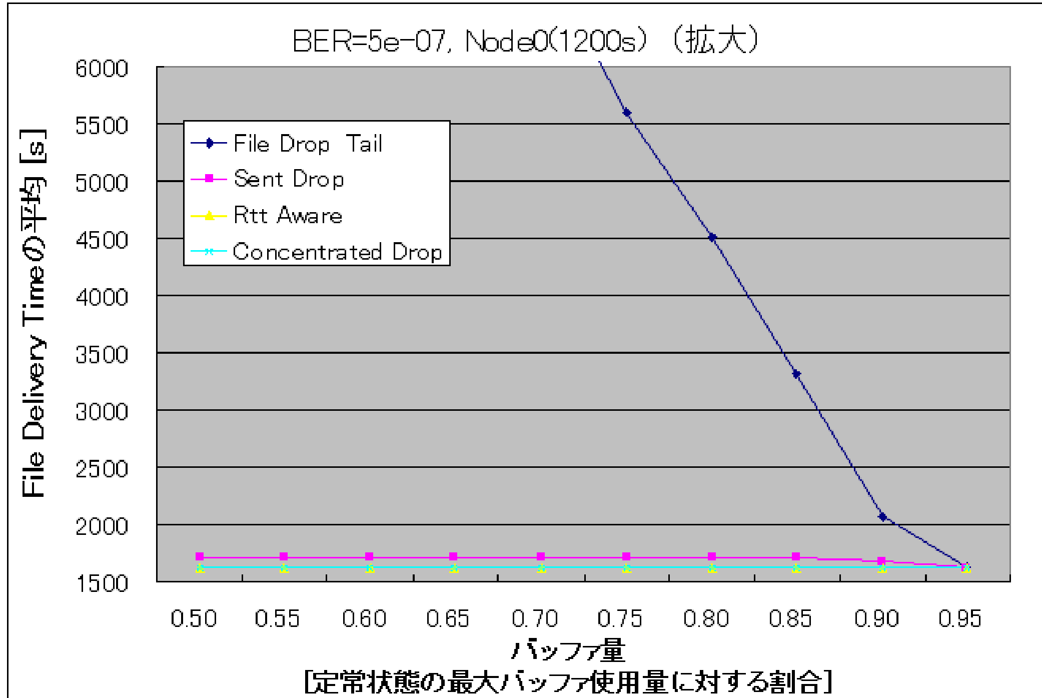

図 19: シミュレーション１の結果（拡大），(BER=5e-07，Node0)

図 19 は，図 18 の一部を拡大したものである．例として，Buffer=0.5 を見ると，Sent Drop 方式では平均 1709[s]，RTT Aware 方式，Concentrated Drop 方式は，平均 1616[s] で転送が完了している．このように，両者の間に 93[s] の差があることが分かる．

これは，Sent Drop 方式だけでは RTT に関係なく確率で破棄するパケットを決めるため，Node0 からのパケットも破棄され，Source Retransmission が発生するためである．Sent Drop 方式では 25,000 ファイル中 950 ファイルが Source Retransmission を起こし，2400[s] だけ File Delivery Time が長くなる．平均では $2400 \times \frac{950}{25000} = 92[s]$ だけ長くなるはずである．

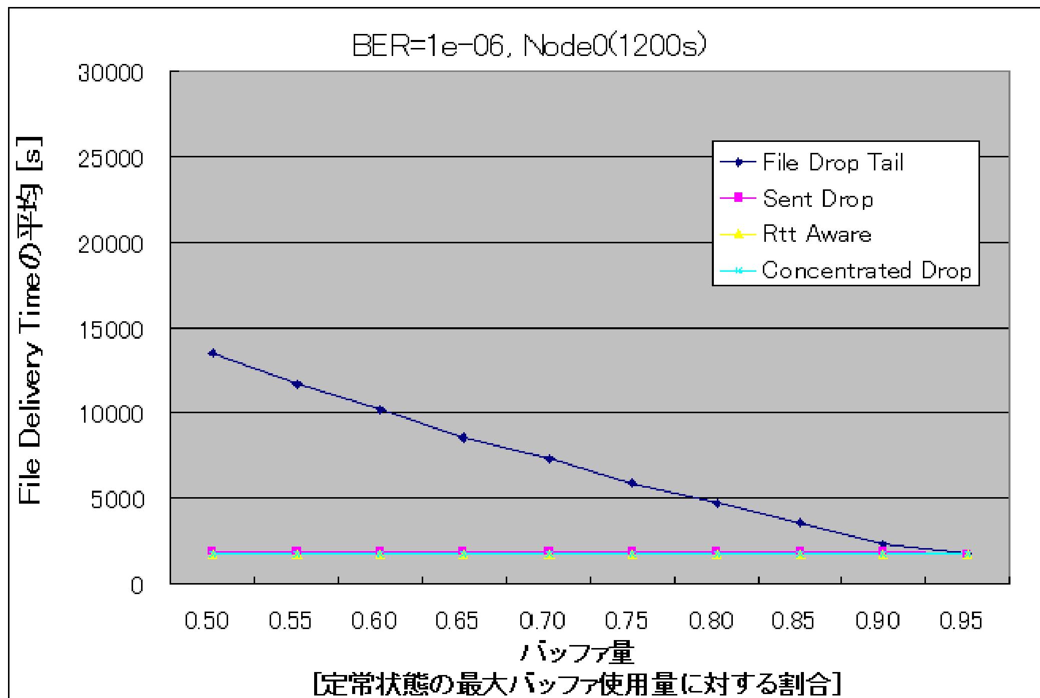

図 20: シミュレーション１の結果,（BER=1e-06，Node0）

図 20 では図 17 と同様に，エラーによる影響があるため，File Delivery Time の平均は全体的に長くなる．例として，Buffer=0.5 を見ると，RTT Aware 方式・Concentrated Drop 方式は，平均 1732[s] で転送が完了している．

ビットエラー率が 10^{-6} のとき，パケットエラー率は $P_{per} = 8 \times 10^{-3}$ 程度であるため，P_{per} の 2 乗以上の項（2 回目以降の再送）を無視する．1 ファイル 10 パケットのうち 1 つでも再送が発生すれば，File Delivery Time は 1RTT だけ長くなり，その確率は，$(1 - P_{per})^{10} = 8 \times 10^{-2}$ である．Node0 からのファイルは 1200[s] と 300[s] のリンクを通るので，File Delivery Time の平均はエラーによって $2400 \times 8 \times 10^{-2} + 600 \times 8 \times 10^{-2} = 232[s]$ 長くなるはずであり，期待される File Delivery Time は 1500[s]+232[s]=1732[s] である．

したがって，RTT Aware，Concentrated Drop の各方式では溢れが起こらなかったときと File Delivery Time がほぼ同一になると言える．これは図 16 と同様に，バッファが溢れたとき，Sent File パケットから破棄することでバッファの空きを確保することができるためと考えられる．また，Node0 は Node1 に対して RTT が長いため優先される．したがって，ほとんどの場合 Node1 から

のパケットから破棄されるため，Node0 からのパケットは溢れが無いときとほぼ同一になると考えられる．

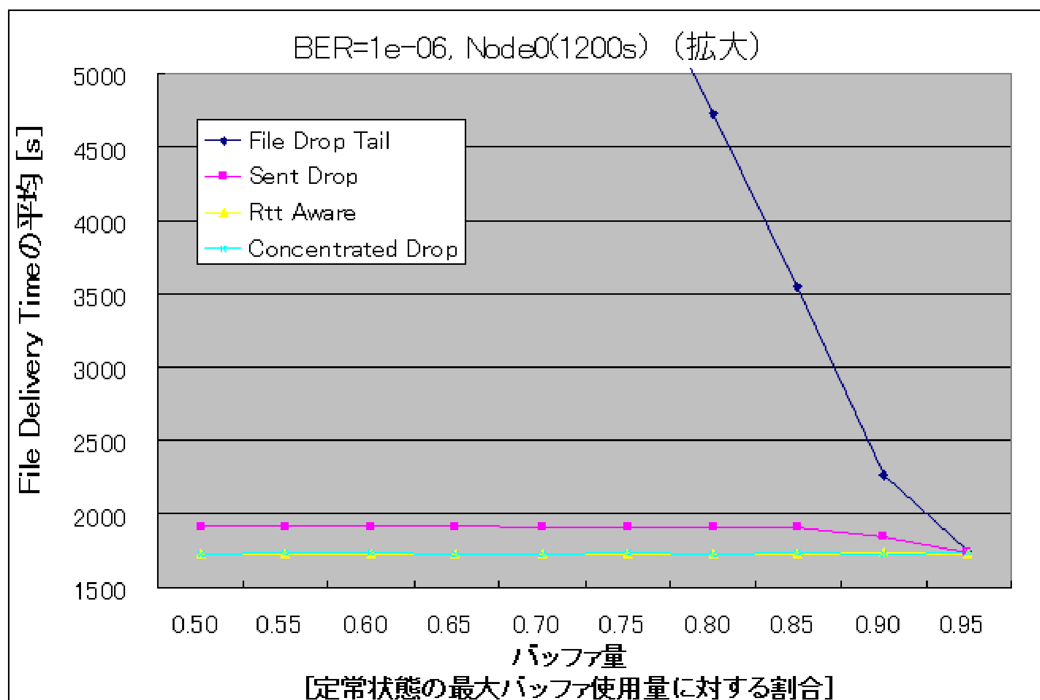

図 21: シミュレーション１の結果（拡大），(BER=1e-06，Node0）

図 21 は，図 20 の一部を拡大したものである．例として，Buffer=0.5 を見ると，Sent Drop 方式では平均 1732[s]，RTT Aware 方式，Concentrated Drop 方式は，平均 1917[s] で転送が完了している．このように，両者の間に 185[s] の差があることが分かる．

これは，Sent Drop 方式だけでは RTT に関係なく確率で破棄するパケットを決めるため，Node0 からのパケットも破棄され，Source Retransmission が発生するためである．Sent Drop 方式では 25,000 ファイル中 1,870 ファイルが Source Retransmission を起こし，2400[s] だけ File Delivery Time が長くなる．平均では $2400 \times \frac{1870}{25000} = 180[s]$ だけ長くなるはずである．

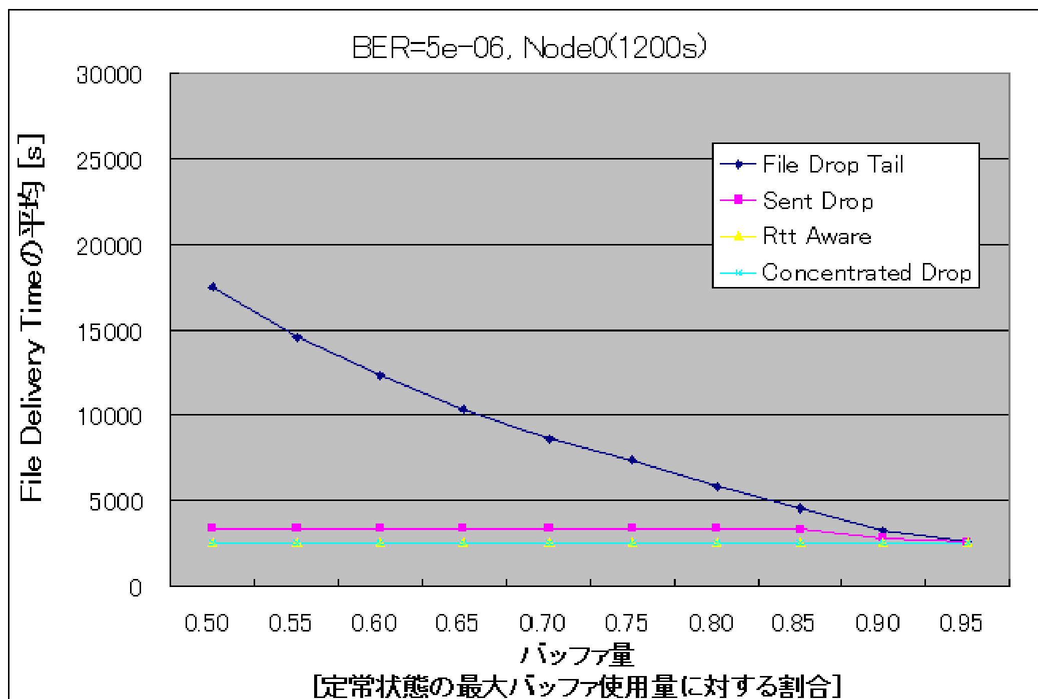

図 22: シミュレーション１の結果,（BER=5e-06，Node0）

図 22 では図 17 と同様に，エラーによる影響があるため，File Delivery Time の平均は全体的に長くなる．例として，Buffer=0.5 を見ると，RTT Aware 方式・Concentrated Drop 方式は，平均 2534[s] で転送が完了している．

ビットエラー率が 5×10^{-6} のとき，パケットエラー率は $P_{per} = 4 \times 10^{-2}$ 程度であるため，P_{per} の 2 乗以上の項（2 回目以降の再送）を無視する．1 ファイル 10 パケットのうち 1 つでも再送が発生すれば，File Delivery Time は 1RTT だけ長くなり，その確率は，$(1 - P_{per})^{10} = 0.34$ である．Node0 からのファイルは 1200[s] と 300[s] のリンクを通るので，File Delivery Time の平均はエラーによって $2400 \times 0.34 + 600 \times 0.34 = 1020[s]$ 長くなるはずであり，期待される File Delivery Time は 1500[s]+1020[s]=2520[s] である．

したがって，RTT Aware，Concentrated Drop の各方式では溢れが起こらなかったときと File Delivery Time がほぼ同一になると言える．これは図 16 と同様に，バッファが溢れたとき，Sent File パケットから破棄することでバッファの空きを確保することができるためと考えられる．また，Node0 は Node1 に対して RTT が長いため優先される．したがって，ほとんどの場合 Node1 から

のパケットから破棄されるため，Node0 からのパケットは溢れが無いときとほぼ同一になると考えられる．

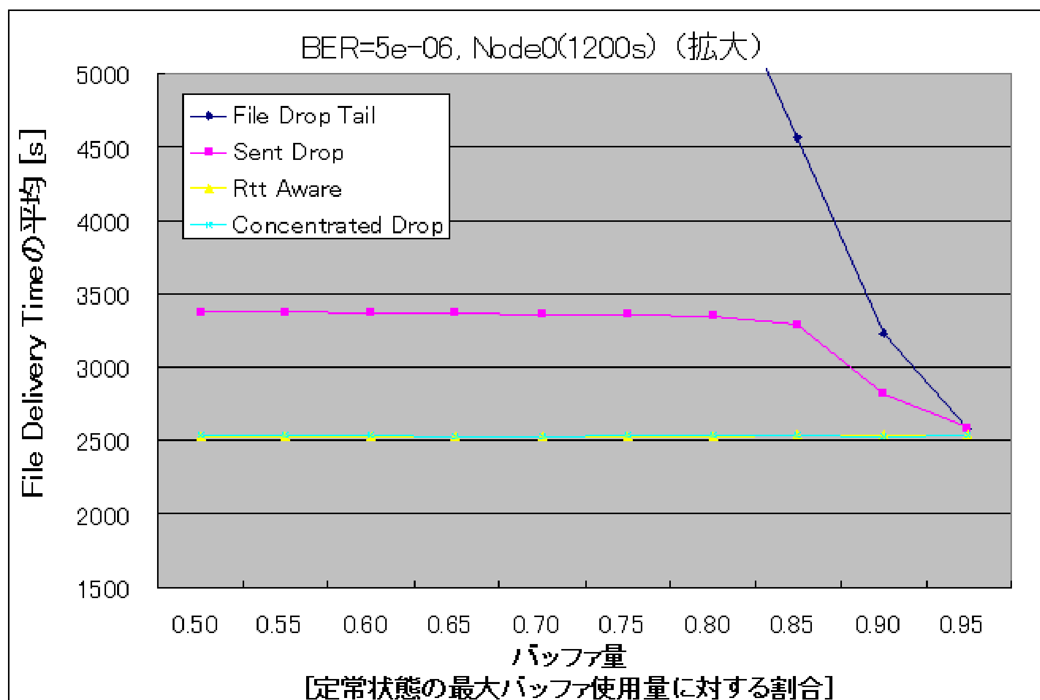

図 23: シミュレーション１の結果（拡大），（BER=5e-06，Node0）

図 23 は，図 22 の一部を拡大したものである．例として，Buffer=0.5 を見ると，Sent Drop 方式では平均 3333[s]，RTT Aware 方式，Concentrated Drop 方式は，平均 2534[s] で転送が完了している．このように，両者の間に 799[s] の差があることが分かる．

これは，Sent Drop 方式だけでは RTT に関係なく確率で破棄するパケットを決めるため，Node0 からのパケットも破棄され，Source Retransmission が発生するためである．Sent Drop 方式では 25,000 ファイル中 8,170 ファイルが Source Retransmission を起こし，2400[s] だけ File Delivery Time が長くなる．平均では $2400 \times \frac{8170}{25000} = 784[s]$ だけ長くなるはずである．

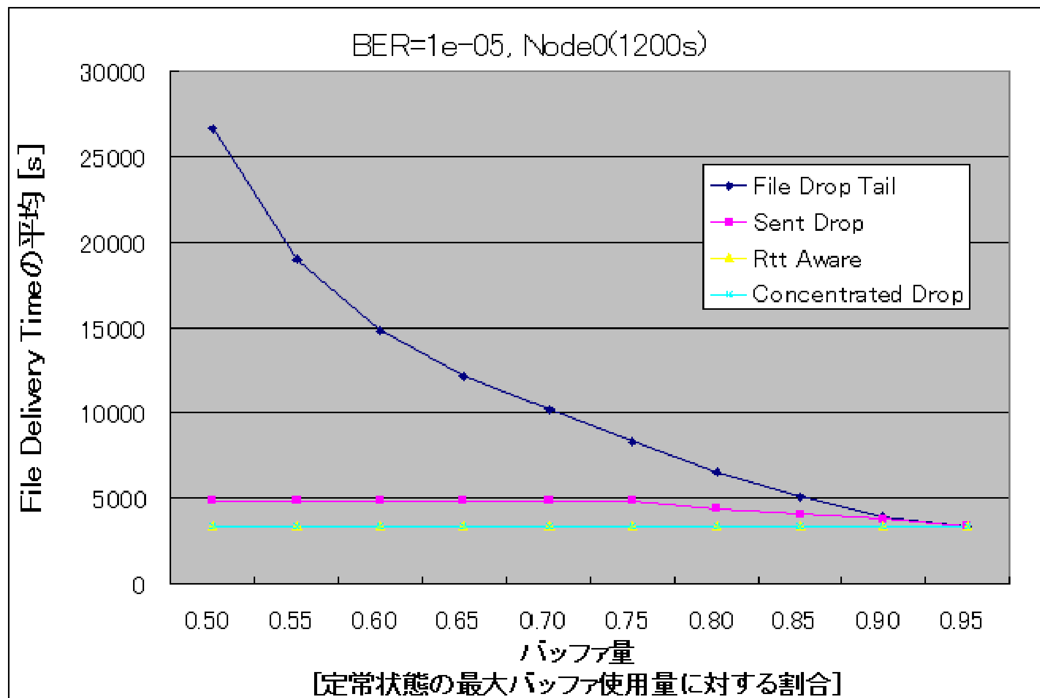

図 24: シミュレーション１の結果,（BER=1e-05，Node0）

図 24 では図 17 と同様に，エラーによる影響があるため，File Delivery Time の平均は全体的に長くなる．例として，Buffer=0.5 では，RTT Aware 方式・Concentrated Drop 方式は，平均 3240[s] で転送が完了している．

ビットエラー率が 10^{-5} のとき，パケットエラー率は $P_{per} = 8 \times 10^{-2}$ 程度であるため，P_{per} の 2 乗以上の項（2 回目以降の再送）を無視する．1 ファイル 10 パケットのうち 1 つでも再送が発生すれば，File Delivery Time は 1RTT だけ長くなり，その確率は，$(1 - P_{per})^{10} = 0.58$ である．Node0 からのファイルは 1200[s] と 300[s] のリンクを通るので，File Delivery Time の平均はエラーによって $2400 \times 0.58 + 600 \times 0.58 = 1743[s]$ 長くなるはずであり，期待される File Delivery Time は 1500[s]+1743[s]=3243[s] である．

したがって，RTT Aware，Concentrated Drop の各方式では溢れが起こらなかったときと File Delivery Time がほぼ同一になると言える．これは図 16 と同様に，バッファが溢れたとき，Sent File パケットから破棄することでバッファの空きを確保することができるためと考えられる．また，Node0 は Node1 に対して RTT が長いため優先される．したがって，ほとんどの場合 Node1 から

のパケットから破棄されるため，Node0 からのパケットは溢れが無いときとほぼ同一になると考えられる．

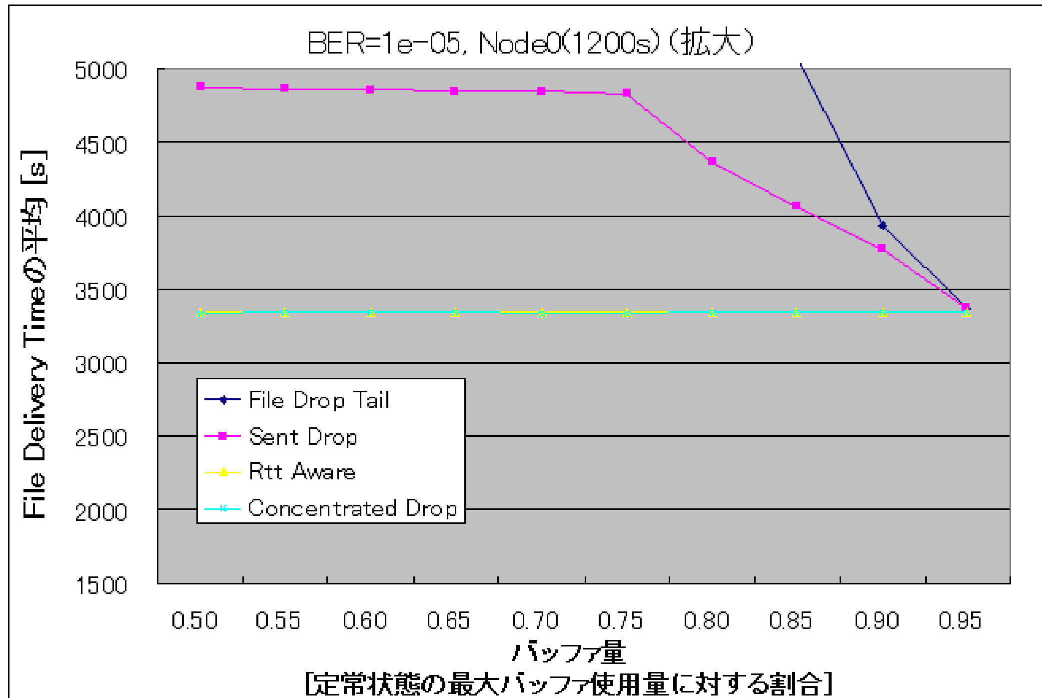

図 25: シミュレーション１の結果（拡大），（BER=1e-05，Node0）

図25は，図24の一部を拡大したものである．Sent Drop方式では平均4785[s], RTT Aware 方式，Concentrated Drop 方式は，平均 3240[s] で転送が完了している．このように，両者の間に 1545[s] の差があることが分かる．

これは，Sent Drop 方式だけではRTTに関係なく確率で破棄するパケットを決めるため，Node0 からのパケットも破棄され，Source Retransmission が発生するためである．Sent Drop 方式では 25,000 ファイル中 16,398 ファイルが Source Retransmission を起こし，2400[s] だけ File Delivery Time が長くなる．平均では $2400 \times \frac{16398}{25000} = 1574[s]$ だけ長くなるはずである．

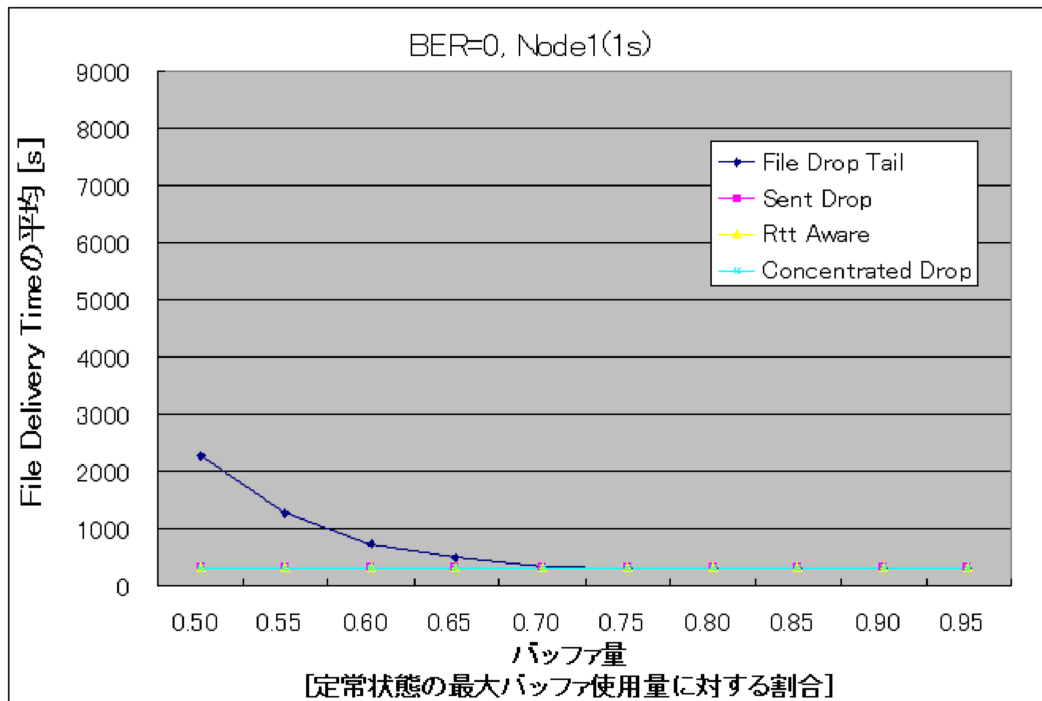

図 26: シミュレーション１の結果，（BER=0，Node1）

図26では図16と同様に，File Drop Tail 方式ではバッファ量が小さくなるにつれて File Delivery Time の平均が長くなる．Sent Drop，RTT Aware，Concentrated Drop の各方式はバッファの溢れが無いときと等しくなる．また，エラーが無いので Source Retransmission によって File Delivery Time が長くなることもない．

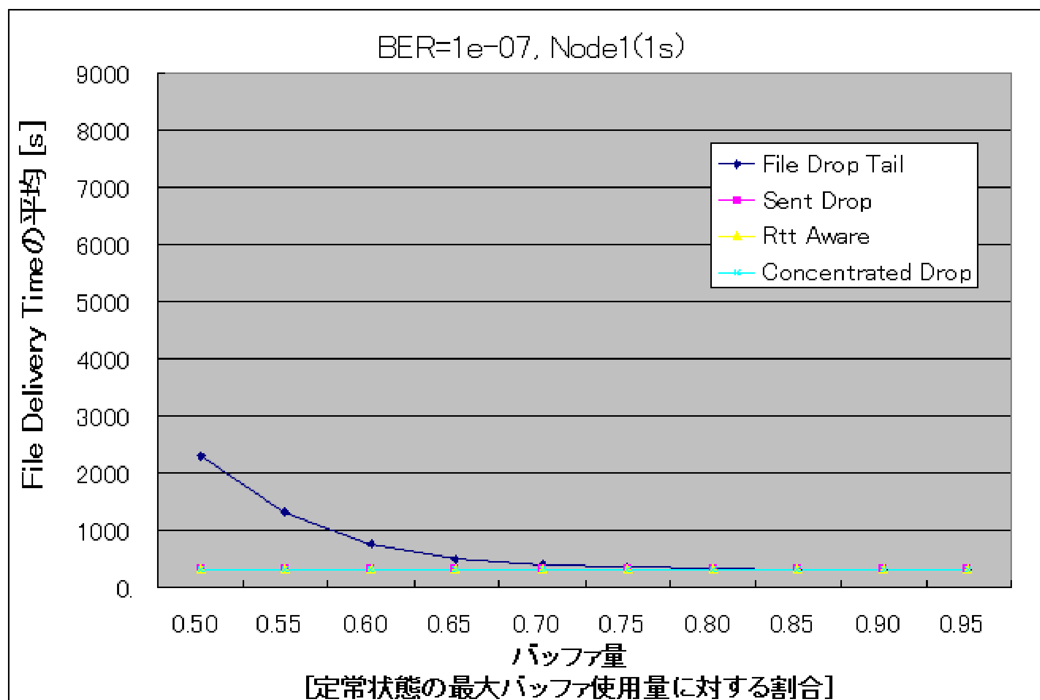

図 27: シミュレーション１の結果,（BER=1e-07，Node1）

図 27 は，図??とは逆で，Node1 は Node0 に対して RTT が短いため優先して破棄される．そのため，多くの場合 Node1 からのパケットが破棄されるが，Node1 への Source Retransmission を経験したファイル数は 200 で RTT が 2[s] であるから，全体の File Delivery Time に対して 0.005 %の影響しかない．

このことから，RTT Aware 方式を採用すると，再送が発生しても，多くの場合短い RTT の再送制御となるため，File Delivery Time を短縮すると言える．

以下，図 28〜図 31 まで同様である．

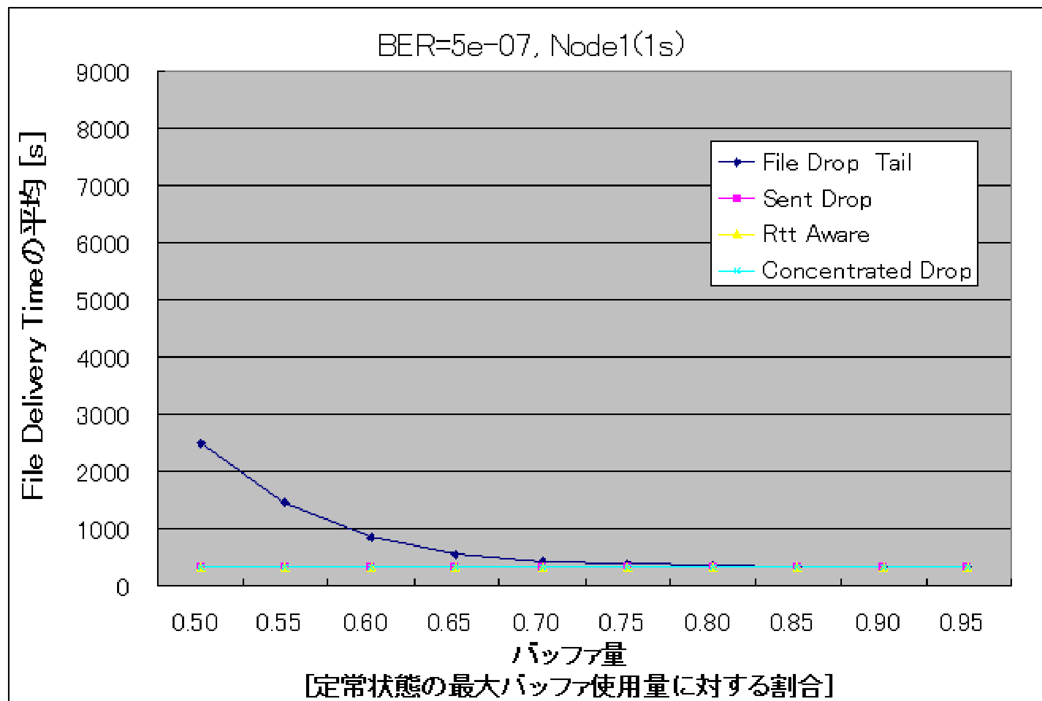

図 28: シミュレーション１の結果,（BER=5e-07，Node1）

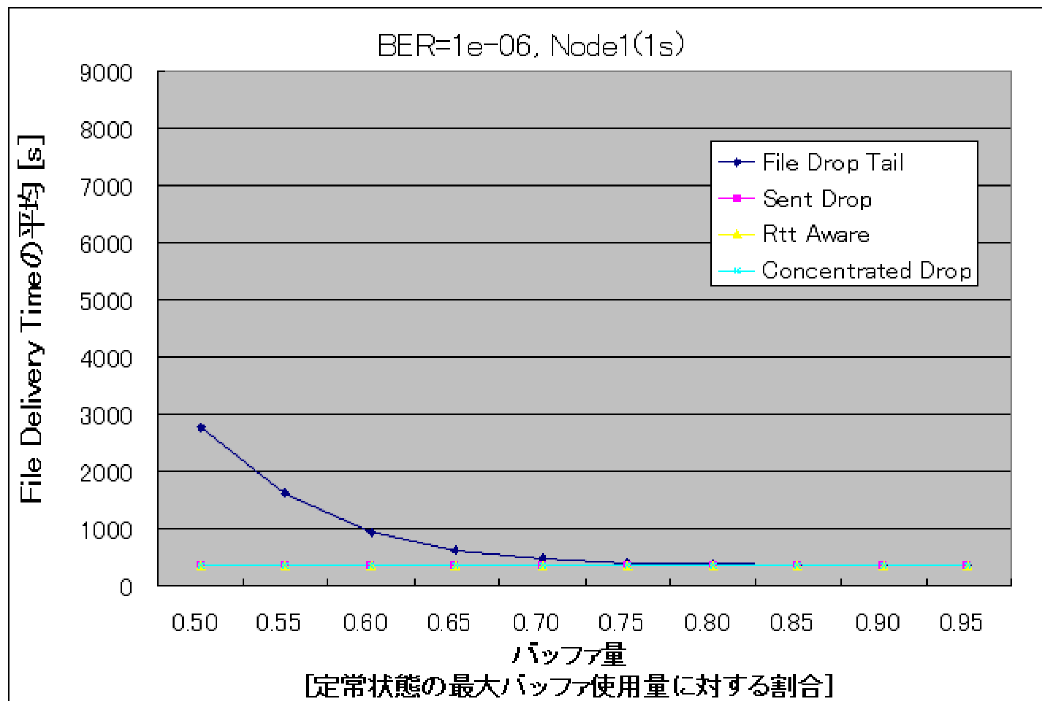

図 29: シミュレーション１の結果,（BER=1e-06，Node1）

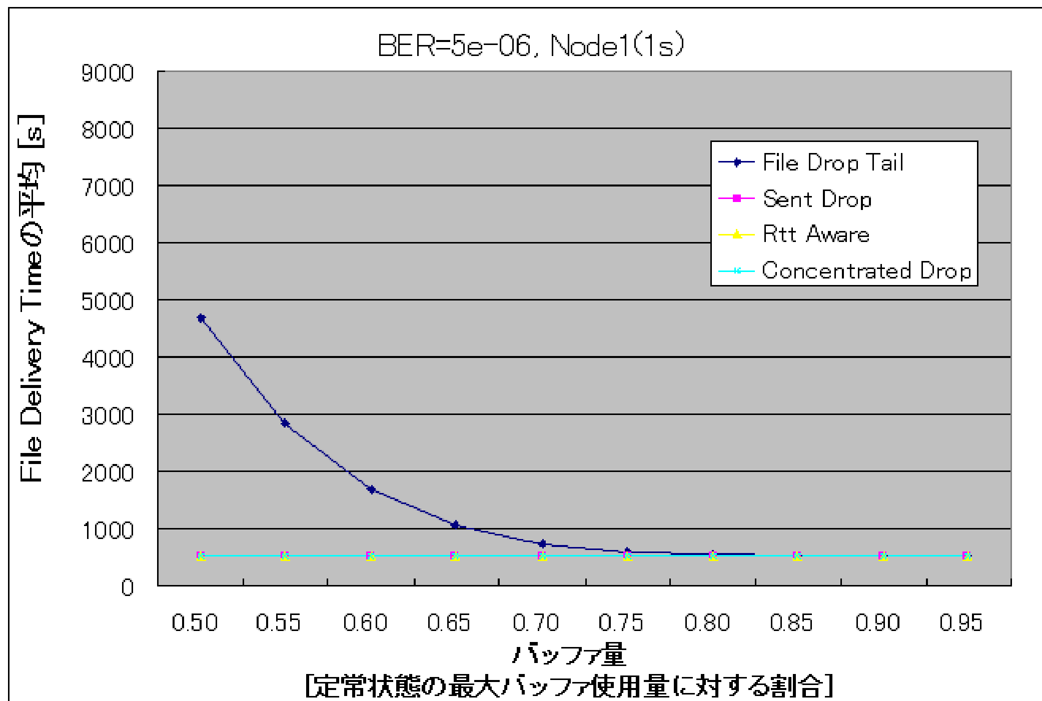

図 30: シミュレーション１の結果,（BER=5e-06，Node1）

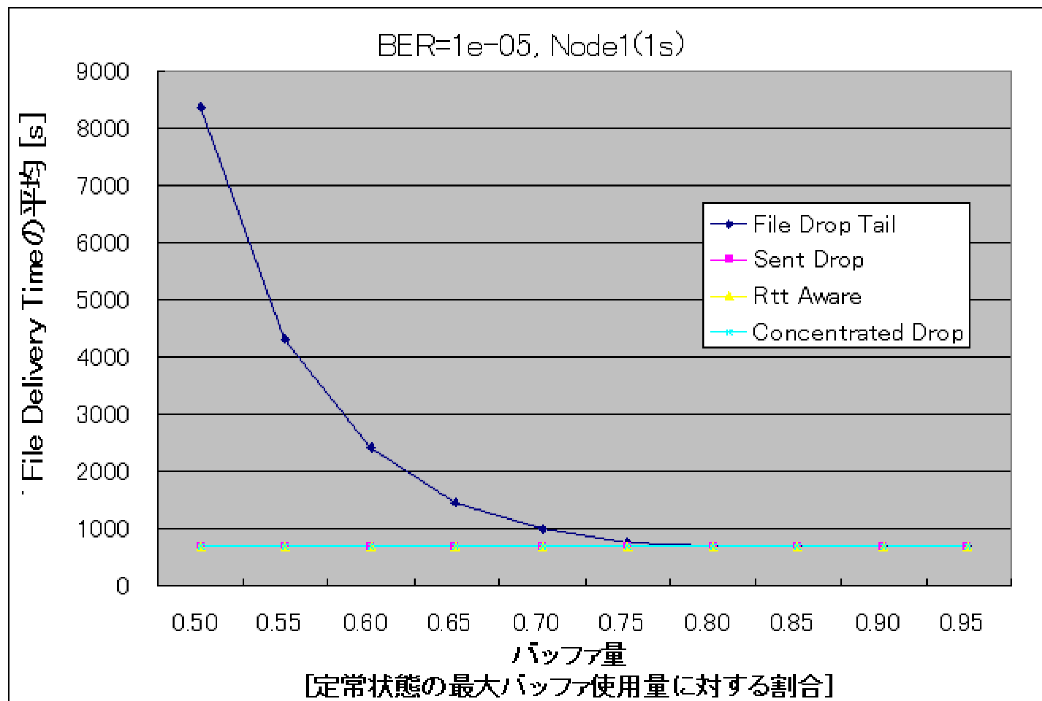

図 31: シミュレーション１の結果,（BER=1e-05，Node1）

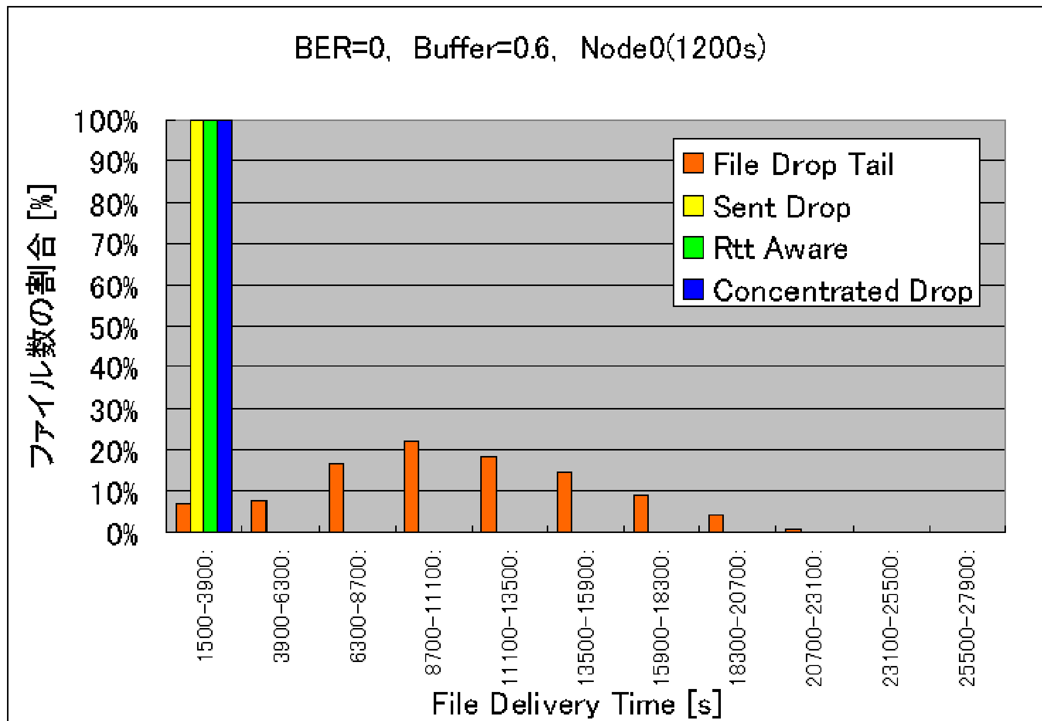

図 32: File Delivery Time の分布（Node0, BER=0, Buffer=0.6）

図 32 では，File Drop Tail 方式だけ File Delivery Time が分散している．これは，バッファ溢れによってパケットを再送する必要が発生するためである．File Drop Tail 以外の方式では，Sent File を破棄してもエラーがないので Source Retransmission は一度も発生せずに済むため，全てのファイルがほぼ最小の File Delivery Time（1500[s]）で届くことになる．

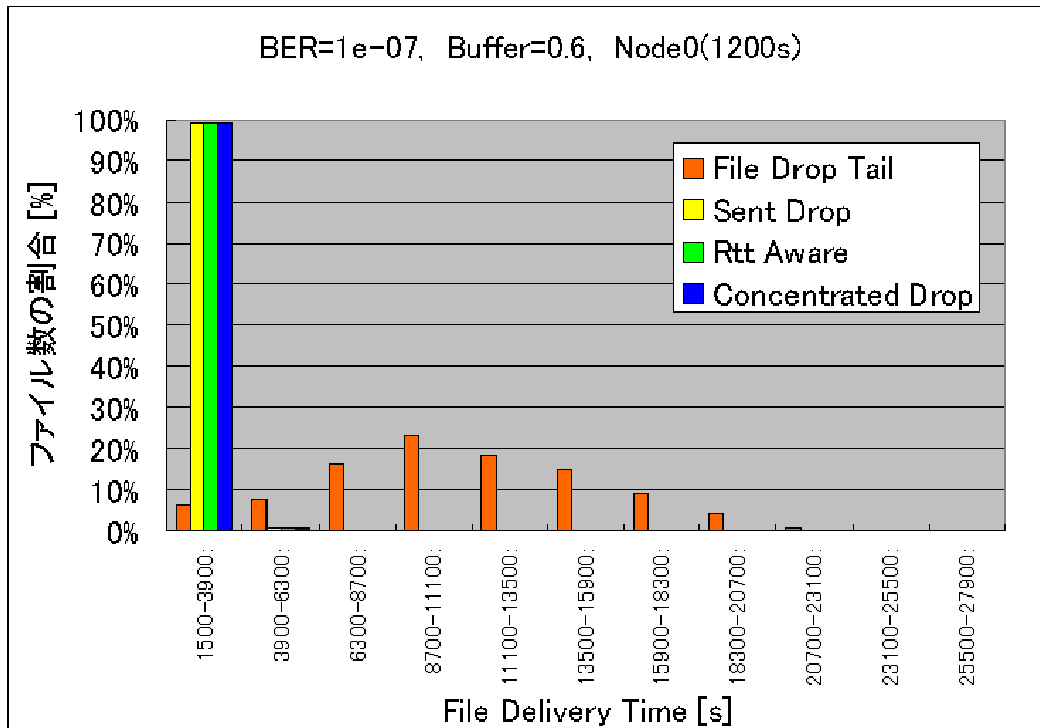

図 33: File Delivery Time の分布（Node0, BER=1e-07, Buffer=0.6）

図 33 では，Sent Drop 方式，RTT Aware 方式，Concentrated Drop 方式についても，エラーによって再送が発生するので最小の File Delivery Time（1500[s]）では到着できないファイルが存在する．

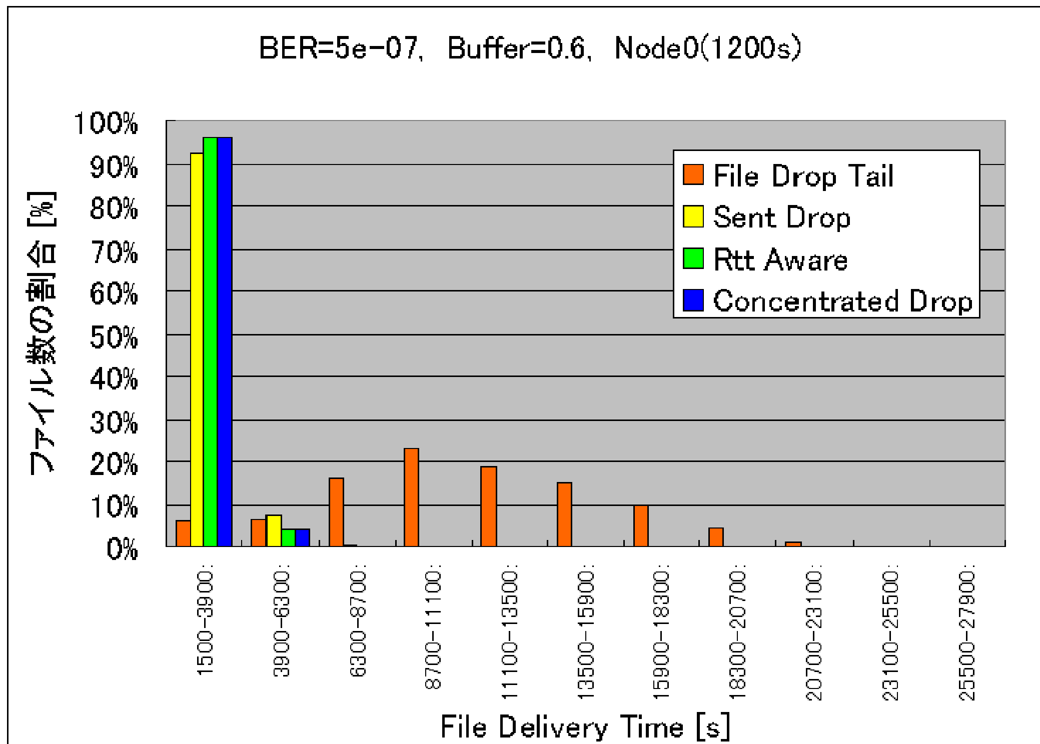

図 34: File Delivery Time の分布（Node0, BER=5e-07, Buffer=0.6）

図 34 では，Sent Drop 方式，RTT Aware 方式，Concentrated Drop 方式についても，エラーによって再送が発生するので最小の File Delivery Time（1500s）では到着できないファイルが存在する．図 33 と比較すると，エラー率が大きくなった分，同じパケットで再送を何度も繰り返すことが発生し，一度の再送で済まない場合があり，最小の File Delivery Time の 5 倍（再送が 2 回）以上かかるファイルが出てくる．

以下，図 35〜図 37 まで同様である．

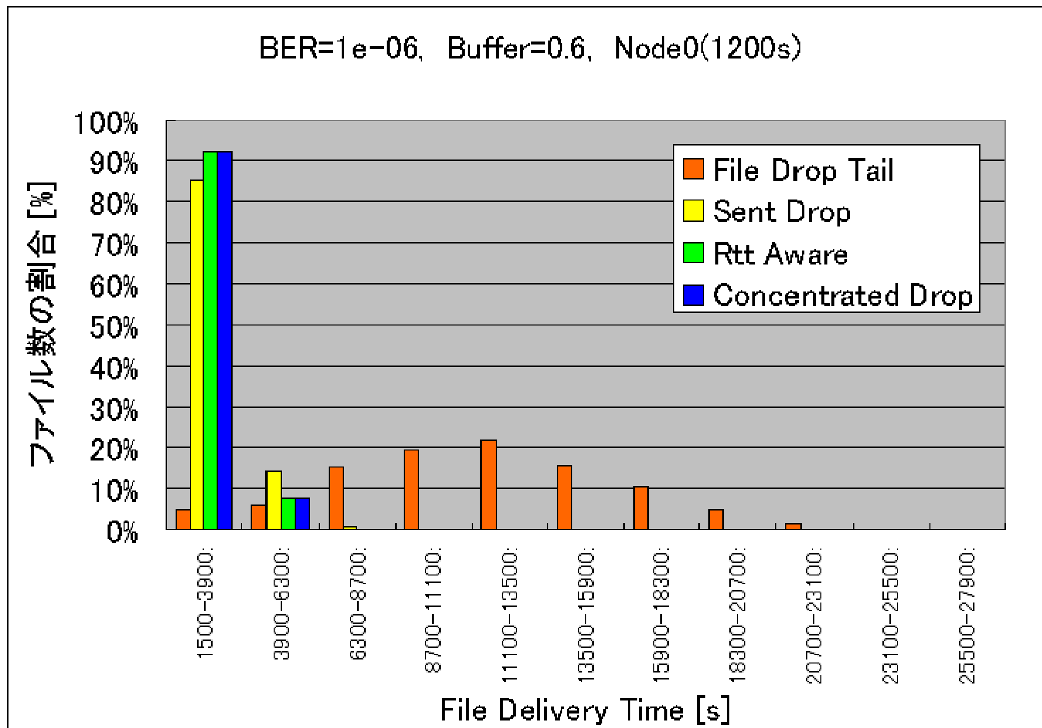

図 35: File Delivery Time の分布（Node0, BER=1e-06, Buffer=0.6）

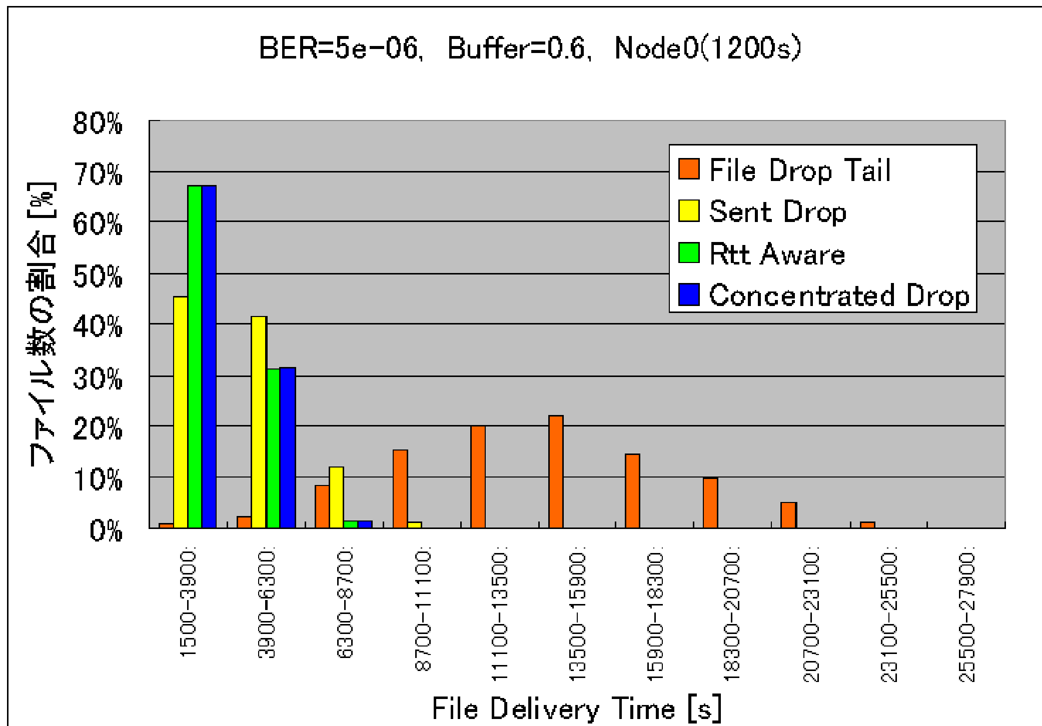

図 36: File Delivery Time の分布（Node0, BER=5e-06, Buffer=0.6）

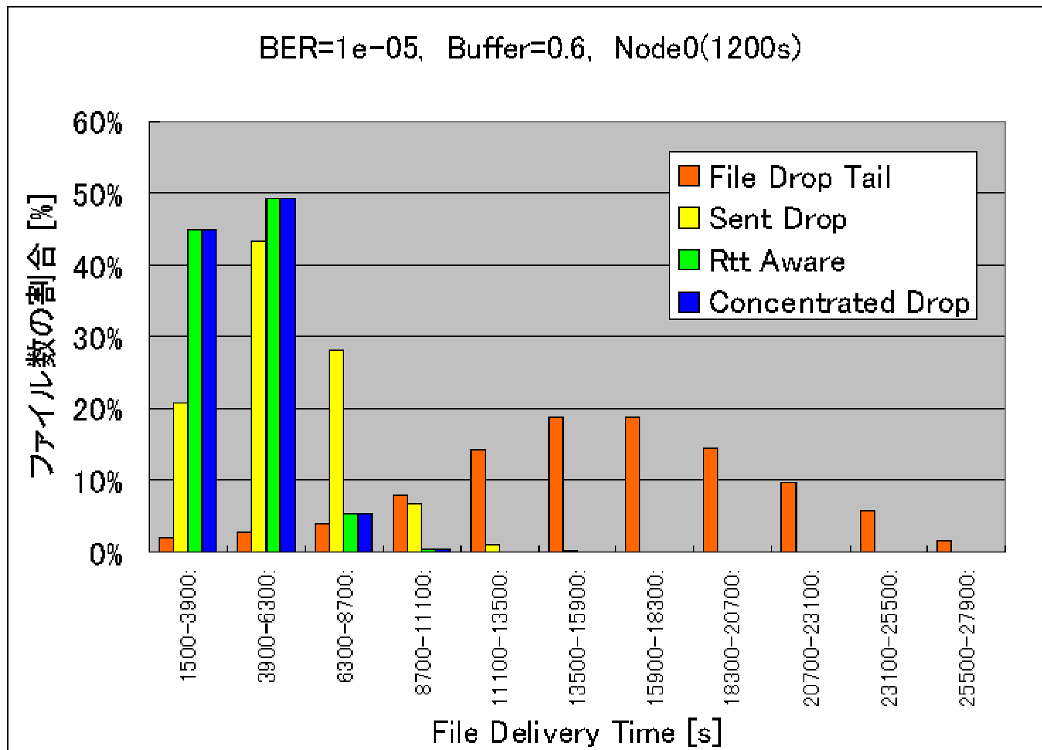

図 37: File Delivery Time の分布（Node0, BER=1e-05, Buffer=0.6）

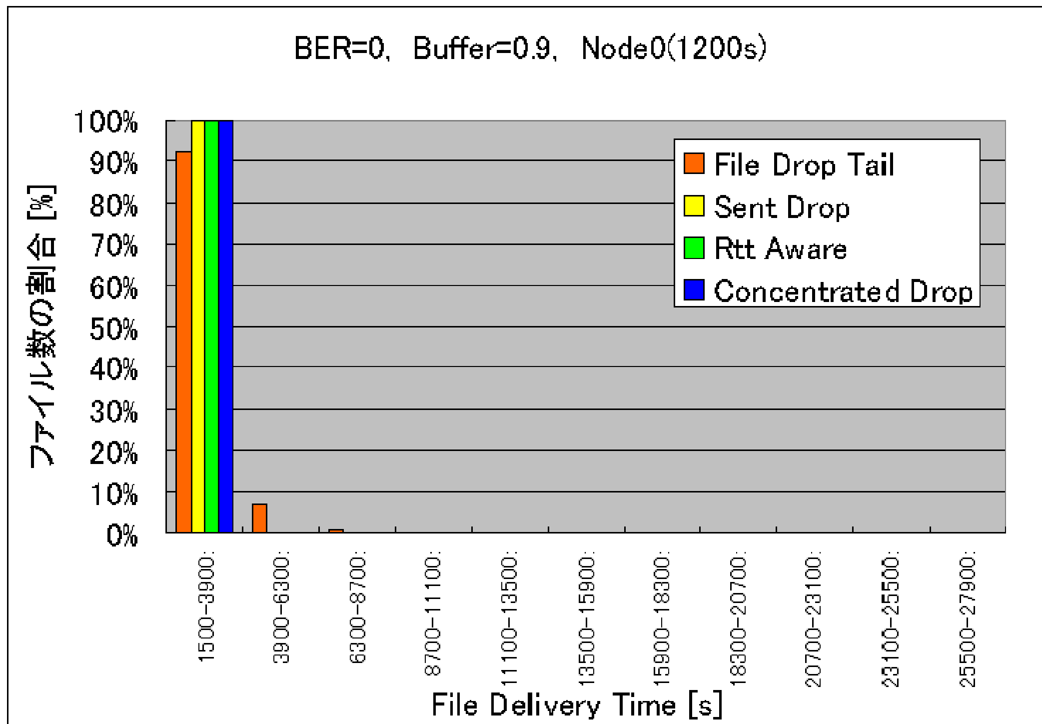

図 38: File Delivery Time の分布（Node0, BER=0, Buffer=0.9）

図 38 では，図 32 と比較すると File Drop Tail 方式の分散が小さくなっている．これは，図 32 に比べてバッファが大きくなり，バッファ溢れによるパケットの再送が少なくなったものと考えられえる．File Drop Tail 以外の方式は，図 32 と同様である．

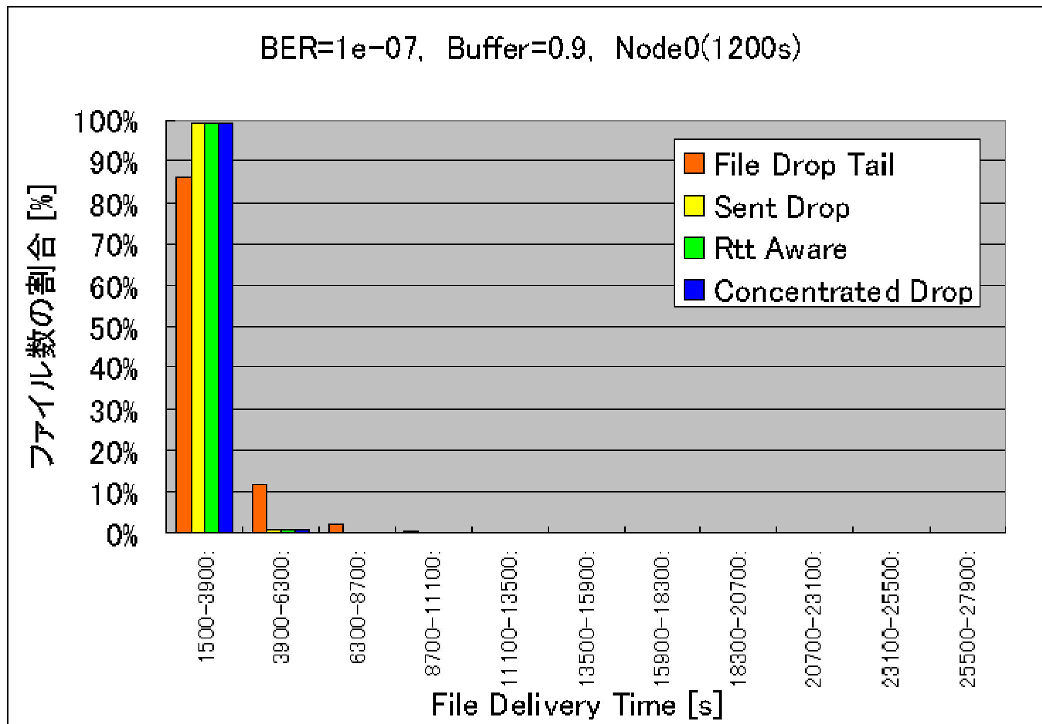

図 39: File Delivery Time の分布（Node0, BER=1e-07, Buffer=0.9）

図 39 では，Sent Drop 方式，RTT Aware 方式，Concentrated Drop 方式についても，エラーによって再送が発生するので最小の File Delivery Time（1500s）では到着できないファイルが存在する．また，図 33 と比較してバッファが大きく，溢れることが少ないため，File Drop Tail 方式も同じような分散となる．

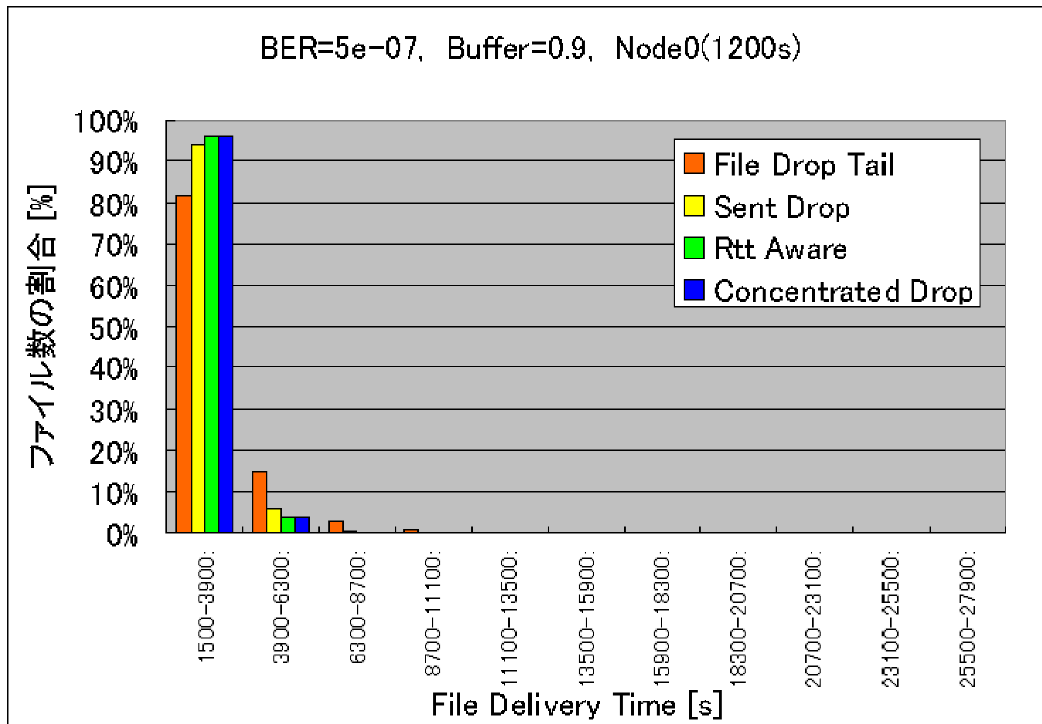

図 40: File Delivery Time の分布（Node0, BER=5e-07, Buffer=0.9）

図 40 では，Sent Drop 方式，RTT Aware 方式，Concentrated Drop 方式についても，エラーによって再送が発生するので最小の File Delivery Time（1500s）では到着できないファイルが存在する．図 34 と比較すると，エラー率が大きくなった分，同じパケットで再送を何度も繰り返すことが発生し，一度の再送で済まない場合があり，最小の File Delivery Time の 5 倍（再送が 2 回）以上かかるファイルが出てくる．また，図 34 と比較してバッファが大きく，溢れることが少ないため，File Drop Tail 方式も同じような分散となる．

以下，図 41〜図 43 まで同様である．

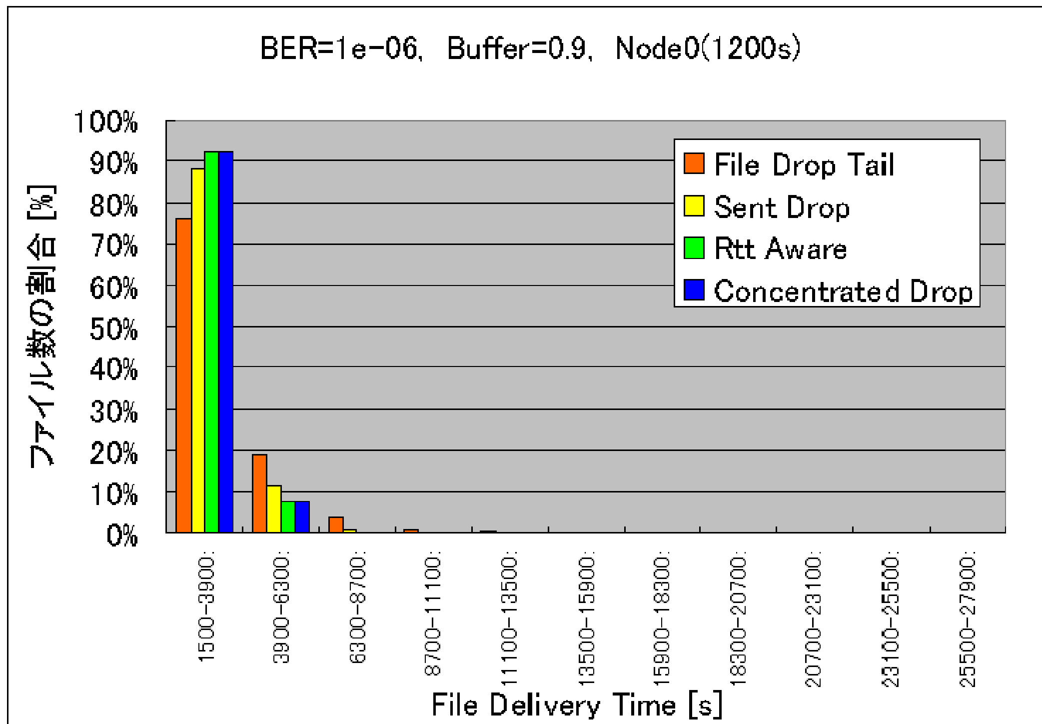

図 41: File Delivery Time の分布（Node0, BER=1e-06, Buffer=0.9）

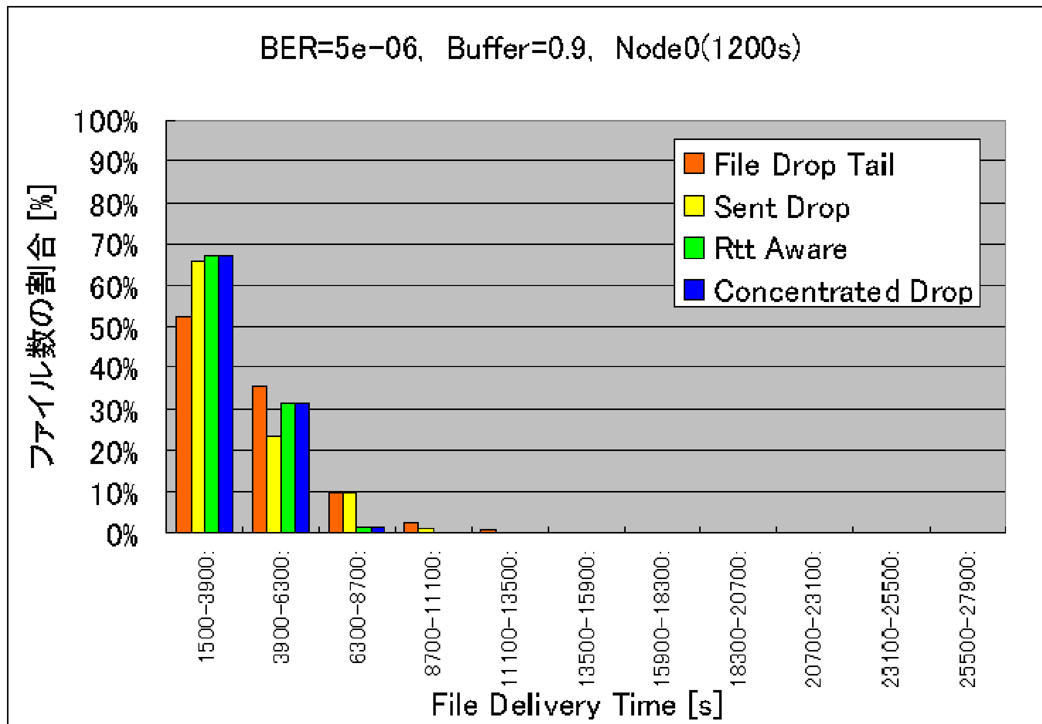

図 42: File Delivery Time の分布（Node0, BER=5e-06, Buffer=0.9）

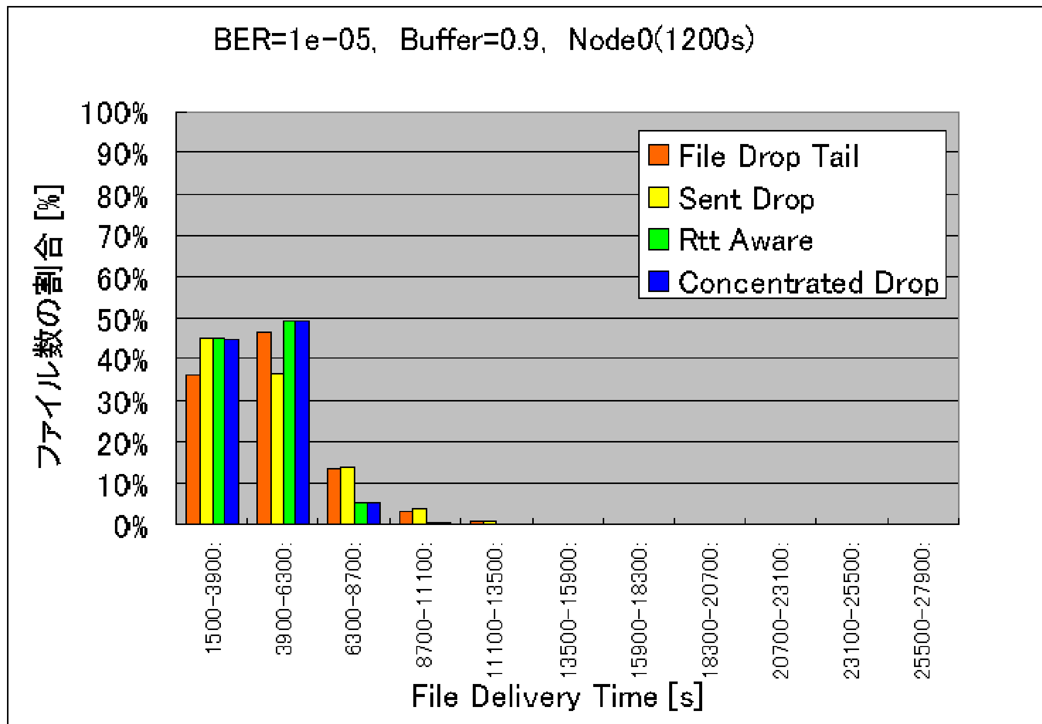

図 43: File Delivery Time の分布（Node0, BER=1e-05, Buffer=0.9）

シミュレーション2

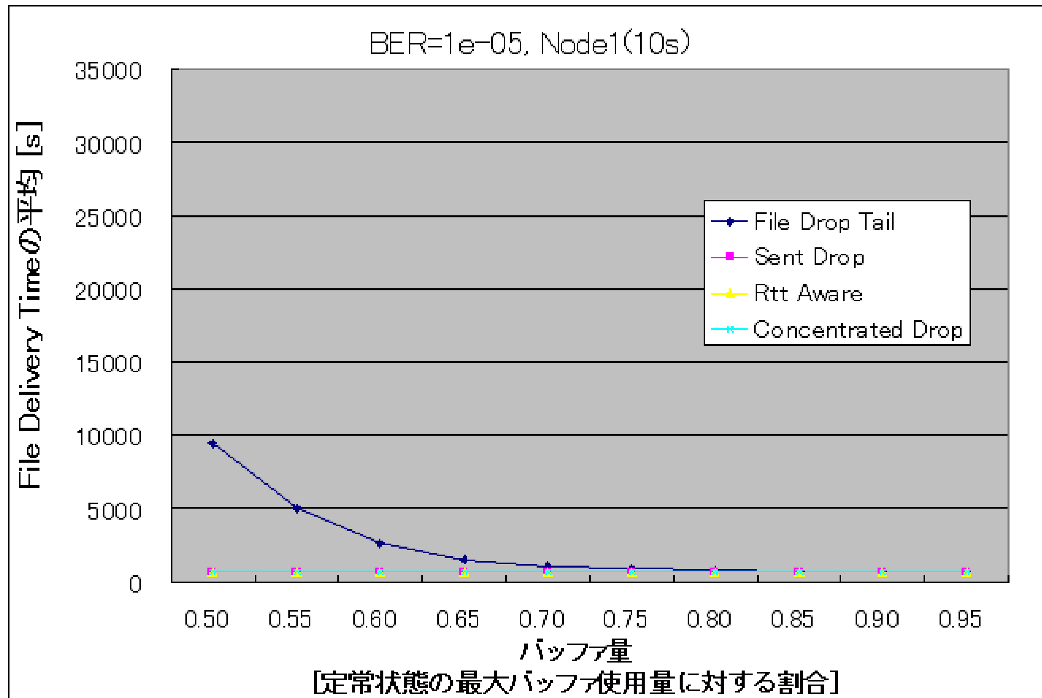

図 44: シミュレーション 2 の結果,（Node1, 1200s-10s）

図 44 で，Node1 は Node0 に対して RTT が短いため優先して破棄される．そのため，多くの場合 Node1 からのパケットが破棄される．例として，Buffer=0.5 を見ると，シミュレーションでは 25,000 ファイル中，13,668 ファイルが Source Retransmission を経験した．しかし，RTT が 20[s] のため，全体の File Delivery Time には 1.5 %程度の影響で済む．これは RTT Aware 方式と Concentrated Drop 方式の差も同様である．これまでのシミュレーションでは，RTT Aware 方式と Concentrated Drop 方式に顕著な差が無かったが，これは Sent File パケットを積極的に破棄される短い方の RTT が 2[s] であり，File Delivery Time の平均では 0.1 %程度の差にしかならないためである．

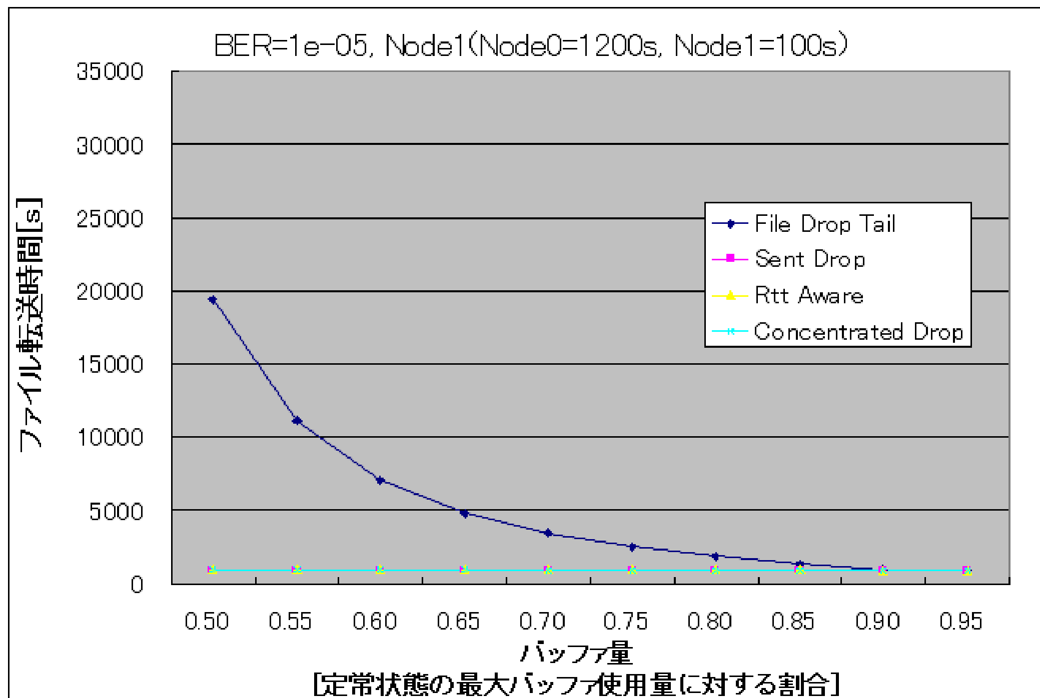

図 45: シミュレーション 2 の結果,（Node1, 1200s-100s）

図 45 で，Node1 は Node0 に対して RTT が短いため優先して破棄される．そのため，多くのの場合 Node1 からのパケットが破棄される．Node1 への Source Retransmission が発生すると，RTT は 200[s] で File Delivery Time に影響を与える．RTT の短い Node1 は優先して破棄される方なので，RTT Aware 方式を用いると Sent Drop 方式よりも File Delivery Time は長くなる．

例として，Buffer=0.5 を見る．シミュレーションでは，Sent Drop 方式の Source Retransmission を経験したファイルが 308 個あったのに対して，RTT Aware 方式は 13,612 個であった．Sent Drop 方式から RTT Aware 方式にすることによって，25,000 ファイル中 13,304 ファイルが新たに Source Retransmission するため，平均では $200 \times \frac{13304}{25000} = 106[s]$ だけ長くなる．

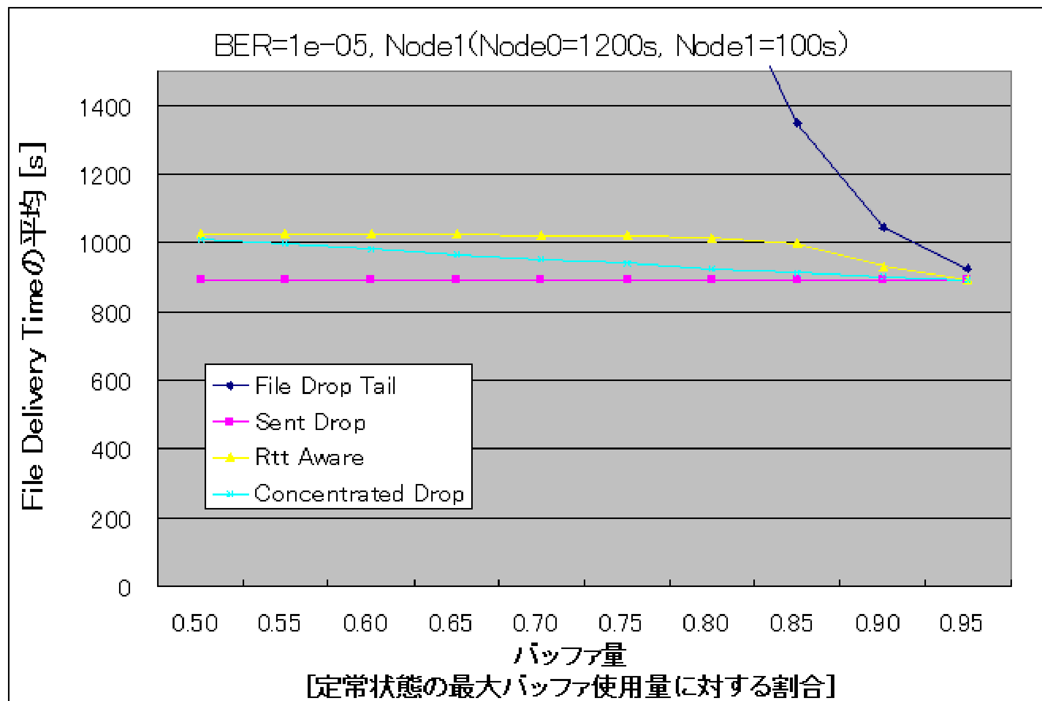

図 46: シミュレーション２の結果（拡大），(Node1, 1200s-100s)

図 46 は，図 45 の一部を拡大したものである．Concentrated Drop 方式は，RTT Aware 方式よりも File Delivery Time が短い．例として Buffer=0.7 を見ると，RTT Aware 方式で Source Retransmission を経験したファイル数は 13,289 であったのに対して，Concentrated Drop 方式では 6,904 であった．したがって，RTT Aware 方式から Concentrated Drop 方式にすることによって，25,000 ファイル中 6,385 ファイルが Source Retransmission せずに済むため，平均では $200 \times \frac{6385}{25000} = 51[s]$ だけ短くなる．

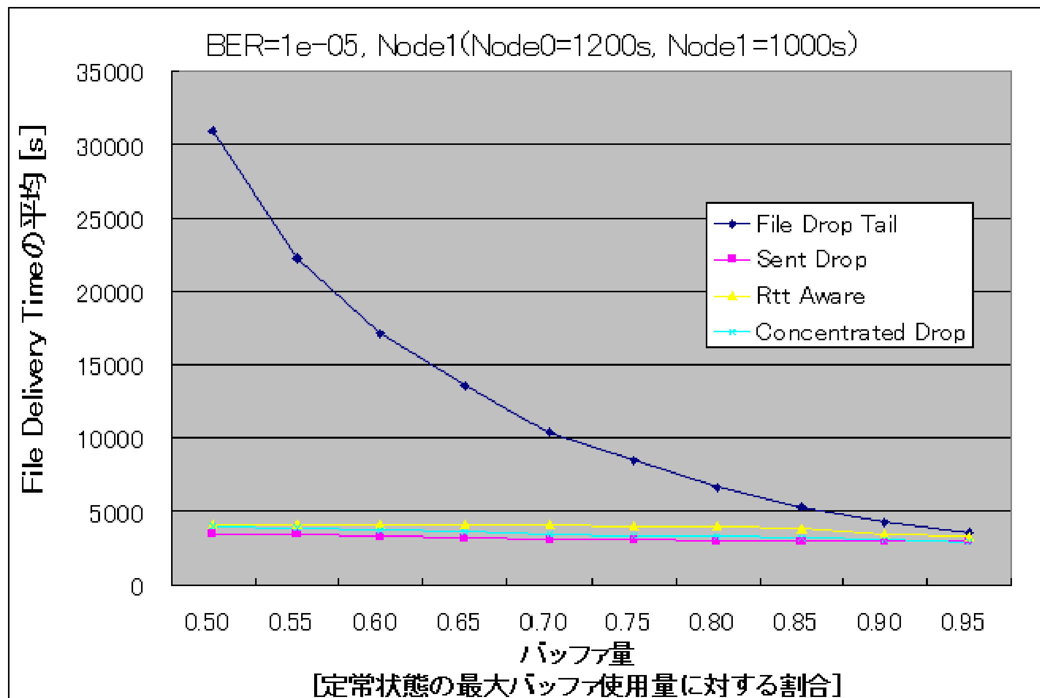

図 47: シミュレーション２の結果,（Node1, 1200s-1000s）

図 47 で，Node1 は Node0 に対して RTT が短いため優先して破棄される．そのため，多くのの場合 Node1 からのパケットが破棄される．Node1 への Source Retransmission が発生すると，RTT は 2000[s] で File Delivery Time に影響を与える．RTT の短い Node1 は優先して破棄される方なので，RTT Aware 方式を用いると Sent Drop 方式よりも File Delivery Time は長くなる．

例として，Buffer=0.5 を見る．シミュレーションでは，Sent Drop 方式の Source Retransmission を経験したファイルが 6612 個あったのに対して，RTT Aware 方式は 12576 個であった．Sent Drop 方式から RTT Aware 方式にすることによって，25000 ファイル中ファイルが新たに Source Retransmission するため，平均では $2000 \times \frac{5964}{25000} = 477[s]$ だけ長くなる．

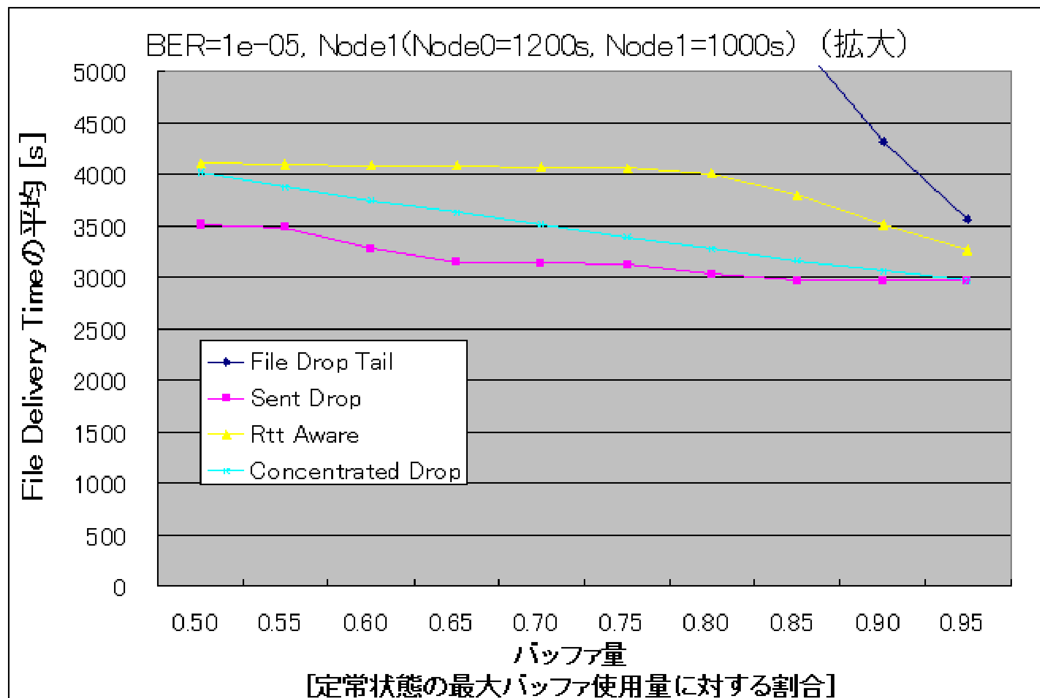

図 48: シミュレーション２の結果（拡大），（Node1, 1200s-1000s）

図 48 は，図 47 の一部を拡大したものである．Concentrated Drop 方式は，RTT Aware 方式よりも File Delivery Time が短い．例として Buffer=0.7 を見ると，RTT Aware 方式で Source Retransmission を経験したファイル数は 12,215 であったのに対して，Concentrated Drop 方式では 5,883 であった．したがって，RTT Aware 方式から Concentrated Drop 方式にすることによって，25,000 ファイル中 6,332 ファイルが Source Retransmission せずに済むため，平均では $2000 \times \frac{6332}{25000} = 507[s]$ だけ短くなる．

5　結論と将来課題

本研究では，惑星間インターネットにおいて効率の良いファイル転送方式について議論し，File Delivery Time を短縮する目標とした．CFDP ではバッファの溢れに関する規定が無く，通常用いられるパケット単位の Drop Tail ではデッドロック状態が発生することが明らかとなった．そこで本研究では CFDP を前提とし，再送制御にかかる時間を最小化するバッファ管理方式を提案した．

Sent Drop 方式は，再送回数を減少させる効果がある．これは，破棄すると必ず再送が必要となるパケット（Incomplete File パケット）を破棄しないためである．トポロジーによっては Source Retransmission の可能性が低くても Incomplete File パケットを破棄したほうが良い場合もあるが，本研究で想定したトポロジーでは常に Sent Drop 方式を用いたほうが File Delivery Time を短縮できた．

RTT Aware 方式は，一回の再送にかかる時間を短縮することができる．これは，パケットの再送にかかる時間の大部分が伝播遅延によるため，短い伝播遅延で済む再送を増やしたとしても，長い伝播遅延かがかる再送を少なくするほうがシステム全体として効率が上がるからである．Source までの RTT の差が少ない場合には，RTT Aware 方式を用いると偏りが生じ，フローごとに見ると File Delivery Time が延長することもある．しかし，システム全体では最悪でも Sent Drop 方式と同じ File Delivery Time で済む．

Concentrated Drop 方式は Source Retransmission を起こすファイルを集中させることによって，File Delivery Time を短縮することができる．再送回数や一回の再送にかかる時間は変化しないが，あるパケットの再送制御中に，他のパケットの再送制御を始めることができるため，時間が短縮される．ファイルごとの公平性はかなり損なわれるが，システム全体の File Delivry Time は改善する．

これらの方式を使用することによって，ビットエラー率が 10^{-5} 以下で定常状態時のバッファ使用量の半分程度しかバッファが使えないときであっても，溢れが起こらないときと概ね同じ File Delivery Time で送信できることがシミュレーションによって確認された．

今後の課題として，今回のシミュレーションでは扱わなかった条件下でどのような挙動をするのか調べたいと思う．この方式を用いたとき，溢れが起こらないときと概ね同じ File Delivery Time で送信するためには，どの程度のバッファ量を確保すればよいのかを明らかにしたい．また，惑星間インターネットを構築する上で，本研究では中継ノードのバッファ管理に焦点を当てたが，実際に惑星間インターネットが使われる場において，どのようなアプリケーショ

ンが必要とされ，そのためにどのような技術を用いればよいかを明らかにすることも今後重要になってくるのではないだろうか.

付録

NS 上の CFDP シミュレータのソースコード

cfdp.h

```
#ifndef ns_cfdp_h
#define ns_cfdp_h

#include "agent.h"
#include "tclcl.h"
#include "packet.h"
#include "address.h"
#include "ip.h"
#include "node.h"
#include <float.h>
#include <climits>

#define CFDP_UNKNOWN '0'

#define CFDP_META '1'
#define CFDP_DATA '2'
#define CFDP_EOF '3'
#define CFDP_NAK '4'
#define CFDP_FIN '5'
#define CFDP_SNAK '6'
#define CFDP_SRET '7'
#define CFDP_FFIN '8'
#define CFDP_ACK '9'
#define CFDP_SEOF 'j'

#define CFDP_PDU_SENDING 's'
#define CFDP_PDU_RECEIVED 'r'
#define CFDP_PDU_REQUESTING 'q'
#define CFDP_PDU_DROPPED 'd'
#define CFDP_PDU_COMPLETE 'c'
```

```
#define CFDP_PDU_SENT 't'

#define CFDP_FILE_INCOMPLETE 'I'
#define CFDP_FILE_COMPLETE 'C'
#define CFDP_FILE_SENT 'S'
#define CFDP_FILE_SENT_ALLDROPPED 'D'
#define CFDP_FILE_FIN 'F'
#define CFDP_FILE_INSRET 'R'

#define CFDP_PDU_END '-'
#define CFDP_ROUTE_END '-'

#define MAX_FILE_NUMBER 50010
#define MAX_PDU_NUMBER 103
#define MAX_NODE_NUMBER 4

#define CFDP_DATA_PDU_SIZE 1000
#define CFDP_CTRL_PDU_SIZE 0
#define CFDP_TIMEOUT 2.1

struct hdr_cfdp {

  char type;
  unsigned int file_seqno;
  short pdu_seqno;
  short eof_pdu_seqno;
  float delay;
};

struct cfdp_file_state {

  char state;
  char sender;
  short eof_pdu_seqno;
  float delay;
```

```
  float total_delay;
};

class CfdpAgent;

class CfdpSendTimer : public TimerHandler{
 public:
  CfdpSendTimer(): TimerHandler(){}
  void send_timer_const(CfdpAgent *a, char type,
                        unsigned int file_seqno,
                        short pdu_seqno, short eof_pdu_seqno,
                        char dest, Packet* pkt){
    a_ = a;
    type_ = type;
    file_seqno_ = file_seqno;
    pdu_seqno_ = pdu_seqno;
    eof_pdu_seqno_ = eof_pdu_seqno;
    dest_ = dest;
    pkt_ = pkt;
}

 protected:
  virtual void expire(Event *e);
  CfdpAgent *a_;
  char type_;
  unsigned int file_seqno_;
  short pdu_seqno_;
  short eof_pdu_seqno_;
  char dest_;
  Packet* pkt_;
} ;

class CfdpLogTimer : public TimerHandler{
 public:
  CfdpLogTimer(): TimerHandler(){}
```

```
  void log_timer_const(CfdpAgent *a){ a_ = a;}
 protected:
  virtual void expire(Event *e);
  CfdpAgent *a_;
} ;

class CfdpAgent : public Agent {
 public:
  CfdpAgent();
  int command(int argc, const char*const* argv);

  void recv(Packet*, Handler*);
  int cfdp_recv_meta(unsigned int file_seqno, short pdu_seqno,
                     short eof_pdu_seqno, char pre_node,
                     char next_node, float delay);
  int cfdp_recv_data(unsigned int file_seqno, short pdu_seqno,
                     short eof_pdu_seqno, char next_node);
  int cfdp_recv_eof(unsigned int file_seqno, short pdu_seqno,
                    short eof_pdu_seqno, char pre_node,
                    char next_node, float delay);
  void cfdp_recv_nak(unsigned int file_seqno, short pdu_seqno,
                     short eof_pdu_seqno, char next_node,
                     char pre_node);
  void cfdp_recv_snak(unsigned int file_seqno, short pdu_seqno,
                      short eof_pdu_seqno, char pre_node,
                      char next_node);
  void cfdp_recv_sret(unsigned int file_seqno, short pdu_seqno,
                      short eof_pdu_seqno, char next_node);
  void cfdp_recv_ffin(unsigned int file_seqno,
                      short eof_pdu_seqno, char pre_node);
  void cfdp_recv_fin(unsigned int file_seqno,
                     short eof_pdu_seqno, char pre_node);
  void cfdp_recv_ack(unsigned int file_seqno,
                     short pdu_seqno, char pre_node);
  void cfdp_recv_seof(unsigned int file_seqno, short pdu_seqno,
```

```
                     short eof_pdu_seqno, char pre_node,
                     char next_node);

 void cfdp_send_pdu(char type, unsigned int file_seqno,
                    short pdu_seqno, short eof_pdu_seqno,
                    char dest);
 void cfdp_send_file(unsigned int file_seqno, int file_size,
                     char next_node);

 void printTable(unsigned int file_seqno);
 void printState();
 void printLog(char node, char act, unsigned int file,
               short pdu, char type, char from, char to,
               double sent_plan_time);

 void cfdp_log_timeout();
 void cfdp_send_timeout(char type, unsigned int file_seqno,
                        short pdu_seqno, short eof_pdu_seqno,
                        char dest, Packet* pkt);

 int packet_drop(char type, unsigned int file_seqno,
                 short pdu_seqno);
 void drop_tail(char type, unsigned int file_seqno,
                short pdu_seqno);
 int drop2();
 int drop3();
 int drop4();

protected:
 CfdpLogTimer cfdp_log_timer;
 CfdpSendTimer cfdp_send_timer[MAX_FILE_NUMBER][MAX_PDU_NUMBER];

 int off_cfdp_;
 int output_log_;
 int drop_mode_;
```

```
  int cfdp_id_;
  double cfdp_storage_size_;
  double cfdp_bandwidth_;
  double cfdp_delay_;
  double cfdp_log_time_;

  double queue_length;
  double queue_length_reserved;
  double comp_queue_length;
  double incomp_queue_length;
  double sent_queue_length;
  int pre_drop_file;
  struct cfdp_file_state file_state[MAX_FILE_NUMBER];
};

unsigned int file_seqno_groval;

double start_time[MAX_FILE_NUMBER];
unsigned short drop_times[MAX_FILE_NUMBER];
unsigned short sret_times[MAX_FILE_NUMBER];
unsigned short sent_drop_times[MAX_FILE_NUMBER];

char route[MAX_FILE_NUMBER][MAX_NODE_NUMBER];
char nak_table[MAX_NODE_NUMBER][MAX_FILE_NUMBER][MAX_PDU_NUMBER];
#endif
```

cfdp.cc

```
#include "cfdp.h"

static class CfdpHeaderClass : public PacketHeaderClass {
public:
  CfdpHeaderClass() : PacketHeaderClass("PacketHeader/Cfdp",
sizeof(hdr_cfdp)) {}
} class_cfdphdr;

static class CfdpClass : public TclClass {
public:
  CfdpClass() : TclClass("Agent/Cfdp") {}
  TclObject* create(int, const char*const*) {
    return (new CfdpAgent());
  }
} class_cfdp;


CfdpAgent::CfdpAgent() : Agent(PT_CFDP)
{
  bind("packetSize_", &size_);
  bind("off_cfdp_", &off_cfdp_);
  bind("output_log_", &output_log_);
  bind("drop_mode_", &drop_mode_);
  bind("cfdp_id_", &cfdp_id_);
  bind("cfdp_storage_size_", &cfdp_storage_size_);
  bind("cfdp_bandwidth_", &cfdp_bandwidth_);
  bind("cfdp_delay_", &cfdp_delay_);
  bind("cfdp_log_time_", &cfdp_log_time_);

  queue_length = 0;
  queue_length_reserved = 0;
  comp_queue_length = 0;
  incomp_queue_length = 0;
```

```
  sent_queue_length = 0;
  pre_drop_file = -1;

  cfdp_log_timer.log_timer_const(this);
  cfdp_log_timer.resched(cfdp_log_time_);

  for(int i=0; i<MAX_FILE_NUMBER; i++){
    file_state[i].state = CFDP_UNKNOWN;
    for(int j=0; j<MAX_PDU_NUMBER; j++){
      nak_table[cfdp_id_][i][j] = CFDP_UNKNOWN;
    }
  }
}

void CfdpSendTimer::expire(Event *e)
{
  a_->cfdp_send_timeout(type_, file_seqno_, pdu_seqno_,
                        eof_pdu_seqno_, dest_, pkt_);
}

void CfdpAgent::cfdp_send_timeout(char type,
                                  unsigned int file_seqno,
                                  short pdu_seqno,
                                  short eof_pdu_seqno,
                                  char dest,
                                  Packet* pkt)
{
  if(type == CFDP_DATA){
    comp_queue_length -= CFDP_DATA_PDU_SIZE;
    sent_queue_length += CFDP_DATA_PDU_SIZE;
  }
  if(type == CFDP_EOF) file_state[file_seqno].state
                                      = CFDP_FILE_SENT;
  if(type == CFDP_DATA || type == CFDP_SRET)
    nak_table[cfdp_id_][file_seqno][pdu_seqno] = CFDP_PDU_SENT;
```

```
  if( (output_log_ & 1) == 1)
    printLog(cfdp_id_, 1, file_seqno, pdu_seqno, type,
             cfdp_id_, dest, 0);
  send(pkt, 0);
}


void CfdpLogTimer::expire(Event *e)
{
  a_->cfdp_log_timeout();
}

void CfdpAgent::cfdp_log_timeout()
{
  if((output_log_ & 2) >> 1 == 1){
    printf("log: --- node[%d] NAK Table at= %f -------------\n",
            cfdp_id_, Scheduler::instance().clock());
    for(unsigned int i=0; i<file_seqno_groval; i++){
      if(file_state[i].state != CFDP_UNKNOWN
         && file_state[i].state != CFDP_FILE_FIN) printTable(i);
    }
  }
  if((output_log_ & 4) >> 2 == 1)
    printf("q= %.0f incomp= %.0f comp= %.0f sent= %.0f
            node= %d at= %f \n",
           queue_length, incomp_queue_length, comp_queue_length,
           sent_queue_length, (int)cfdp_id_,
           Scheduler::instance().clock());
  cfdp_log_timer.resched(cfdp_log_time_);
}

void CfdpAgent::printTable(unsigned int file_seqno)
{
  printf("log: %f nak_table[id=%d][file=%d]= ",
          Scheduler::instance().clock(), cfdp_id_, file_seqno);
```

```
  for(int i=0; i<MAX_PDU_NUMBER; i++){
    printf("%c,", nak_table[cfdp_id_][file_seqno][i]);
  }
  printf("\n");
}

void CfdpAgent::printLog(char node, char act, unsigned int file,
                         short pdu, char type, char from, char to,
                         double sent_plan_time)
{
  char type_s[20];
  switch(type){
    case CFDP_META:
      strcpy( type_s, "META" );
      break;
    case CFDP_DATA:
      strcpy( type_s, "DATA" );
      break;
    case CFDP_EOF:
      strcpy( type_s, "EOF " );
      break;
    case CFDP_NAK:
      strcpy( type_s, "NAK " );
      break;
    case CFDP_FIN:
      strcpy( type_s, "FIN " );
      break;
    case CFDP_ACK:
      strcpy( type_s, "ACK " );
      break;
    case CFDP_FFIN:
      strcpy( type_s, "FFIN" );
      break;
    case CFDP_SNAK:
      strcpy( type_s, "SNAK" );
```

```
      break;
    case CFDP_SRET:
      strcpy( type_s, "SRET" );
      break;
    case CFDP_SEOF:
      strcpy( type_s, "SEOF" );
      break;
    default:
      printf("!ERROR!  PDU type(log) type=%d.\n", type);
  }
  if(act==1){
    printf("log: %05.5f, node%02d Send,(%u,%d),
            %s,(node%d->node%d), Q=%.0f, %.0f, %.0f, %.0f\n",
             Scheduler::instance().clock() + sent_plan_time,
             node, file, pdu, type_s, from, to, queue_length,
             incomp_queue_length, comp_queue_length,
             sent_queue_length);
  }else{
    printf("log: %05.5f, node%02d Recv,(%u,%d),
            %s,(node%d->node%d), Q=%.0f, %.0f, %.0f, %.0f\n",
             Scheduler::instance().clock() + sent_plan_time, node,
             file, pdu, type_s, from, to, queue_length,
             incomp_queue_length, comp_queue_length,
             sent_queue_length);
  }
}


int CfdpAgent::cfdp_recv_meta(unsigned int file_seqno,
                              short pdu_seqno,
                              short eof_pdu_seqno,
                              char pre_node,
                              char next_node, float delay)
{
  if(nak_table[cfdp_id_][file_seqno][0] == CFDP_PDU_RECEIVED
```

```
  || nak_table[cfdp_id_][file_seqno][0] == CFDP_PDU_SENT)
    return 1;

 if(next_node != CFDP_ROUTE_END){
  if(drop_mode_ == 0 | drop_mode_ == 1){
    if(cfdp_storage_size_ <
            (eof_pdu_seqno-1)*CFDP_DATA_PDU_SIZE
              + queue_length_reserved){
      file_state[file_seqno].delay = delay;  //for nak timeout
      nak_table[cfdp_id_][file_seqno][0] = CFDP_UNKNOWN;
      drop_times[file_seqno]++;
      if( (output_log_ & 1) == 1)
        printf("log: File[%d] is dropped because of full storage.
                storage=%f, reserved=%f \n", file_seqno,
                cfdp_storage_size_,
                (eof_pdu_seqno-1)*CFDP_DATA_PDU_SIZE
                 + queue_length_reserved);
      return 1;
    }else{
      queue_length_reserved
                += (eof_pdu_seqno-1)*CFDP_DATA_PDU_SIZE;
    }
  }
 }

  file_state[file_seqno].state = CFDP_UNKNOWN;
  file_state[file_seqno].sender = pre_node;
  file_state[file_seqno].eof_pdu_seqno = eof_pdu_seqno;
  file_state[file_seqno].total_delay += delay;
  file_state[file_seqno].delay = delay;

  nak_table[cfdp_id_][file_seqno][0] = CFDP_PDU_RECEIVED;
  return 0;
}
```

```
int CfdpAgent::cfdp_recv_data(unsigned int file_seqno,
                              short pdu_seqno,
                              short eof_pdu_seqno,
                              char next_node)
{
  if(file_state[file_seqno].state == CFDP_FILE_FIN) return 1;

  if(nak_table[cfdp_id_][file_seqno][pdu_seqno]
                                        == CFDP_PDU_RECEIVED
  || nak_table[cfdp_id_][file_seqno][pdu_seqno]
                                        == CFDP_PDU_COMPLETE
  || nak_table[cfdp_id_][file_seqno][pdu_seqno] == CFDP_PDU_SENT)
    return 1;

  if(nak_table[cfdp_id_][file_seqno][0] != CFDP_PDU_RECEIVED
  && nak_table[cfdp_id_][file_seqno][0] != CFDP_PDU_SENT)
    return 1; // File Drop Tail

  if(next_node != CFDP_ROUTE_END){
    queue_length += CFDP_DATA_PDU_SIZE;
    incomp_queue_length += CFDP_DATA_PDU_SIZE;
  }
  nak_table[cfdp_id_][file_seqno][pdu_seqno] = CFDP_PDU_RECEIVED;
  if(file_state[file_seqno].state == CFDP_UNKNOWN)
          file_state[file_seqno].state = CFDP_FILE_INCOMPLETE;
  return 0;
}

int CfdpAgent::cfdp_recv_eof(unsigned int file_seqno,
                             short pdu_seqno,
                             short eof_pdu_seqno,
                             char pre_node,
                             char next_node,
                             float delay)
{
```

```
file_state[file_seqno].delay = delay;

if(file_state[file_seqno].state == CFDP_FILE_FIN
|| file_state[file_seqno].state == CFDP_FILE_SENT
|| file_state[file_seqno].state == CFDP_FILE_COMPLETE)
  return 1;

nak_table[cfdp_id_][file_seqno][pdu_seqno] = CFDP_PDU_RECEIVED;

//File Drop Tail
if(drop_mode_ == 0 || drop_mode_ == 1){
  if(nak_table[cfdp_id_][file_seqno][0] == CFDP_UNKNOWN){
    nak_table[cfdp_id_][file_seqno][eof_pdu_seqno]
                                          = CFDP_UNKNOWN;
    for(int w=0; w<=eof_pdu_seqno; w++){
      nak_table[cfdp_id_][file_seqno][w] = CFDP_PDU_REQUESTING;
      cfdp_send_pdu(CFDP_NAK, file_seqno, w, eof_pdu_seqno,
                    pre_node);
      }
    return 0;
  }
}

// Error occured at DATA
int flag=0;
for(int i=0; i <= eof_pdu_seqno; i++){
  if(nak_table[cfdp_id_][file_seqno][i] == CFDP_PDU_SENDING){
    cfdp_send_pdu(CFDP_NAK, file_seqno, i, eof_pdu_seqno,
                  pre_node);
    nak_table[cfdp_id_][file_seqno][i] = CFDP_PDU_REQUESTING;
    flag=1;
  }else if(nak_table[cfdp_id_][file_seqno][i]
                                    == CFDP_PDU_REQUESTING){
    flag=1;
  }
```

```
  }
  if(flag == 1) return 0;

  // Complete!
  file_state[file_seqno].state = CFDP_FILE_COMPLETE;
  cfdp_send_pdu(CFDP_FIN, file_seqno, -1, eof_pdu_seqno, pre_node);

  if(next_node == CFDP_ROUTE_END){
    printf("%f %f %d %d %d %d %d \n",
           Scheduler::instance().clock() - start_time[file_seqno],
           Scheduler::instance().clock(),
           file_seqno, route[file_seqno][0],
           drop_times[file_seqno], sent_drop_times[file_seqno],
                                         sret_times[file_seqno]);
    file_state[file_seqno].state = CFDP_FILE_FIN;
    cfdp_send_pdu(CFDP_FFIN, file_seqno, -2, eof_pdu_seqno,
                  pre_node);

  }else{
    if(drop_mode_ == 0){
      int file_size = (eof_pdu_seqno-1)*CFDP_DATA_PDU_SIZE;
      cfdp_send_file(file_seqno, file_size, next_node);
    }else{
      cfdp_send_pdu(CFDP_EOF, file_seqno, eof_pdu_seqno,
                    eof_pdu_seqno, next_node);
    }
  }
  return 0;
}

void CfdpAgent::cfdp_recv_nak(unsigned int file_seqno,
                              short pdu_seqno,
                              short eof_pdu_seqno,
                              char next_node, char pre_node)
{
```

```
if(cfdp_id_ != pre_node
   && nak_table[cfdp_id_][file_seqno][pdu_seqno]
                                   == CFDP_PDU_DROPPED){
// SRET is needed
  if( (output_log_ & 1) == 1)
    printf("log: This PDU(%03d, %04d) is Dropped from node%d.
            So, let's try SRET.\n",
            file_seqno, pdu_seqno, cfdp_id_);
  nak_table[cfdp_id_][file_seqno][pdu_seqno]
                                      = CFDP_PDU_REQUESTING;
  file_state[file_seqno].state = CFDP_FILE_INSRET;
  cfdp_send_pdu(CFDP_SNAK, file_seqno, pdu_seqno,
                                     eof_pdu_seqno, pre_node);

}else{
// SRET is not needed
nak_table[(int)next_node][file_seqno][pdu_seqno]
                                           = CFDP_PDU_SENDING;
if(pdu_seqno == 0){
  cfdp_send_pdu(CFDP_META, file_seqno, pdu_seqno,
                                    eof_pdu_seqno, next_node);
}else if(pdu_seqno == eof_pdu_seqno){
  // do nothing, to Other PDU is in NAK?
}else{
  sent_queue_length -= CFDP_DATA_PDU_SIZE;
  incomp_queue_length += CFDP_DATA_PDU_SIZE;
  cfdp_send_pdu(CFDP_DATA, file_seqno, pdu_seqno,
                                    eof_pdu_seqno, next_node);
}
  //Other PDU is in NAK?
  for(int i=0; i<eof_pdu_seqno; i++)
    if(nak_table[(int)next_node][file_seqno][i]
                                      == CFDP_PDU_REQUESTING)
      return;
  cfdp_send_pdu(CFDP_EOF, file_seqno, eof_pdu_seqno,
```

```
                                      eof_pdu_seqno, next_node);
  }
}

void CfdpAgent::cfdp_recv_snak(unsigned int file_seqno,
                               short pdu_seqno,
                               short eof_pdu_seqno,
                               char pre_node, char next_node)
{
  if(pre_node == cfdp_id_){ // this node is Src
    cfdp_send_pdu(CFDP_SRET, file_seqno, pdu_seqno,
                                      eof_pdu_seqno, next_node);
    cfdp_send_pdu(CFDP_SEOF, file_seqno, pdu_seqno,
                                      eof_pdu_seqno, next_node);
    sret_times[file_seqno]++;

  }else{
    cfdp_send_pdu(CFDP_SNAK, file_seqno, pdu_seqno,
                                       eof_pdu_seqno, pre_node);
  }
}

void CfdpAgent::cfdp_recv_sret(unsigned int file_seqno,
                               short pdu_seqno,
                               short eof_pdu_seqno,
                               char next_node)
{
  if(file_state[file_seqno].state == CFDP_FILE_SENT ){
    // this node don't request SRET
    cfdp_send_pdu(CFDP_SRET, file_seqno, pdu_seqno,
                                      eof_pdu_seqno, next_node);
    nak_table[cfdp_id_][file_seqno][pdu_seqno] = CFDP_PDU_DROPPED;

  }else if(file_state[file_seqno].state == CFDP_FILE_INSRET ){
    //this node requested SRET
```

```
    queue_length += CFDP_DATA_PDU_SIZE;
    incomp_queue_length += CFDP_DATA_PDU_SIZE;
    nak_table[cfdp_id_][file_seqno][pdu_seqno]
                                        = CFDP_PDU_RECEIVED;
    nak_table[(int)next_node][file_seqno][pdu_seqno]
                                         = CFDP_PDU_SENDING;
    cfdp_send_pdu(CFDP_DATA, file_seqno, pdu_seqno,
                  eof_pdu_seqno, next_node);

    //if other PDU is not in SRET, so send EOF
    for(int i=0; i<eof_pdu_seqno; i++){
      if(nak_table[(int)next_node][file_seqno][i]
                                    == CFDP_PDU_REQUESTING){
        return;
      }
    }
    cfdp_send_pdu(CFDP_EOF, file_seqno, eof_pdu_seqno,
                                  eof_pdu_seqno, next_node);

  }else{
    printf("!ERROR! wrong SRET node=%d, file=%d, pdu=%d, ",
           (int)cfdp_id_, file_seqno, pdu_seqno);
    exit(-1);
  }
}

void CfdpAgent::cfdp_recv_seof(unsigned int file_seqno,
                               short pdu_seqno,
                               short eof_pdu_seqno,
                               char pre_node, char next_node)
{

  if(file_state[file_seqno].state == CFDP_FILE_FIN ){
    // this node don't request SRET
    cfdp_send_pdu(CFDP_SEOF, file_seqno, pdu_seqno,
```

```
                                    eof_pdu_seqno, next_node);
    nak_table[cfdp_id_][file_seqno][pdu_seqno] = CFDP_PDU_DROPPED;

  }else if(file_state[file_seqno].state == CFDP_FILE_INSRET ){
    // this node requested SRET
    if(nak_table[cfdp_id_][file_seqno][pdu_seqno]
                                        == CFDP_PDU_SENDING){
      // ERROR at SRET
      cfdp_send_pdu(CFDP_SNAK, file_seqno, pdu_seqno,
                                    eof_pdu_seqno, pre_node);
      nak_table[(int)next_node][file_seqno][pdu_seqno]
                                       = CFDP_PDU_REQUESTING;

    }else if(nak_table[cfdp_id_][file_seqno][pdu_seqno]
                                        == CFDP_PDU_RECEIVED
          || nak_table[cfdp_id_][file_seqno][pdu_seqno]
                                        == CFDP_PDU_SENT
          || nak_table[cfdp_id_][file_seqno][pdu_seqno]
                                        == CFDP_PDU_COMPLETE
          || nak_table[cfdp_id_][file_seqno][pdu_seqno]
                                        == CFDP_PDU_DROPPED){
      // do nothing
    }else{
      printf("!ERROR! nak_table(%d,%d)=%c at=%d\n",
              file_seqno, pdu_seqno,
              nak_table[cfdp_id_][file_seqno][pdu_seqno],
              (int)cfdp_id_);
      exit(-1);
    }
  }
}


void CfdpAgent::cfdp_recv_ack(unsigned int file_seqno,
                              short pdu_seqno, char pre_node)
```

```
{
  if(pre_node != cfdp_id_){
    if(nak_table[cfdp_id_][file_seqno][pdu_seqno]
                                                  == CFDP_PDU_SENT){

      nak_table[cfdp_id_][file_seqno][pdu_seqno]
                                                = CFDP_PDU_DROPPED;
      queue_length -= CFDP_DATA_PDU_SIZE;
      sent_queue_length -= CFDP_DATA_PDU_SIZE;
      queue_length_reserved -= CFDP_DATA_PDU_SIZE;
    }
  }
}

void CfdpAgent::cfdp_recv_fin(unsigned int file_seqno,
                              short eof_pdu_seqno, char pre_node)
{
  file_state[file_seqno].state = CFDP_FILE_FIN;

  if(pre_node != cfdp_id_){
    for(int i=1; i<eof_pdu_seqno; i++){
      if(nak_table[cfdp_id_][file_seqno][i] == CFDP_PDU_RECEIVED
      || nak_table[cfdp_id_][file_seqno][i] == CFDP_PDU_SENT){

          nak_table[cfdp_id_][file_seqno][i] = CFDP_PDU_DROPPED;
          queue_length -= CFDP_DATA_PDU_SIZE;
          sent_queue_length -= CFDP_DATA_PDU_SIZE;
          queue_length_reserved -= CFDP_DATA_PDU_SIZE;
      }
    }
  }else{
    // if this node is Src, delete by ffin
  }
}
```

```
void CfdpAgent::cfdp_recv_ffin(unsigned int file_seqno,
                               short eof_pdu_seqno,
                               char pre_node)
{
  if(pre_node == cfdp_id_){ // this node is Src
      queue_length -= CFDP_DATA_PDU_SIZE*(eof_pdu_seqno-1);
      sent_queue_length -= CFDP_DATA_PDU_SIZE*(eof_pdu_seqno-1);

  }else{
    cfdp_send_pdu(CFDP_FFIN, file_seqno, -2, eof_pdu_seqno,
                  pre_node);
  }
}

void CfdpAgent::drop_tail(char type, unsigned int file_seqno,
                          short pdu_seqno)
{
  if( (output_log_ & 1) == 1)
    printf("log: Overflow occured(Drop-tail), file=%d, pdu=%d\n",
            file_seqno, pdu_seqno);

  drop_times[file_seqno]++;
  nak_table[cfdp_id_][file_seqno][pdu_seqno] = CFDP_PDU_REQUESTING;
  cfdp_send_pdu(CFDP_NAK, file_seqno, pdu_seqno,
                file_state[file_seqno].eof_pdu_seqno,
                file_state[file_seqno].sender);
  if(type == CFDP_DATA || type == CFDP_SRET){
    queue_length -= CFDP_DATA_PDU_SIZE;
    incomp_queue_length -= CFDP_DATA_PDU_SIZE;
  }else{
    // do nothing
  }
}
int CfdpAgent::drop2(){
```

```
char flag=0;
unsigned int i;
int k;

for(i=0; i<file_seqno_groval; i++){
  if(file_state[i].state != CFDP_FILE_FIN
     || file_state[i].state != CFDP_UNKNOWN){
    for(k=1; k<file_state[i].eof_pdu_seqno; k++){
      if(nak_table[cfdp_id_][i][k] == CFDP_PDU_SENT){
        flag = 1;
        break;
      }
    }
  }
if(flag == 1) break;
}

if(flag == 0){
  printf("!ERROR! cannot drop sent PDU at:%f.\n",
          Scheduler::instance().clock());
  exit(-1);
}

int min_file = i;
int min_pdu = k;

if( (output_log_ & 1) == 1){
  printf("log: Overflow occured,(%d,%d),
           nak_table[%d][][]=%c\n",
           min_file, min_pdu, cfdp_id_,
           nak_table[cfdp_id_][min_file][min_pdu]);
}
sent_drop_times[min_file]++;
nak_table[cfdp_id_][min_file][min_pdu] = CFDP_PDU_DROPPED;
```

```
  queue_length -= CFDP_DATA_PDU_SIZE;
  sent_queue_length -= CFDP_DATA_PDU_SIZE;
  return 0;
}

int CfdpAgent::drop3(){

//  unsigned int i= rand()%file_seqno_groval;
  unsigned int i;
  int k;
  unsigned int min_file=0;
  unsigned int min_pdu=0;
  double cost_min=DBL_MAX;
  char flag=0;

  //drop sent pdu(Sent File or Incomplete File)
  for(i=0; i<file_seqno_groval; i++){
    if(file_state[i].state != CFDP_FILE_FIN
       && file_state[i].state != CFDP_UNKNOWN
    && file_state[i].total_delay < cost_min){
      for(k=1; k<file_state[i].eof_pdu_seqno; k++){
        if(nak_table[cfdp_id_][i][k] == CFDP_PDU_SENT ){
          cost_min = file_state[i].total_delay;
          min_file=i;
          min_pdu=k;
          flag=1;
        }
      }
    }
  }

  if(flag == 0){
    printf("!ERROR! cannot drop sent PDU at:%f.\n",
            Scheduler::instance().clock());
    exit(-1);
```

```
  }
  if( (output_log_ & 1) == 1){
    printf("log: Overflow occured,(%d,%d),
            nak_table[%d][][]=%c\n",
            min_file, min_pdu, cfdp_id_,
            nak_table[cfdp_id_][min_file][min_pdu]);
  }
  sent_drop_times[min_file]++;
  nak_table[cfdp_id_][min_file][min_pdu] = CFDP_PDU_DROPPED;

  queue_length -= CFDP_DATA_PDU_SIZE;
  sent_queue_length -= CFDP_DATA_PDU_SIZE;
  return 0;
}

int CfdpAgent::drop4(){

  unsigned int i;
  int k;
  unsigned int min_file=0;
  unsigned int min_pdu=0;
  double cost_min=DBL_MAX;
  char flag=0;

  //drop sent pdu(Sent File or Incomplete File)
  if(pre_drop_file == -1) pre_drop_file = file_seqno_groval;
  i=pre_drop_file;

  for(i=file_seqno_groval; i>0; i--){
    if(file_state[i].state != CFDP_FILE_FIN
       && file_state[i].state != CFDP_UNKNOWN
    && file_state[i].total_delay < cost_min){
      for(k=1; k<file_state[i].eof_pdu_seqno; k++){
        if(nak_table[cfdp_id_][i][k] == CFDP_PDU_SENT ){
          cost_min = file_state[i].total_delay;
```

```
          min_file=i;
          min_pdu=k;
          flag=1;
          break;
        }
      }
    }
  // not break
  }
  if((int)min_pdu == file_state[min_file].eof_pdu_seqno-1){
    pre_drop_file = -1;
  }else{
    pre_drop_file = i;
  }

  if(flag == 0){
    printf("!ERROR! cannot drop sent PDU at:%f.\n",
            Scheduler::instance().clock());
    exit(-1);
  }
  if( (output_log_ & 1) == 1){
    printf("log: Overflow occured,(%d,%d),
            nak_table[%d][][]=%c\n",
            min_file, min_pdu, cfdp_id_,
            nak_table[cfdp_id_][min_file][min_pdu]);
  }
  sent_drop_times[min_file]++;
  nak_table[cfdp_id_][min_file][min_pdu] = CFDP_PDU_DROPPED;

  queue_length -= CFDP_DATA_PDU_SIZE;
  sent_queue_length -= CFDP_DATA_PDU_SIZE;
  return 0;
}

int CfdpAgent::packet_drop(char type, unsigned int file_seqno,
```

```
                               short pdu_seqno)
{
  switch(drop_mode_){
    case 4:
      return drop4();
    case 3:
      return drop3();
    case 2:
      return drop2();
    case 1:
      drop_tail(type, file_seqno, pdu_seqno);
      return 1;
    case 0:
      return 0;
    default:
      printf("!ERROR! Drop Mode.\n");
      exit(-1);
  }
}

void CfdpAgent::recv(Packet* pkt, Handler*)
{

  hdr_cfdp* hdr = (hdr_cfdp*)pkt->access(off_cfdp_);
  hdr_ip* iph = hdr_ip::access(pkt);

  int drop = 0;
  int is_tail_drop = 0;
  char type = hdr->type;
  char counterpart = (char)iph->saddr();
  unsigned int file_seqno = hdr->file_seqno;
  short pdu_seqno = hdr->pdu_seqno;
  short eof_pdu_seqno = hdr->eof_pdu_seqno;
  float delay = hdr->delay;
```

```
if(file_seqno == 0) printTable(file_seqno);

  if(file_state[file_seqno].state == CFDP_FILE_FIN){
    if(type == CFDP_META || type == CFDP_DATA || type == CFDP_EOF
       || type == CFDP_NAK || type == CFDP_FIN){
      return;
    }
  }

  int i=0;
  while(true){
    if(route[file_seqno][i] == cfdp_id_) break;
    i++;
  }
  char next_node = route[file_seqno][i+1];
  char pre_node;
  if(i==0){
    pre_node = cfdp_id_;   // at Src, pre_node = this_node
  }else{
    pre_node = route[file_seqno][i-1];
  }
  switch(hdr->type){
    case CFDP_META:
      drop = cfdp_recv_meta(file_seqno, pdu_seqno, eof_pdu_seqno,
                            pre_node, next_node, delay);
      break;
    case CFDP_DATA:
      drop = cfdp_recv_data(file_seqno, pdu_seqno, eof_pdu_seqno,
                            next_node);
      break;
    case CFDP_EOF:
      drop = cfdp_recv_eof(file_seqno, pdu_seqno, eof_pdu_seqno,
                           pre_node, next_node, delay);
      break;
    case CFDP_NAK:
```

```
    cfdp_recv_nak(file_seqno, pdu_seqno, eof_pdu_seqno,
                  next_node, pre_node);
    break;
  case CFDP_ACK:
    cfdp_recv_ack(file_seqno, pdu_seqno, pre_node);
    break;
  case CFDP_FIN:
    cfdp_recv_fin(file_seqno, eof_pdu_seqno, pre_node);
    break;
  case CFDP_SNAK:
    cfdp_recv_snak(file_seqno, pdu_seqno, eof_pdu_seqno,
                   pre_node, next_node);
    break;
  case CFDP_SEOF:
    cfdp_recv_seof(file_seqno, pdu_seqno, eof_pdu_seqno,
                   pre_node, next_node);
    break;
  case CFDP_SRET:
    cfdp_recv_sret(file_seqno, pdu_seqno, eof_pdu_seqno,
                   next_node);
    break;
  case CFDP_FFIN:
    cfdp_recv_ffin(file_seqno, eof_pdu_seqno, pre_node);
    break;
  default:
    printf("!ERROR! PDU type(recv). type=%d\n", hdr->type);
    exit(-1);
}

if( (output_log_ & 1) == 1 && drop != 1)
  printLog(cfdp_id_, 0, file_seqno, pdu_seqno, type,
           counterpart, cfdp_id_, 0);

if( queue_length > cfdp_storage_size_ ){
  if(type == CFDP_DATA || type == CFDP_SRET){
```

```
      is_tail_drop = packet_drop(type, file_seqno, pdu_seqno);
    }
  }
  if(drop_mode_ != 0){
    if(type == CFDP_DATA && is_tail_drop == 0 && drop != 1){
      cfdp_send_pdu(CFDP_ACK, file_seqno, pdu_seqno,
                    eof_pdu_seqno, pre_node);
      if(next_node != CFDP_ROUTE_END){
        cfdp_send_pdu(CFDP_DATA, file_seqno, pdu_seqno,
                      eof_pdu_seqno, next_node);
      }
    }else if(type == CFDP_META && next_node != CFDP_ROUTE_END){
      cfdp_send_pdu(CFDP_META, file_seqno, 0,
                    eof_pdu_seqno, next_node);
    }
  }

  if(cfdp_id_ == 2)
     printf("--- (%d,%d) recv storage_used=%d\n",
            file_seqno, pdu_seqno,
            file_state[file_seqno].storage_used);

  if(queue_length < 0 || incomp_queue_length < 0 ||
     comp_queue_length < 0 || sent_queue_length < 0){
    printf("!ERROR! queue=%.0f, incomp=%.0f, comp=%.0f,
            sent=%.0f\n",
            queue_length, incomp_queue_length, comp_queue_length,
             sent_queue_length);
    exit(-1);
  }
  Packet::free(pkt);
}

void CfdpAgent::cfdp_send_pdu(char type, unsigned int file_seqno,
                              short pdu_seqno,
```

```
                              short eof_pdu_seqno, char dest)
{
  // these need to be before allocpkt()
  if(type == CFDP_DATA || type == CFDP_SRET){
    size_ = CFDP_DATA_PDU_SIZE;
  }else{
    size_ = CFDP_CTRL_PDU_SIZE;
  }

  Packet* pkt = allocpkt();
  hdr_cfdp* hdr = (hdr_cfdp*)pkt->access(off_cfdp_);
  hdr_ip* iph = hdr_ip::access(pkt);

  hdr->type = type;
  hdr->file_seqno = file_seqno;
  hdr->pdu_seqno = pdu_seqno;
  hdr->eof_pdu_seqno = eof_pdu_seqno;
  hdr->delay = cfdp_delay_;
  iph->daddr() = (int)dest;

  double sent_plan_time = 0;

  if(type == CFDP_ACK || type == CFDP_FIN || type == CFDP_NAK
     || type == CFDP_SNAK || type == CFDP_FFIN){
    // return PDU depart immediately
    if( (output_log_ & 1) == 1)
      printLog(cfdp_id_, 1, file_seqno, pdu_seqno, type,
               cfdp_id_, dest, 0);
    send(pkt, 0);

  }else if(type == CFDP_SRET){
      if( (output_log_ & 1) == 1)
        printLog(cfdp_id_, 1, file_seqno, pdu_seqno, type,
                 cfdp_id_, dest, 0);
      send(pkt, 0);
```

```
  }else{ // wait in complete queue

    sent_plan_time = (double)comp_queue_length*8/cfdp_bandwidth_
                 + (double)CFDP_DATA_PDU_SIZE*80/cfdp_bandwidth_;
    if(type == CFDP_DATA){
      incomp_queue_length -= CFDP_DATA_PDU_SIZE;
      comp_queue_length += CFDP_DATA_PDU_SIZE;
    }

    if(type == CFDP_DATA || type == CFDP_SRET)
        nak_table[cfdp_id_][file_seqno][pdu_seqno]
                                         = CFDP_PDU_COMPLETE;
        nak_table[(int)dest][file_seqno][pdu_seqno]
                                         = CFDP_PDU_SENDING;
    if(type == CFDP_EOF)
        nak_table[(int)dest][file_seqno][pdu_seqno+1]
                                         = CFDP_PDU_END;

    if(type == CFDP_META)
        sent_plan_time
               -= (double)CFDP_DATA_PDU_SIZE*80/cfdp_bandwidth_;
    if(type == CFDP_EOF || type == CFDP_SEOF)
        sent_plan_time
               += (double)CFDP_DATA_PDU_SIZE*64/cfdp_bandwidth_;
    cfdp_send_timer[file_seqno][pdu_seqno].send_timer_const(
        this, type, file_seqno, pdu_seqno,
        eof_pdu_seqno, dest, pkt);
    cfdp_send_timer[file_seqno][pdu_seqno].resched(sent_plan_time);
  }
}

void CfdpAgent::cfdp_send_file(unsigned int file_seqno,
                               int file_size, char next_node)
{
```

```
  short eof_pdu_seqno
     = (int)ceil((double)file_size/CFDP_DATA_PDU_SIZE)+1;

  short pdu_seqno = 0;
  cfdp_send_pdu(CFDP_META, file_seqno, pdu_seqno, eof_pdu_seqno,
                next_node);
  for(pdu_seqno=1; pdu_seqno<eof_pdu_seqno; pdu_seqno++){
    cfdp_send_pdu(CFDP_DATA, file_seqno, pdu_seqno, eof_pdu_seqno,
                  next_node);
  }
  cfdp_send_pdu(CFDP_EOF, file_seqno, pdu_seqno, eof_pdu_seqno,
                next_node);
}

int CfdpAgent::command(int argc, const char*const* argv)
{
 if (argc > 3) {
    if (strcmp(argv[1], "send") == 0) {

      int file_size = atoi(argv[2]);
      if(file_size + queue_length > cfdp_storage_size_){
        if( (output_log_ & 1) == 1)
          printf("log: can't be accepted by full queue.
                  at= %f\n",  Scheduler::instance().clock());
        return (TCL_OK);
      }

      unsigned int file_seqno = file_seqno_groval++;

      drop_times[file_seqno] = 0;
      sret_times[file_seqno] = 0;
      sent_drop_times[file_seqno] = 0;
      start_time[file_seqno] = Scheduler::instance().clock();

      // route[][]
```

```
      route[file_seqno][0] = cfdp_id_;
      int i = 3;
      while( i < argc ){
         route[file_seqno][i-2] = atoi(argv[i]);
        i++;
      }
      route[file_seqno][i-2] = CFDP_ROUTE_END;

      // For SNAK
      queue_length
         += ((int)ceil((double)file_size/CFDP_DATA_PDU_SIZE))
                                                *CFDP_DATA_PDU_SIZE;
      incomp_queue_length
         += ((int)ceil((double)file_size/CFDP_DATA_PDU_SIZE))
                                                 *CFDP_DATA_PDU_SIZE
      cfdp_send_file(file_seqno, file_size, route[file_seqno][1]);

      return (TCL_OK);
    }
  }
  return (Agent::command(argc, argv));
}
```

シミュレーションに用いたシナリオファイル

scenario.tcl

```
set drop_mode 1
set log_node2 4

set storage2 13750000
set ber 0
set interval_average 0.5
set finish_time 200000

set rand_seed [lindex $argv 0]
set filesize 10000
set num_of_files0 25000
set num_of_files1 25000

set delay0_2 1200
set delay1_2 1
set delay2_3 300

set bandwidth0_2 [expr 800*pow(10, 3)]
set bandwidth1_2 [expr 800*pow(10, 3)]
set bandwidth2_3 [expr 800*pow(10, 3)]
set storage0 [expr 6*pow(10, 11)]
set storage1 [expr 6*pow(10, 11)]
set storage3 [expr 10000]
set log_node0 0
set log_node1 0
set log_node3 0

set per [expr 1-pow(1-$ber,8*1000)]
set byer [expr 1-pow(1-$ber,8)]
# puts "ber=$ber"
# puts "per=$per"
```

```
#Create a simulator object
set ns [new Simulator]
$ns use-scheduler Heap

#set f [open out.tr w]
#$ns trace-all $f

#Define a 'finish' procedure
proc finish {} {
#  global ns f
#  $ns flush-trace
#  close $f
  exit 0
}

set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]

$ns duplex-link $n0 $n2 $bandwidth0_2 $delay0_2 DropTail
$ns queue-limit $n0 $n2 [expr $storage0*1.1]
$ns duplex-link $n1 $n2 $bandwidth1_2 $delay1_2 DropTail
$ns queue-limit $n1 $n2 [expr $storage1*1.1]
$ns duplex-link $n2 $n3 $bandwidth2_3 $delay2_3 DropTail
$ns queue-limit $n2 $n3 [expr $storage2*1.1]

global defaultRNG
set rng [new RNG]
for {set i 1} {$i < $rand_seed} {incr i} {
  $rng next-substream
}
```

```
set loss_random_variable [new RandomVariable/Uniform]
$loss_random_variable use-rng $rng
set loss_module [new ErrorModel]
$loss_module unit byte
$loss_module set rate_ $byer
$loss_module ranvar $loss_random_variable
$loss_module drop-target [new Agent/Null]
$ns link-lossmodel $loss_module $n0 $n2

$rng next-substream

set loss_random_variable2 [new RandomVariable/Uniform]
$loss_random_variable2 use-rng $rng
set loss_module2 [new ErrorModel]
$loss_module2 unit byte
$loss_module2 set rate_ $byer
$loss_module2 ranvar $loss_random_variable
$loss_module2 drop-target [new Agent/Null]
$ns link-lossmodel $loss_module2 $n1 $n2

$rng next-substream
set loss_random_variable1 [new RandomVariable/Uniform]
$loss_random_variable1 use-rng $rng
set loss_module1 [new ErrorModel]
$loss_module1 unit byte
$loss_module1 set rate_ $byer
$loss_module1 ranvar $loss_random_variable
$loss_module1 drop-target [new Agent/Null]
$ns link-lossmodel $loss_module1 $n2 $n3

set p0 [new Agent/Cfdp]
$ns attach-agent $n0 $p0
$p0 set cfdp_id_ 0
$p0 set cfdp_storage_size_ $storage0
$p0 set cfdp_error_rate_ $per
```

```
$p0 set cfdp_delay_ $delay0_2
$p0 set cfdp_bandwidth_ $bandwidth0_2
$p0 set output_log_ $log_node0
$p0 set drop_mode_ $drop_mode

set p1 [new Agent/Cfdp]
$ns attach-agent $n1 $p1
$p1 set cfdp_id_ 1
$p1 set cfdp_storage_size_ $storage1
$p1 set cfdp_error_rate_ $per
$p1 set cfdp_delay_ $delay1_2
$p1 set cfdp_bandwidth_ $bandwidth1_2
$p1 set output_log_ $log_node1
$p1 set drop_mode_ $drop_mode

set p2 [new Agent/Cfdp]
$ns attach-agent $n2 $p2
$p2 set cfdp_id_ 2
$p2 set cfdp_storage_size_ $storage2
$p2 set cfdp_error_rate_ $per
$p2 set cfdp_delay_ $delay2_3
$p2 set cfdp_bandwidth_ $bandwidth2_3
$p2 set output_log_ $log_node2
$p2 set drop_mode_ $drop_mode

set p3 [new Agent/Cfdp]
$ns attach-agent $n3 $p3
$p3 set cfdp_id_ 3
$p3 set cfdp_storage_size_ $storage3
$p3 set cfdp_error_rate_ 0 ; # this node is Dst
$p3 set cfdp_delay_ 0 ; # this node is Dst
$p3 set cfdp_bandwidth_ 0 ; # this node is Dst
$p3 set output_log_ $log_node3
$p3 set drop_mode_ $drop_mode
```

```
$ns connect $p0 $p2
$ns connect $p1 $p2
$ns connect $p2 $p3

#Schedule events

$rng next-substream
set arrival_time $delay0_2
for {set i 1} {$i < [expr $num_of_files1 + 1]} {incr i} {
  set interval [$rng exponential $interval_average]
  set arrival_time [expr $arrival_time+$interval]
  $ns at $arrival_time "$p1 send $filesize 2 3"
}

$rng next-substream
set arrival_time 0.0
for {set i 1} {$i < [expr $num_of_files0 + 1]} {incr i} {
  set interval [$rng exponential $interval_average]
  set arrival_time [expr $arrival_time+$interval]
  $ns at $arrival_time "$p0 send $filesize 2 3"
}

$ns at $finish_time "finish"
$ns run
```

謝辞

本研究にあたり，多くの方々に助言をいただきました．本研究の機会を与えてくださり，数多くのご指導をいただいた，若原恭教授，知識不足の私の質問に対しても懇切丁寧に答えてくださった，中村文隆助教に心から御礼申し上げます．また、中山雅哉准教授、関谷勇司助教を始め、研究室の皆様にも、ミーティングや研究室等で様々なアドバイスをいただきました。大変感謝しております。そして誰よりも，父と母に最大の感謝を．

参考文献

[1] “Mars Science Laboratory”, http://mars.jpl.nasa.gov/msl/

[2] “月周回衛星計画「セレーネ」”, http://www.jaxa.jp/missions/projects/sat/exploration/selene/index_j.html

[3] Sagan, C., “The Planet Venus”, Science 133, pp.849-858, 1961.

[4] “InterPlaNetary Internet Project”, http://www.ipnsig.org/home.htm

[5] Akyildiz, I. F., Akan, O. B., Chen, C., Fang, J., Su, W., “InterPlaNetary Internet: State-of-the-Art and Research Challenges“ ,Computer Networks, Journal (Elsevier), vol. 43, pp.75-112, Oct 2003.

[6] Cerf, V., Burleigh, S., Hooke, A., Burst, R., Scott, K., Travis, E., and Weiss, H., “Delay Tolerant Network Architecture”, draft-irtf-dtnrg-arch-05.txt, Sep 2006.

[7] Burleigh, S., Hooke, A., Torgerson, L., Fall, K., Cerf, V., Durst, B., and Scott, K., “Delay-Tolerant Networking: An Approach to Interplanetary Internet”, IEEE Communications Magazine, vol.41, pp.128-136, Jun 2003.

[8] Consultative Committee for Space Data Systems, “Proximity-1 Space Link Protocol”, Recommendation for Space Data System Standards, CCSDS 211.0-B-1, Blue Book, Oct 2002.

[9] Chiaraluce, F., Gambi, E., Garello, R, Pierleoni, P., Calzolari, G.P., Vassallo, R., “On the new CCSDS standard for space telemetry: turbo codes and symbol synchronization”, Proceedings of the IEEE ICC 2000, vol.1, 2000, pp.451-454.

[10] 鶴　正人, 熊副和美, 尾家祐二, “長距離高速通信のための TCP 性能改善技術の動向”, IPSJ Magazine Vol.44 No.9 Sep. 2003

[11] Akan, O. B., Fang, J., Akyildiz, I. F., “TP-Planet: A Relible Transport Protocol for InterPlaNetary Internet”, IEEE Journal of Selected Areas in Communications (JSAC), Vol.22, Issue 2, pp.348-361, Feb 2004.

[12] “The Consultative Committee for Space Data Systems”, http://public.ccsds.org/

[13] Consultative Committee for Space Data Systems, “CCSDS File Delivery Protocol (CFDP)”, Recommendation for Space Data System Standards, CCSDS 727.0-B-1, Blue Book, Jan 2002.

[14] “The Network Simulator - ns-2”, http://www.isi.edu/nsnam/ns/

[15] Gao, J. L., SeGui, J. S., “Performance Evaluation of the CCSDS File Delivery Protocol - Latency and Storage Requirement”, Jet Propulsion Laboratory, December 10, 2004

[16] “The ns Manual”, http://www.isi.edu/nsnam/ns/doc/ns_doc.pdf

[17] Scott, K., Burleigh, S., “Bundle Protocol Specification”, draft-irtf-dtnrg-bundle-spec-04.txt, May 2006.

[18] Akan, O. B., Fang, J., Akyildiz, I. F., “Performance of TCP Protocols in Deep Space Communication Networks”, IEEE Communications Letters, Vol. 6, No.11, pp.478-480, Nov 2002.

[19] Durst, R. C., Feighery, P. D., Scott, K. L., “Why not Use the Standard Internet Suite for the Interplanetary Internet?”, http://www.ipnsig.org/reports/TCP_IP.pdf, Apr 2001

[20] Chen, C., “A Routing Framework for Interplanetary Internet”, to appear in Computer Networks Journal (Elsevier).

[21] 豊嶋　守生, “光宇宙通信技術の研究開発動向”, Performance EvaluSpace Japan Review, No. 44, December 2005 / January 2006

[22] Krupiarz, C., Jennings, E., Segui, J., Pang, J., Schoolcraft, J., Torgerson, L. J., “Spacecraft Data and Relay Management Using Delay Tolerant Networking”, SpaceOps 2006

[23] Kevin K. Choi, GeH rard Maral and Robert Rumeau, “Space link simulator for development of new packet-based space communication protocols”, INTERNATIONAL JOURNAL OF SATELLITE COMMUNICATIONS, Int. J. Satell. Commun. 17, 383-398, 1999.

[24] Choi, kevin K., Maral, Gtrard, Rumeau, Robert, “Space Link Simulator for Development of a New Standard CCSDS File Delivery Protocol”

[25] Daniel C. Lee, and Wonseok Baek, “Expected File-Delivery Time of Deferred NAK ARQ in CCSDS File-Delivery Protocol”, IEEE, TRANSACTIONS ON COMMUNICATIONS, VOL. 52, NO. 8, AUGUST 2004.

[26] Wonseok Baek and Daniel C. Lee, “Analysis of CCSDS File Delivery Protocol: Immediate NAK Mode”

[27] John Segui and Esther Jennings, “Delay Tolerant Networking ? Bundle Protocol Simulation”, 2nd IEEE International Conference on Space Mission Challenges for Information Technology (SMC-IT’06)

[28] Consultative Committee for Space Data Systems, “CCSDS FILE DELIVERY PROTOCOL (CFDP)- Part 1 Introduction and Overview”, CCSDS 720.1-G-2, GREEN BOOK, September 2003.

[29] Consultative Committee for Space Data Systems, “CCSDS FILE DELIVERY PROTOCOL (CFDP)- Part 2 Implementers Guide”, CCSDS 720.2-G-2, GREEN BOOK, September 2003.