51－S 003

コンピュータ・ネットワークJIPNETの研究開発

昭 和 52 年 3 月

JIPDEC

財団法人 日本情報処理開発協会

この報告書は，日本自転車振興会から競輪収益の一部である機械工業振興資金の補助を受けて昭和５１年度に実施した「コンピュータ・ネットワークシステムの研究開発」の成果をとりまとめたものであります。

序

当財団は情報処理技術の研究開発の一環として、昭和４８年度より「コンピュータ・ネットワーク・システム」の研究開発に着手いたしました。

複数個のコンピュータを相互に連結し、ハードウェア、ソフトウェアおよびデータ・ベースの共用を目的とする、リソース・シェアリング・コンピュータ・ネットワークの開発は、米国におけるARPANETの成功をきっかけに、現在世界的に急速に進展しつつあります。

我が国においても、電電公社をはじめ、官庁、大学、各種企業等の多くの機関でそれぞれの立場に応じた研究開発が実施されつつありますが、実用的なネットワーク・システムを構築するには、まだ相当の技術の蓄積と経験が必要であると考えられます。

当財団は異なったメーカの複数台のコンピュータを有しており、それらの相互間リソースを共有するための技術的問題点を具体的なインプリメントを通して解明することを目的として本研究開発を進めてまいりました。

本年度は、当プロジェクトの最終年にあたり前年度までに開発を進めてきた基本ファシリティとしてのネットワークを更に発展させ、ユーザのネットワーク使用機能の増大を目的とするネットワークアクセス用のプロセッサや、そのための言語の設計、分散リソースの制御方式の検討、ネットワーク上での新しい処理形態の実験等、コンピュータ・ネットワークにおける今後の新しい技術分野について検討を行い、本書はそれらの成果をまとめたものであります。

本報告書が、この方面に興味ある方々に広く利用され、我が国情報処理技術向上の一助として寄与できることを念願いたす次第であります。

昭和５２年３月

財団法人　日本情報処理開発協会

会長　　植　村　甲午郎

目　　次

まえがき

1. ＪＩＰＮＥＴの概要 …… 1

1.1 ＪＩＰＮＥＴの目的 …… 1

1.2 ＪＩＰＮＥＴのシステム概要 …… 2

1.2.1 システム構成 …… 2

1.2.2 ＪＩＰＮＥＴの機能 …… 4

1.2.3 プロトコル …… 6

1.2.4 サブネット …… 7

1.2.5 ＮＣＰ …… 10

1.2.6 高位プロトコルの処理 …… 13

2. ＪＩＰＮＥＴの性能測定・評価 …… 15

2.1 概　要 …… 15

2.2 サブネット …… 16

2.2.1 測定目的と方法 …… 16

2.2.2 測定結果 …… 19

2.2.3 検討および評価 …… 23

2.3 ホスト・ホスト・プロトコル …… 30

2.3.1 測定目的と方法 …… 30

2.3.2 測定結果 …… 36

2.3.3 検討および評価 …… 39

2.4 高位のプロトコル …… 50

2.4.1 ファイル転送プロトコル …… 50

A. 測定目的と方法 …… 50
B. 測定結果 …… 54
C. 検討および評価 …… 56
2.4.2 デマンドサービスプロトコル …… 72
A. 測定目的と方法 …… 72
B. 測定結果 …… 74
C. 検討および評価 …… 79
2.4.3 リモートバッチプロトコル …… 84
A. 測定目的と方法 …… 85
B. 測定結果 …… 86
C. 検討および評価 …… 89

3. NAPの開発 …… 93
3.1 開発の背景 …… 93
3.2 NAPハードウェア構成 …… 95
3.3 NAP外部仕様 …… 96
3.3.1 ステーション …… 97
3.3.2 NAPコマンド …… 99
3.4 NAP内部仕様 …… 103
3.4.1 NAPタスク群 …… 104
3.4.2 NAPモニター …… 107
3.5 リモート・ホスト－IMPプロトコル …… 113
3.5.1 データ形式 …… 114
3.5.2 初期状態処理手順 …… 119
3.5.3 データ転送手順 …… 120
3.5.4 終了処理手順 …… 124

4. NFEPの開発 …… 125
4.1 NFEP開発の背景 …… 125
4.2 NFEPの有効性 …… 129
4.3 NFEPの概要 …… 139
4.3.1 NFEPの機能 …… 142
4.3.2 NFEPとホストの結合 …… 156
4.4 プロトタイプNFEPの論理設計 …… 160
4.4.1 ホストとNFEPのインターフェース …… 160
4.4.2 NFEPの実現方法 …… 165
4.4.3 NFEPの論理設計 …… 170
4.4.4 VTAM高位プロトコル …… 177
4.4.5 エンドプロセスの機能 …… 180
第4章の参考文献 …… 186

5. 協同ジョブ処理（ＣＪＰ） …… 189
5.1 ＣＪＰの処理実験 …… 189
5.1.1 ＣＪＰの概要 …… 189
A. ＣＪＰの目的 …… 189
B. ＣＪＰの定義 …… 193
5.1.2 実験計画 …… 194
A. 目　的 …… 194
B. 方　法 …… 194
5.1.3 実験環境 …… 197
A. グラフィック・システム …… 197
B. アプリケーション・プログラム …… 198
5.2 結果の解析と評価 …… 200
5.2.1 結果の推定 …… 200

A. 一般式 …… 201
B. 実験結果の推測 …… 208
5.2.2 CPU時間の解析 …… 218
5.2.3 コストの解析 …… 220
5.2.4 レスポンス・タイムの解析 …… 223
5.2.5 結 論 …… 226
5.3 CJP評価式 …… 227
5.3.1 ターン・アラウンド・タイム …… 228
5.3.2 コスト …… 229
5.3.3 アプリケーション・プログラムのリソース要求 …… 232
5.4 今後の課題 …… 241

6. ネットワーク・ジョブ制御言語：NJCL …… 245
6.1 はじめに …… 245
6.2 NJCLの概要 …… 246
6.2.1 目 的 …… 246
6.2.2 言語の概要 …… 247
6.2.3 簡単なセッション・プログラムの例 …… 251
6.2.4 プログラムの実行形態 …… 254
6.2.5 プログラムの実行環境 …… 255
6.2.6 プログラム実行の完了コード …… 256
6.2.7 サイト指定 …… 257
6.3 文 …… 259
6.3.1 プログラム構造文 …… 259
6.3.2 宣 言 文 …… 260
6.3.3 プログラム実行文 …… 261
6.3.4 インストラクション実行文 …… 271

6.4 NJCLの構文法 …… 275
6.5 インプリメントの方法 …… 289
6.5.1 ネットワーク・オペレーティング・システム …… 289
6.5.2 ネットワーク・フロント・エンド・システム …… 291
6.5.3 ILSCプログラム …… 295
6.5.4 ロード・レベリング …… 298
6.5.5 データ・マイグレーション …… 301
第6章の参考文献 …… 302

7. 分散型データ・ベース …… 309
7.1 はじめに …… 309
7.2 分散型データ・ベースの定義 …… 313
7.3 分散型データ・ベースの背景 …… 316
7.3.1 利用者のニーズ …… 316
7.3.2 技術的背景 …… 324
7.4 分散型データ・ベースのとらえ方 …… 327
7.4.1 機能面での分類 …… 327
7.4.2 分散型データ・ベース・システムの紹介 …… 335
7.5 分散型データ・ベースの現状および今後の方向 …… 349
7.5.1 現 状 …… 349
7.5.2 今後の方向 …… 351
第7章の参考文献 …… 354

8. JIPNETの開発を終えて …… 359
8.1 開発経過 …… 359
8.2 成 果 …… 362
あ と が き …… 366

ま　え　が　き

１９７０年春、米国の代表的なコンピュータ・カンファレンスであるＳＪＣＣに於て、数個の論文からなるARPANET の成果が発表されて大きな反響を呼び、これを契機としてコンピュータ・ネットワークの実用の可能性への第一歩が大きく踏み出された。

ARPANET の創始者であるL.G.Roberts は「コンピュータ・ネットワークとは、お互いにその資源（Resource：ハードウェア、ソフトウェアおよびデータ）を共用（Sharing）し合うことができるように結合され、それぞれ独立した機能を持つコンピュータ・システムの集まり」と定義している。

コンピュータ・ネットワークの基本的に意図するところは、各種のコンピュータのそれぞれ特色あるリソースを相互に共有し合うことによりハードウェア、ソフトウェア、あるいは、データベースにかかわる種々の二重投資を回避するとともに、総合的な強力なリソースの利用を可能とし、且つトータルな信頼性の強化、異機種間の互換性の確立、システム間相互の負荷分担による総合的な処理効率や稼動率の向上、あるいはリソースの分散によるセキュリティ機能の強化等、各種のメリットを産み出そうというものである。

ARPANET の成功をきっかけに、現在世界各国で競ってコンピュータ・ネットワークの計画、開発が行われつゝある。我が国に於ても電々公社をはじめ、大学、官庁、各種企業等、多くの機関でそれぞれの立場に応じた研究開発が実施されつゝあるが、ＪIPNETは当財団が数個の異った種類のコンピュータ・システムーFACOM、HITAC、NEACーを持つ立場を利用して、それらの相互間のリソース共用のための技術的問題点を具体的なインプリメントを通して解明する事を目的としている。

本プロジェクトは１９７３年度より着手し、１９７４年末にはデータ交換用のサブネットと、それを通しての２つのHOSTコンピュータ間でのコミュニケーション

が可能となり、これを第一バージョンとして、その後各種コントロール機能の効率測定をおこない、その結果によるシステムの評価改良、信頼性の向上、１９７５年３月にリプレースされ新しいHOST コンピュータの結合等を含め、１９７５年度中にバージョン２が完成した。１９７６年度は本プロジェクトの最終年度にあたり、前年度迄に完成した基本ファシリティを利用し、ネットワークの総合的性能測定および評価、ユーザのネットワーク使用効果の増大を目的とするNAP（Network Access Processor ）のインプリメント、新しいホストコンピュータのネットワーク加入を容易とするNFEP（ Network Front End Processor）の実現の可能性の検討、ネットワーク上における新しい処理形態であるCJP（Cooperated Job Processing）の実験、ユーザの操作性向上のためのNJCL（Network Job Control Language ）の言語仕様の設定、今後の焦点の一つである分散型データベースの問題点の検討など、コンピュータ・ネットワークにおける今後の新しいテーマにつき研究開発を行なった。本報告書はこれらの結果をまとめたものである。

1．JIPNETの概要

1．JIPNETの概要

1.1　目　的

ＪＩＰＮＥＴ（ＪＩＰＤＥＣ　Integrated Project Network）は昭和４８年度より４ケ年計画で実現すべく手掛けられた、リソース・シェアリング・コンピュータ・ネットワークの研究開発プロジェクトである。

ＪＩＰＮＥＴは下記に示すような実用と実験の両方の目的を有しており、ネットワークの規模は、協会内の国産３機種を結合したインハウス・システムではあるが、その構成は広域の分散型リソース・シェアリング・コンピュータ・ネットワークのモデルを目指している。

先づ実用上の要求としては次の様なものがある。

(1)　ＪＩＰＤＥＣに現在設置されている異なる３機種のハードウエア、ソフトウェア、データセット等の限界を除き、総合したリソースを利用したい。

(2)　プログラムやデータ・ファイルの互換性を標準化とは異なるアプローチにより確保したい。

(3)　漢字入出力、図形処理装置、マークシート・リーダ等の特殊周辺装置をどのＣＰＵからも共用したい。

(4)　数年前から実施しているＴＳＳに於て、端末からどのシステムでもアクセス出来るようにしたい。

(5)　現在機種によるロードのアンバランスがかなりあるがなるべく平均化したい。

(6)　ある機種のダウンによって不可能になっていた処理を他の機種でカバーする。

(7)　コンピュータ・ネットワーク・システムの効率評価のデータを得る。

一方、実験的な目的としては次のようなものがある。

(1) 個々のシステムの、結合したが故の負荷の増大、機能や処理効率の低下などを実測する。

(2) 個々のシステムの結合前と結合後における運用上の互換性がどの程度保てるか。変更せねばならぬ点がどの程度であるかを体得する。

(3) 実用上、ユーザレベルのプロトコル（protocol）としてどの様なものが必要であるか、またそれらの処理が異なる機種間でどの程度効率よくおこなわれるか。

(4) ネットワーク利用のメリットをコスト効率という観点からとらえる。

(5) 異機種間の分散型データベースが常識的な効率下においてマネージ出来る方式および限界を研究する。

(6) 異機種間のオートマチックなハードウェアおよびソフトウェア・シェアリングの方法および有効性などを研究する。

コンピュータ・ネットワークの本質は、夫々特色ある各種リソースを相互に有効に利用することにある。真に効果のあるリソースの共用を実現するには、夫々のＨＯＳＴ側でのリソース提供のサポート機能が、経済性、効率性等の諸点からも充分納得し得る質を保つ事が要求される。しかしながら、従来のＯＳとの関連もあり、現在のネットワークに於てはこの点に関しては必ずしもまだ充分満足し得る状態にあるとは言い難い。ＪＩＰＮＥＴに於ける主たる目標は、この事実に焦点を当て、特に国産コンピュータを対象とし、今後我国に於けるコンピュータ・ネットワーク利用の実現可能性と実用性を検証し、その促進のための考察を具体的なレベルで行うことにある。

1.2 ＪＩＰＮＥＴのシステム概要

1.2.1 システム構成

ＪＩＰＮＥＴは分散型ネットワークのモデルとして現在は図１－１に示すように、３つのＨＯＳＴコンピュータとＮＥＡＣ３２００／５０をＮＯＤＥプロセッサとする３ＮＯＤＥのサブネットおよび各種の最新端末群をＮＯＤＥ

に接続した形で形成されている。ＮＯＤＥ間は４８Ｋbps の高速通信回線で接続し、HOST－NODE 間は８ビット・パラレル転送のインタフェース・アダプタを製作し、セレクタ・チャネルにより結合している。またＮＯＤＥ２に結合されているＵ200はＮＡＰ（Network Aceess Processor）と呼びARPANETのANTS（ARPA Network Terminal System）に相当する。

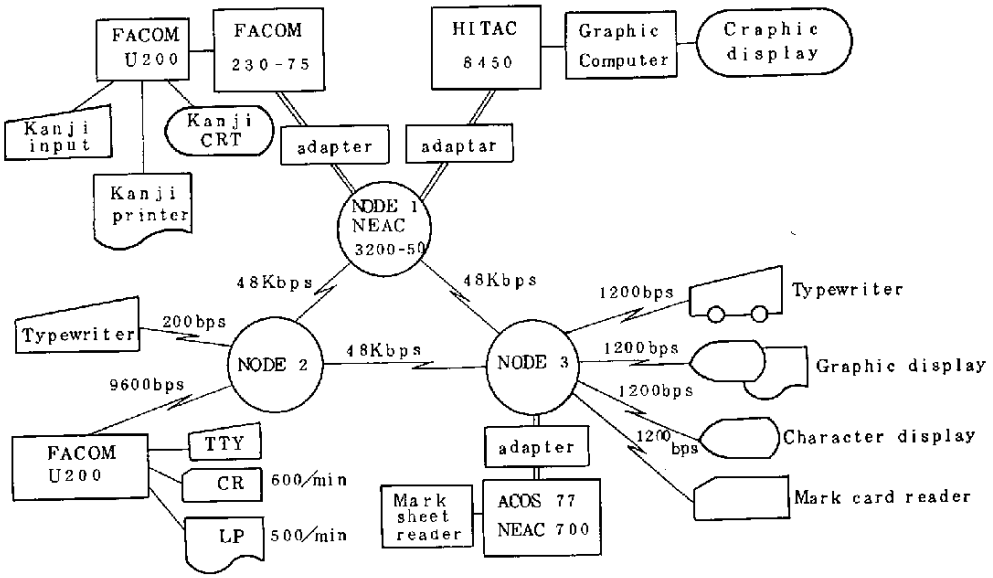


図１－１　ＪＩＰＮＥＴ構成図

システムの構成にあたって、以下のような諸点に留意した。

(1)　コンピュータ・ネットワークとしての基本的な機能が全て実験可能な構成であること。

(2)　コンピュータ・コンプレックス・システムとして、当協会特有の利用法の実験にも適した構成であること。

(3) 我が国での実用性が比較的高いと思われるARPANETでいうＴＩＰ（Terminal IMP）あるいはANTS等を重視した形をとった。

(4) 将来、他のシステムあるいはネットワークとの結合などの拡張が容易であること。

(5) 従来、各HOST コンピュータで行ってきた作業が、何等変更なしに引き続き稼動可能のこと。

なお現在のソフトウェアの変更なしに１６ＮＯＤＥまでは拡張可能で、各ＮＯＤＥには３つまでのＨＯＳＴが結合し得る。

1.2.2 ＪＩＰＮＥＴの機能

ＪＩＰＮＥＴにおける主な処理パターンの例を図１－２に示す。①～④は２つのＨＯＳＴを対象としたもので、例えば

	HOST A	HOST B	HOST C	HOST D	端末
①	S,O	E,P,D			
②	S,O,D	E,P			
③	S,O,D,SP	E,C			
④	S,O,C,E	SP,D			
⑤	S,O	E,P	D		
⑥	S,O,D	E,C	SP		
⑦	S,O	SP,D	E,C		
⑧	S,O,SP	D	E,C		
⑨	S,D	E,P	O		
⑩	T,D	E,P			
⑪	T	E,P,D			
⑫	T	E,P	D		
⑬	E,P,D				T
⑭	E,P	D			T
⑮	E,P	D1	D2		T
⑯	T	E,P	D1	D2	
⑰	S,O	E1,P1	E2,P2	D	

S：制御情報の入力
O：結果の出力
E（またはE1、E2）：実行
C：コンパイルまたはアセンブル
SP：ソース プログラム ファイル
P（またはP1、P2）：実行形式 プログラム ファイル
D（またはD1、D2）：データ ファイル
T：端末（SとOを含む）

図１－２ 処理パターンの例

①はHOST Aから制御情報をHOST Bに送り(S)、HOST Bにあるプログラム(P)およびデータファイル(D)を使用して処理(E)を行うことを依頼し、結果をHOST Aに戻し出力(O)するケースである。また②はAからBに同様に処理を依頼するが、その際、データファイルがAにあり、これを予め、あるいは処理中にBに送りながら処理を実行する。また③はその際ソースプログラムファイル(SP)もAにあり、これをBに送りコンパイル(C)も依頼する。

次の⑤から⑨は、A、B、Cの3つのHOSTにわたって処理を行なう場合の例で、たとえば⑤は、Cにあるデータファイルを使用してBにあるプログラムで処理をすることをAから依頼する形である。また⑨は、Cにある特殊出力装置(たとえば漢字プリンタ)を共用するような場合の例で、Aにあるデータを Bに送って処理を行ない結果をCに出力する。

⑩から⑭は、リモートバッチや会話型端末からネットワークにアクセスする形で、端末(T)はここでは制御情報の入力(S)と結果の出力(O)の両機能を含むものとする。

このうち⑩から⑫は、あるHOST(この場合A)につながる端末から他HOST(この場合B)の会話型あるいはリモートバッチ処理の使用であり、⑬、⑭は特定のHOSTを経由せず、NODE に直接結合された端末からのネットワーク使用である。

⑮、⑯、⑰は、ネットワークのやや進んだ使用法であり、⑮、⑯は2つのHOSTにデータファイルが分散しており(D1とD2)、⑰は処理自体が2つのHOSTで分担(E1とE2) されている。これらはそれぞれ、分散型データベースと複数HOSTの協業により、1つのジョブを実行する形の例であり、より多くのHOSTを対象とすることももちろん考えられる。

1.2.3 プロトコル

図１－３にＪＩＰＮＥＴプロトコルの階層を示す。

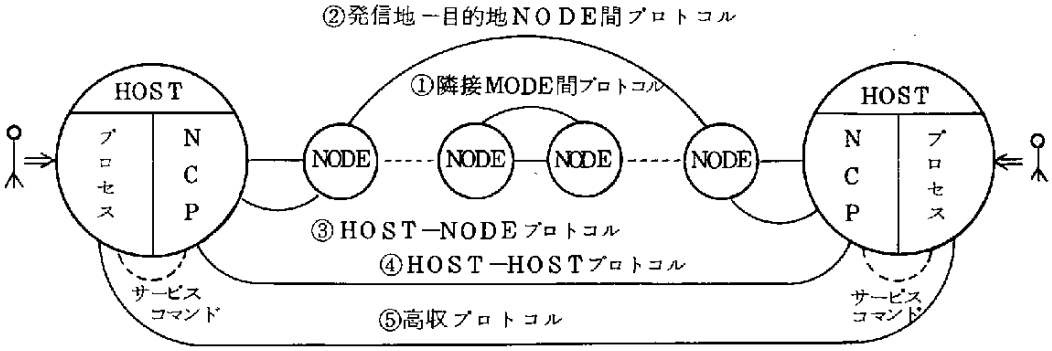


図１－３　ＪＩＰＮＥＴのプロトコル

① 隣接ＮＯＤＥ間プロトコル

隣接したＮＯＤＥ間での伝送データのフォーマット、受信確認方式、伝送エラーの検出と処置、再送の手続、パケットのルート選択および相互の障害と回復の検出と処置等に関する規約がある。

② 発信地ＮＯＤＥ－目的地ＮＯＤＥ間プロトコル

発信地と目的地間のEnd-to-End処理にともなう規約であり、メッセージのパケット分割やリアセンブル（reassemble）、メッセージの紛失、重複等に関する伝送エラーの検出と処置、サブネット内のフローコントロール、伝送されたメッセージの順序付けなどに関する規約がある。

③ ＮＯＤＥ－ＨＯＳＴプロトコル

このプロトコルはサブネットとＨＯＳＴの間のデータの授受に関する規約であり、データフォーマット、送受信の手順および確認の方法、相互の障害状態の検出などに関するものがある。

④ ＨＯＳＴ－ＨＯＳＴプロトコル

このプロトコルは、ＨＯＳＴどうしが交信し、相互にコントロール情

報およびデータを転送し合い、ネットワークを通して各種の処理を行なうための基本となる規約である。たとえば、ＨＯＳＴ間のデータフォーマット、相互交信のためのＨＯＳＴ間のコネクションの確立、コネクションを通したデータの転送、データ転送に伴うＨＯＳＴ間のフローコントロール、相手側の障害の検出などに関するものである。

⑤　高位プロトコル

ネットワークユーザが直接使用するコマンドの形で規定されるハイレベルのプロトコルで、現在ＤＳＰ（Demand Service Protocol）、ＲＢＰ（Remote Batch Protocol）およびＦＴＰ（File Transfer Protocol）をサポートしている。

これらのうち①～④はシステム・レベルのプロトコルである。これらのプロトコルはＮＯＤＥプロセッサ内のＳＣＰ（Subnet Control Program）と各ＨＯＳＴ内のＮＣＰ（Network Control Program）により分担して処理されるが、プロトコルの設定におけるＨＯＳＴとサブネットの負荷分担は、ＨＯＳＴの負荷軽減を考慮し、メッセージのパケット化、リアセンブリ、伝送エラーのリカバリ、シーケンシング等、End－to－End の殆んどをサブネットの分担としている。

1.2.4　サブネット（Subnet）

パケット交換方式を採用しており、ネットワーク全体の信頼性、会話型処理および大量データ転送における効率性、ＨＯＳＴやＮＯＤＥ数の増加、任意の回線速度等に応じられる拡張性および可変性を重視し設計した。全体的にARPANETの実績をかなり利用したが、各ＮＯＤＥにサブネットのトポロジを意識させている点、長短メッセージのパイプ管理、ルーティングのアルゴリズム、発信地、目的地間のメッセージのＡＣＫをＨＯＳＴ間で行なっている点等に関しては、かなり異なっている。Artificial Traffic Generaterを用いて測定した結果、最大スループットは単一パ

ケット・メッセージ（１４４byte以下）の場合、５８K bit/sec、多重パケット・メッセージ（１１５２byte以下）の場合、６４K bit/secであった。またround trip timeは１００byteメッセージの場合1 Hopで２７ms、2 Hopで５２msであり、２０Hopでは約0.５秒と推定される。

以下に各機能の概略を述べる。

(1) 蓄積交換機能

サブネットはＨＯＳＴから受取ったメッセージをパケット化し、蓄積交換を行って目的地ＮＯＤＥに届ける。目的地ＮＯＤＥではこれらのパケットをリアセンブルし、元のメッセージに再組立し、目的地ＨＯＳＴに渡す。隣接ＮＯＤＥ間の伝送確認は基本的にはパケット単位のＡＣＫ信号によって行われ、両ＨＯＳＴ間のメッセージの伝送確認はＲＥＣＥＩＶＥ信号によって行われる。隣接ＮＯＤＥ間は回線を論理的な4チャネルに分割し、回線の利用率を上げている。なおＡＣＫ、ＲＥＣＥＩＶＥの返送は原則として逆方向のメッセージに便乗させている。

(2) ルーティング

ルーティングの手法には種々のものがあるが、パケットの高速伝送、トラフィック変動に対する適応性、トポロジー変化への追従性、NODEの計算負荷の軽減等を考慮して、「Shortest Q＋Bias」の手法を採用した。この手法は定常状態では外部からは情報をとり入れずに、NODE内部における動的な送信待ちキューの長さと、ネットワークの静的構造を示すバイアス値との和を各出力回線ごとに比較し、その最小のものを選びパケットを送り出すというものである。バイアス値は、ＮＯＤＥおよび回線のアップ・ダウンにともなうサブネットの重大な変化の際のみ動的に計算し更新している。

(3) フロー・コントロール

フロー・コントロールはARPANETのパイプの概念を採用し、発信地―目的地ＮＯＤＥ間にパイプという論理的なパスを設定し、このパイ

プの容量を制限することによりフローを制御している。ＪＩＰＮＥＴでは単一パケット・メッセージ（１４４バイト以下）用のＳ－Pipe と多重パケット・メッセージ（１１５２バイト以下）用のＬ－Pipe との２種類のパイプを設け、夫々別個に管理している。現在Ｓ－Pipe には４個までの単一パケット・メッセージを同時に流すことができ、多重パケット・メッセージは目的地においてバッファがアロケートされた事を確認した後Ｌ－Pipe を通し、一時に一個だけ流すことができる。これにより、短かいメッセージの応答性が長大メッセージのディレイにより受ける影響を少なくしている。

(4) シーケンシングとエラー検出

サブネットに入る各メッセージには、各パイプごとに０～２５５までの番号がラウンディング（rounding）につけられて送り出される。この番号により目的地ＮＯＤＥではメッセージのシーケンシング（sequencing）を行っており、発信地―目的地ＮＯＤＥ間でのメッセージの紛失、重複あるいはＲＥＣＥＩＶＥの紛失等のエラー検出にもこのシーケンシング機構を利用している。

(5) ロック・アップ

サブネットでは次の４種のロック・アップ（lock up）を想定し対策を考慮した。

- store & forward lock up
- sequeneing lock up
- reassemble lock up
- ＨＯＳＴ／ＮＯＤＥ lock up

これらの lock up に対してはＮＯＤＥ内におけるバッファの管理アルゴリズムを十分に検討して防御手段を講じた。なお、sequencing lock up は一つの目的地にメッセージが集中する場合に発生するものであり、ＨＯＳＴ／ＮＯＤＥ lock up はＨＯＳＴ－ＮＯＤＥ間のアダプタが半二重の

ために発生するものである。

(6) 障害対策

サブネットの障害対策は次のような点を基本方針としている。

- できる限り早期に発見する。
- 障害の発生およびその回復過程で、サブネット全体の運転は中断させない。
- 一部の構成要素が障害で失われても、それを除いた環境下での運転が可能なこと。
- 障害の回復は自動的に行われる。
- 異なった要素間における障害の切りわけが容易であること。
- ＨＯＳＴ障害の影響をネットワーク全体に及ぼさぬとともにネットワークの障害を各ＨＯＳＴのローカル処理に影響させない。

1.2.5 ＮＣＰ

ＮＣＰ（Network Control Program）は各ＨＯＳＴに存在するネットワーク用のコントロール・プログラムである。

ＪＩＰＮＥＴは汎用のネットワークであり、ＨＯＳＴ間の通信は各処理プロセス間通信の形で実現している。即ち図１－４に示す様に各プロセスは夫々のＨＯＳＴのＮＣＰを通して交信をおこなう。

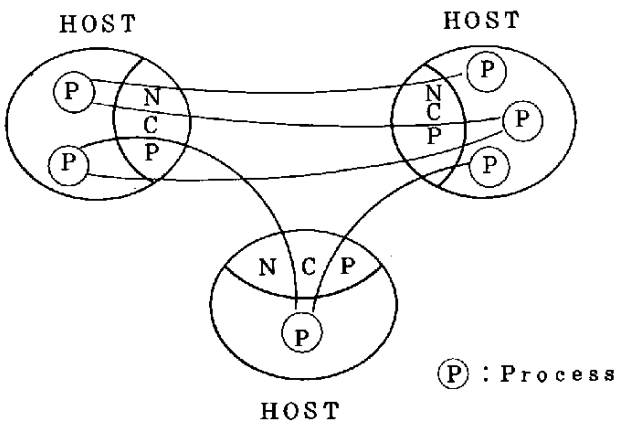


図１－４　プロセス間コミュニケーション

各ＮＣＰは既存ＯＳのもとに付け加えられ、FACOMではサブモニタとして、他の２機種ではユーザ・プログラム、即ちＨＩＴＡＣではリアルタイム・ジョブとして、ＡＣＯＳでは特権スレーブとして作成している。

ＮＣＰはＨＯＳＴ－ＨＯＳＴおよびＨＯＳＴ－ＮＯＤＥ プロトコルの規定にもとづく処理をおこなうものであり、次のような機能を持つ。

(1) プロセス間のコミュニケーション

バーチャル・コール（Virtual Call）方式にもとづき、プロセス間のコネクションの確立とその閉鎖、コネクションを通してのデータの授受、プロセス間の同期などの処理を行なう。コネクションの確立とは、２つのＨＯＳＴのそれぞれのプロセス間で、データ流のためのロジカルなパスを作ることである。各プロセスにはＨＯＳＴ番号とユーザ番号とからなるネットワークに対してグローバルな idがつけられ、ＪＩＰＮＥＴではこれを port-idと呼ぶ。１つのコネクションは２つのプロセスのそれぞれの pord-idのペアに対して確立される。

また１つのコネクションで両方向のデータ流を許す。

(2) ＨＯＳＴ間のフロー・コントロール

バッファ確保方式を採用している。ＨＯＳＴ間で１つのコネクションが確立すると、受信側ＨＯＳＴはメッセージを受信するためのバッファを１個以上（各ＨＯＳＴのその時点のバッファ事情により２～１０個）確保してその個数を送信側ＨＯＳＴに通知する。送信側は受信側において準備されたバッファ個数分のメッセージ送信を完了すると、新たにバッファ確保の通知がくるまでそのコネクションに対するメッセージ送信を一時中断する。受信側は出来るだけ速かにバッファを確保し、送信側に通知する。

１つのＨＯＳＴで１時に確立し得るコネクション数は３０前後であり、ＨＯＳＴあたり 100ないし 150パケット分のバッファを持つ。

(3) コントロール・コマンドとサービス・コマンド

2つのHOSTのNCP間は、コントロール・コマンド、即ちHOST−HOSTプロトコル・コマンドにより交信される。一方ユーザ・プロセスとNCP間にはサービス・コマンドが設けられており、サービス・コマンドには多重コネクション管理、フロー・コントロールなどのマクロ機能を含んでいる。表1−1に両コマンドの種類を、図1−5に両者の関連の例を示す。(b)はMがユーティリティ・プロセスの場合である。

表1−1 コントロール・コマンドとサービス・コマンド

コントロール・コマンド		サービス・コマンド	
RFC	コネクション確立要求	OPEN	コネクション確立要求
CLS	コネクションの閉鎖	CLOSE	コネクションの閉鎖
ALLOC	メッセージ・バッファ確保通知	READ	データの受信
POST	プロセス間の情報の通知	WRITE	データの送信
SPOST	システム情報の通知	LISTEN	コネクション確立要求待ち
LOADE	プロセスの起動	WAIT	イベント待ち
DELP	プロセスの消滅	POST	プロセス間の情報の通知

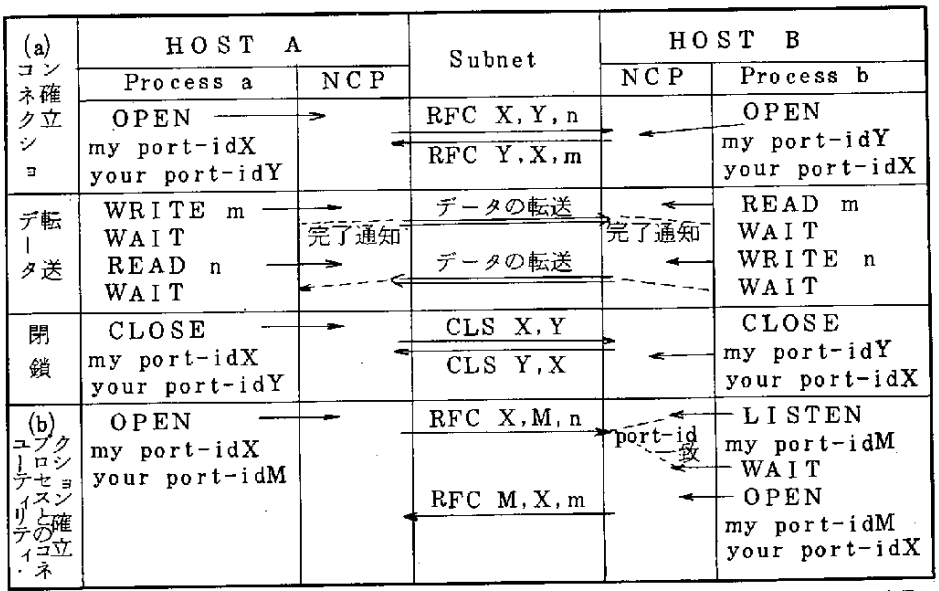


図1−5 両コマンドの関連 (n,mは受信側コネクション番号)

1.2.6 高位プロトコルの処理

現在は、ＤＳＰ（Demand Service Protocol）、ＲＢＰ（Remote Batch Protocol）およびＦＴＰ（File Transfer Protocol）がサポートされている。

各ＨＯＳＴ内のプロセスにはユーザ・プロセスとシステムレベルのユーティリティ・プロセスとを設け、高位プロトコルの処理は拡張性を重視し、夫々ユーティリティ・プロセスとしてサポートしている。従ってＨＯＳＴは夫々のプロトコル毎にＵＳＥＲプロセスとＳＥＲＶＥＲプロセスとを持つ。図１－６に各プロトコル処理の概念図を示し、図１－７、１－８はそれに対応するジョブ・デックおよび会話の例である。

```
¥ ID  ××××
# USER NO. ××××
# HOST B
# OUTPUT S
# END

 JCL for HOST B

¥ JEND
```

¥ HOST AのＪＣＬ

ＲＢＰコマンド

図１－７　ＲＢＰのジョブデック

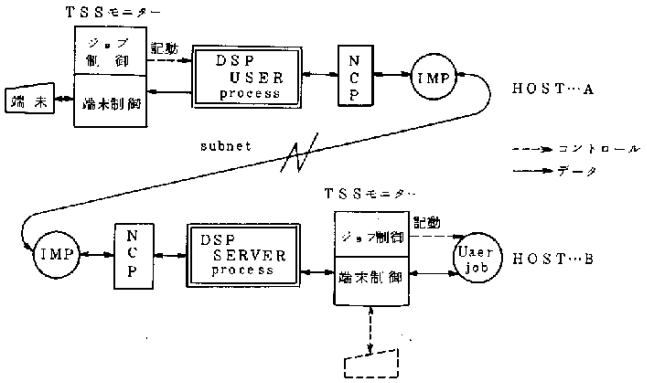
```
BK ?   @OPEN   DSP   B
       USER NO =   ×××××
       DSP SERVICE START

           HOST－Bとの会話

BK ?   @CLOSE
       DSP SERVICE END
```

under lineはユーザの入力

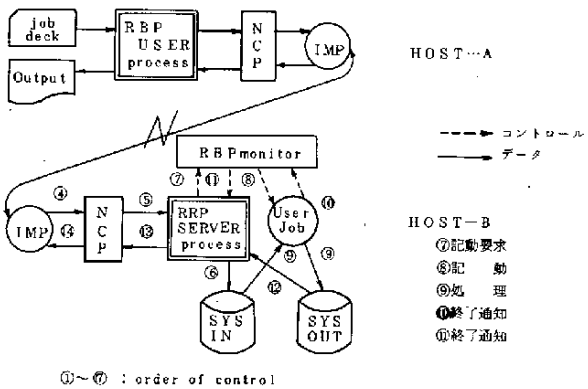
BKはBreak KeyのKey in

図１－８　ＤＳＰの会話の例

DSP処理概念図



RBP処理概念図



①～⑦ : order of control

FTP処理概念図

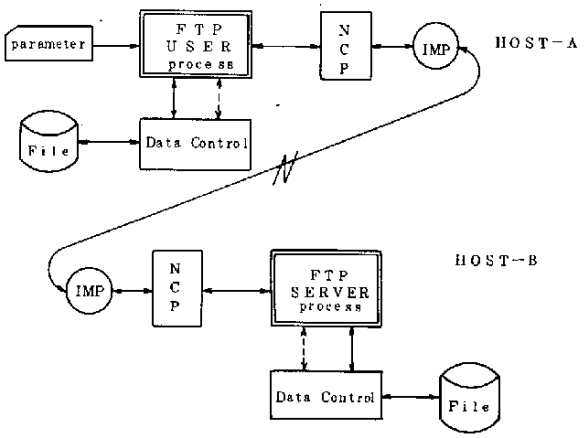


図１－６　高位プロトコル処理の概念図

2．JIPNETの性能測定・評価

2．JIPNETの性能測定・評価

2.1 概 要

システムの作成は、プラニング、設計、インプリメンテーション、テストという一連のプロセスを経ておこなわれる。

しかしながら、システムの規模が大きくなるにつれて、必ずしも一通りの経過で最適なシステムを完成させることはむづかしい。このため、出来上ったシステムに対して性能測定および評価をおこなって、最適システムへ近づく可能性があるかどうかの判断をおこなわなければならない。

測定の結果、当初目標としたシステム基準に達していない場合、あるいは更に改良の余地があると推定される場合、再三再四、設計、インプリメンテーション、測定、評価のプロセスが繰り返されることがあり得る。

使用者にとっても、設計者にとっても、実際に稼動しているコンピュータ・ネットワークがどの位の性能があるのかは、大きな問題である。ＴＳＳ利用者にとっては、会話的な短いメッセージのラウンド・トリップ時間（round-trip time）がどの位なのか、ファイル転送のような大量データ伝送を行う使用者にとっては、スループットがどの位なのかが関心事である。設計者にとっては、これら以外にネットワーク全体でどの位の性能を呈するかが関心事となる。

ＪＩＰＮＥＴの開発においても同様に、大量データ転送のスループット、会話形メッセージのラウンドトリップ時間、ホストのＣＰＵ使用コストが開発者の関心事となった。

ＪＩＰＮＥＴは多層構造よりなっているために、それぞれの性能は各層ごとに測定評価されるべきである。

この測定評価においては、図２－１に示すような３つのレベルにおける測定評価をおこなった。この３つのレベルとは、サブネットレベル、ホスト－

ホスト・レベルおよびハイレベルであり、以下の項にその結果を示す。

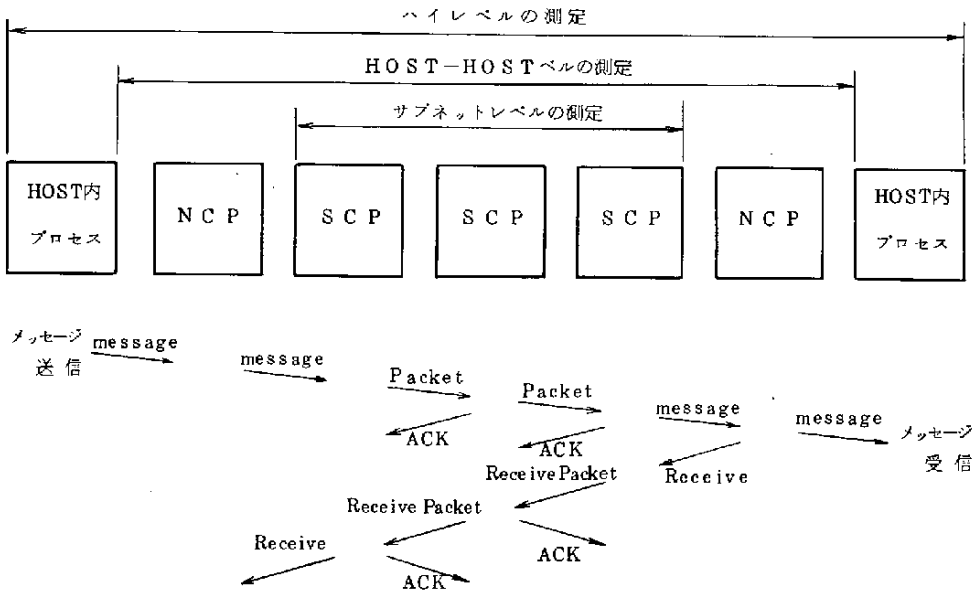


図 2 − 1

2.2 サブネット

2.2.1 測定目的と方法

A 測 定 目 的

サブネットにおいては、メッセージ伝送のスループットと、ラウンドトリップ時間の測定をおこなった。

サブネットは、メッセージあるいはパケットを効率良く目的地に送りとどけるのが役割である。この意味においてスループットおよびラウンドトリップ時間を測定することは、サブネットの性能を測定するに等しいということができる。

さらに、この測定結果よりそれぞれのＩＭＰにおいて、メッセージあるいはパケット当りどれだけのリソース（主としてＣＰＵ時間）が使用されているかを推定することができ、ＩＭＰのパケット処理能力を知ることができる。

B　測　定　方　法

サブネットの処理するメッセージは、S-pipeメッセージ、L-pipeメッセージの２種類である。

S-pipeメッセージとは、１１５２ビット（１４４バイト）以下のメッセージであって、会話形処理のようにラウンドトリップ時間が小さいことを望まれるものである。

L-pipeメッセージとは、９２１６ビット（１１５２バイト）以下のメッセージで、ファイル転送などのように大量データ転送のスループットを重要視されるものに使われる。サブネット内においてL-pipeメッセージは、１４４バイトのパケットに分割されて送られ、目的地においてリアセンブルをおこないメッセージにされる。

サブネットにおいて、測定の変数となる要素に、トポロジーがある。ＪＩＰＮＥＴのサブネットは、３ノードと３回線でなっており、任意の２つのホスト間におけるサブネットの形態としては、図２－２に示す３つのパターンだけが存在するだけである。

以上述べたように、本測定においてはメッセージのスループットおよびラウンドトリップ時間を、S-pipe およびL-pipeメッセージ、サブネットの形態を変えて測定をおこなった。

本来サブネットはホストよりメッセージを受け取って指定されたホストに正確にメッセージを渡たすのが役割りであるので、自分自身でメッセージを発生することはない。このためにサブネット中のプログラムに変更を加えてメッセージを発生する機能、receive を作成する機能を追加した。

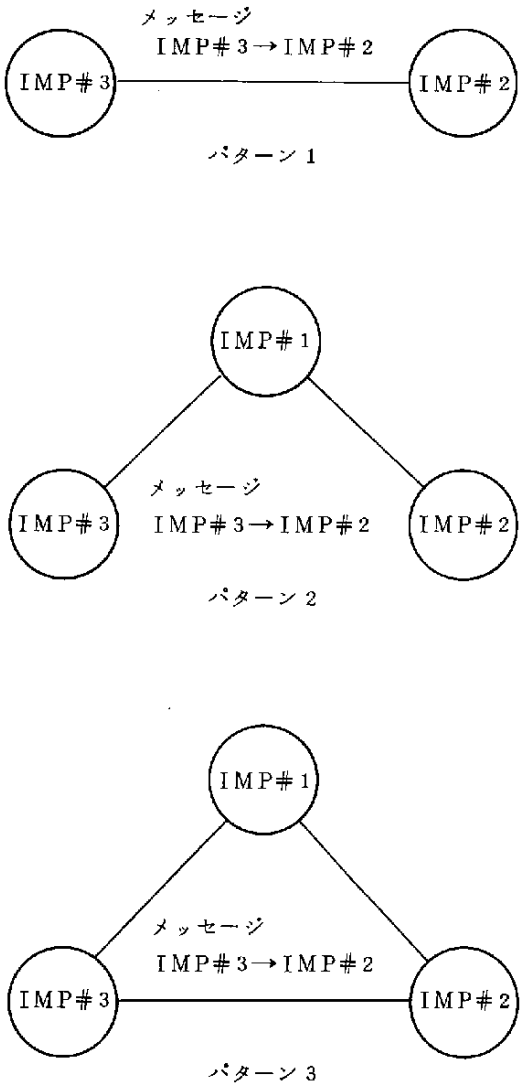


図2－2　サブネットの測定パターン

メッセージを発生する機能をはたすプログラムをartificial traffic generatorという。これは200msecあるいは100msec毎に任意の個数のメッセージを発生することができる。

この機構はtime outタスク中に組み込まれており、fastタイムア

ウトがかかると、バッファの余裕があるかぎり（バッファの残り個数が30個になるまで）指定された個数ずつのメッセージを作り出す。

このメッセージの行き先は指定されたＩＭＰであり、これを受け取るのはホスト／ＩＭＰ３タスクである。このタスクではメッセージを受け取ると、受け取ったメッセージ数をカウントし、receive を作成して送り返す。

送り返えされたreceive は、メッセージを発信したＩＭＰのパイプ管理において、処理が終わり次第捨てられた。

スループットの測定は、ホスト／ＩＭＰ３タスクがおこなっている受け取ったメッセージのカウンタを０にして、１００秒後に再びカウンタを参照することによりおこなった。１００秒の計測は、ストップウオッチでおこなったが、この測定の誤差は±０.２％以内である。

ラウンドトリップ時間はサブネットにおいては、メッセージをサブネットに流入させてからreceive を受け取り、ホスト／ＩＭＰタスク渡されるまでの時間と定義して測定した。

この測定は、パイプ管理がメッセージをサブネットに入れる時の時刻を（ハードウエアタイマーを利用しているのでmsecまで計測できる）憶えておき、対応するreceive を受け取りホスト／ＩＭＰタスクに渡たそうとするときの時刻と比較し、その差をカウンタに加算して入れる。このとき他のカウンタを＋１する。

これらのカウンタをある時点で０にして、４秒後に両者を参照することにより計測した。

2.2.2　測　定　結　果

図２－３から図２－６までにＳ-pipe メッセージのスループット、図２－７にＬ-pipe メッセージのスループットを示す。

また、図２－８にＳ-pipe メッセージのラウンドトリップ時間、図２

－９にＬ－pipe　メッセージのラウンドトリップ時間を示している。

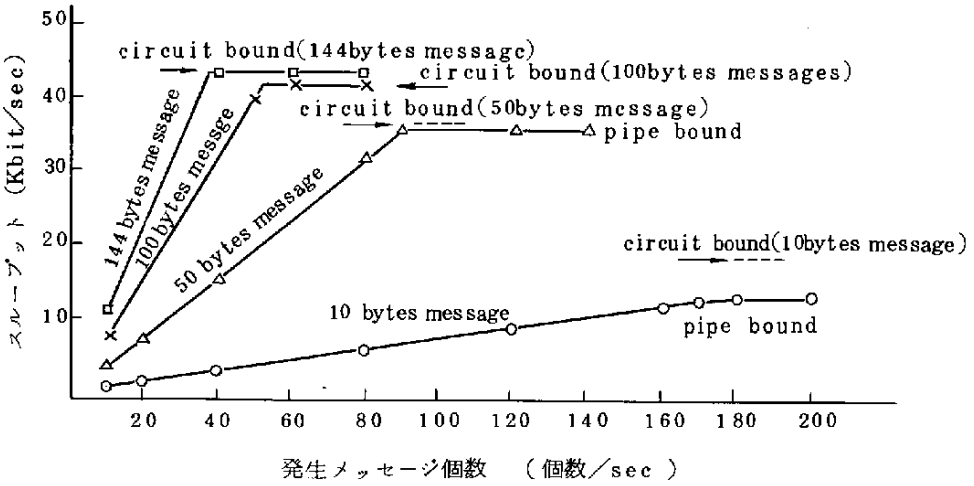


図２－３　S－pipeメッセージのスループット　　パターン１

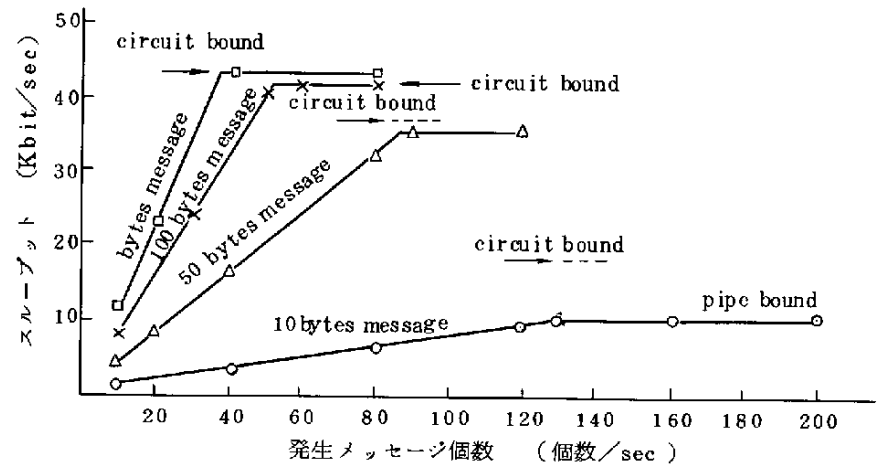


図２－４　S－pipeメッセージのスループット　　パターン２

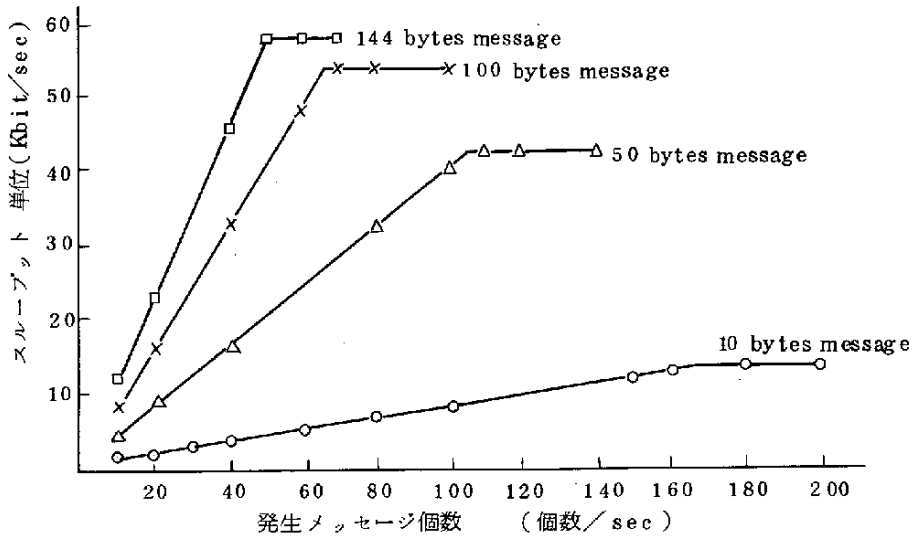


図2－5　S－pipeメッセージのスループット　パターン3

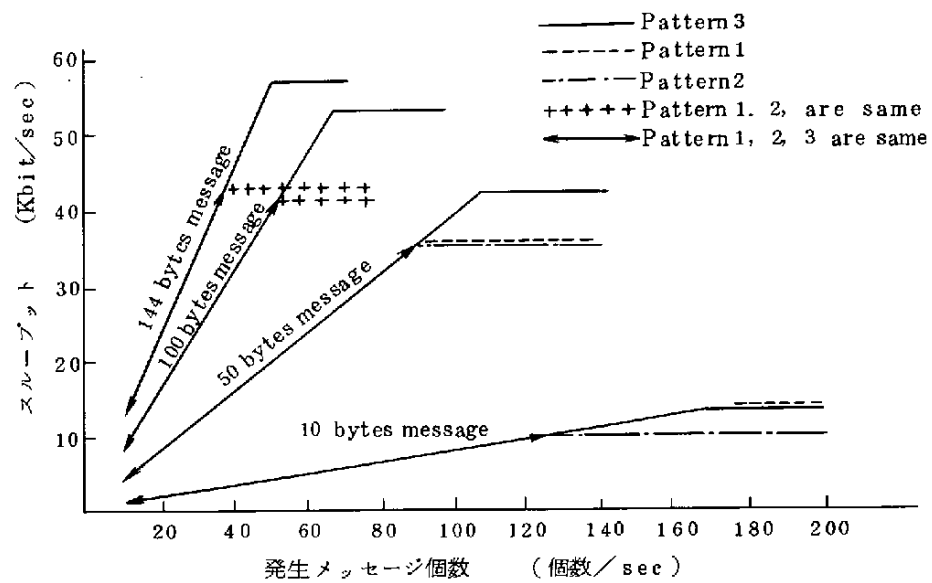


図2－6　S－pipeメッセージのスループット

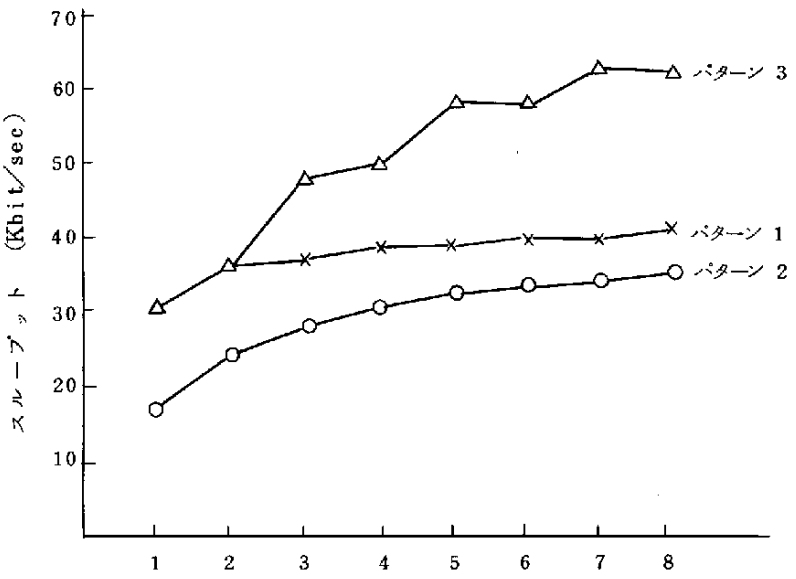


図２－７　Ｌ－ｐｉｐｅメッセージのスループット

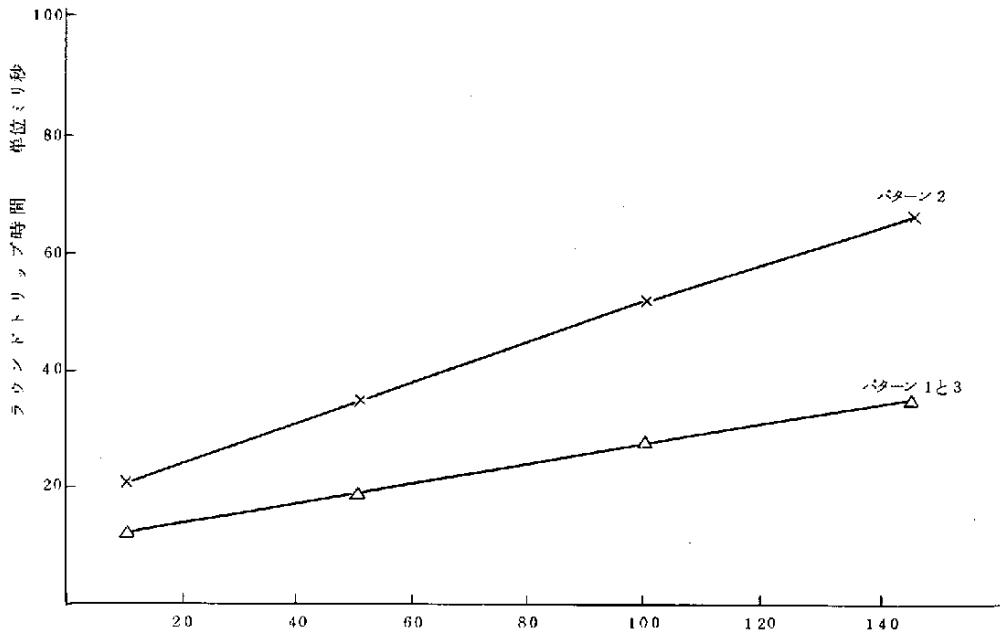


図２－８　Ｓ－ｐｉｐｅメッセージのラウンドトリップ時間

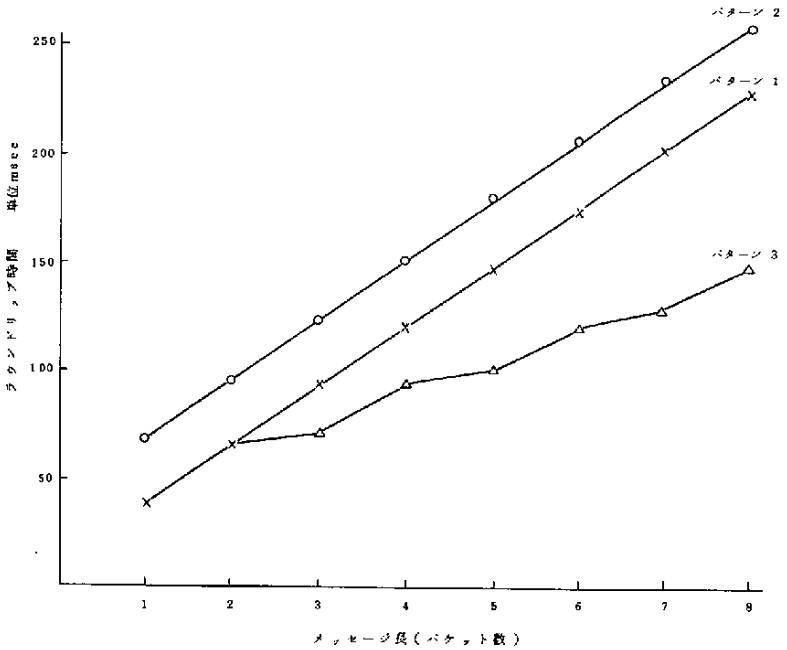


図２－９　Ｌ－pipeメッセージのラウンドトリップ時間

2.2.3　検討および評価

A　S－pipe　メッセージのスループット

図２－３～図２－６をみれば明らかなように、１０バイトメッセージのように短いメッセージにおいては、サブネットのフロー制御機構によって伝送量がおさえられるが、メッセージ長が長くなるにしたがって４８Kbit／sec の回線容量によって伝送量がおさえられるのが分かる。

またパスが１個のものと２個のものとを比較すると、パスが１個のものが回線バウンドになっても、パス２個の場合には、目的地にゆくためのルートがルーティング機構によって選択されるために、フロー制御によるバウンドになっている。

フロー制御の機構によって、伝送量がおさえられているといっても、

パターン２において１０バイトメッセージを毎秒１２９個、パターン３において１４４バイトメッセージを毎秒５０個伝送できるということは、会話形処理において主として使われることを考えれば十分な性能であると考える。

（図２－６）において、１０バイトメッセージは、パターン１がスループットが一番大きく、次にパターン３、最後がパターン２である。フローコントロールにより伝送量がおさえられている場合には、メッセージのラウンドトリップタイムが小さい方が伝送量は大きくなるので、パターン２が他のものよりスループットが小さいのは納得できるが、パターン１がパターン３より大きい点に興味がある。考え得ることとしては、再伝送のアルゴリズムを変えると、この状態が若干変化するところから、この測定の過程で現実には再伝送がかなり発生し、そのアルゴリズムが影響しているのではないかという事と、modem outタスクの優先度がやや低いので、パターン３に於て、経路が２つになると割り込みが一定時間内に多発し、そのオーバーヘッドが大きくなるため、modem outタスクに対してＣＰＵの供給が充分おこなわれないせいではないか等の理由が考えられる。

B　L－pipe メッセージのスループット

図２－７の測定結果のグラフが示すように、L－pipe のスループットは、８パケットメッセージのパターン３における値が６２.７Kbit／secと予想外によくない。これは、L－pipe のフロー制御機構に問題がある。

L－pipe のフロー制御は、ある２つのＩＭＰの一方から他方に向かうL－pipe メッセージは、同時には１個しかサブネットに存在させないこと、目的地にバッファを予約してからメッセージを送り始めるという機構になっている。

この２つの機構のうちで、後者が特に重要な意味を持っている。一対

のＩＭＰの一方から他方へ向かうＬ－pipe メッセージは同時に１個しか許さなくしたのは、バッファ予約にからめてのことだからである。

このようにサブネットにおけるメッセージのリアセンブルは、結果的にメッセージ伝送のスループットを必要以上におさえる原因になるばかりでなく、メッセージ処理プログラムの構造も複雑にしている。

これを避ける方法としては、サブネットにおいては、メッセージのシーケンシングだけをおこない、リアセンブルはＨＯＳＴにまかせるなどが考えられる。

パターン３に於てＩＭＰ３からＩＭＰ２に伝送される８パケットメッセージはルーティング機構により図２－１０のように分流される。１つのメッセージのみを考えると２パケットメッセージまではパターン１とパターン３は同じスループットを示す。

パターン３に於てパケット数が３、５、７の場合スループットが急に上昇するのは、図２－１０でわかる様に２つの経路が使用されることの効果である。またパケット数が８になると７の場合より下がるのは長いメッセージのラウンドトリップの影響であろう。

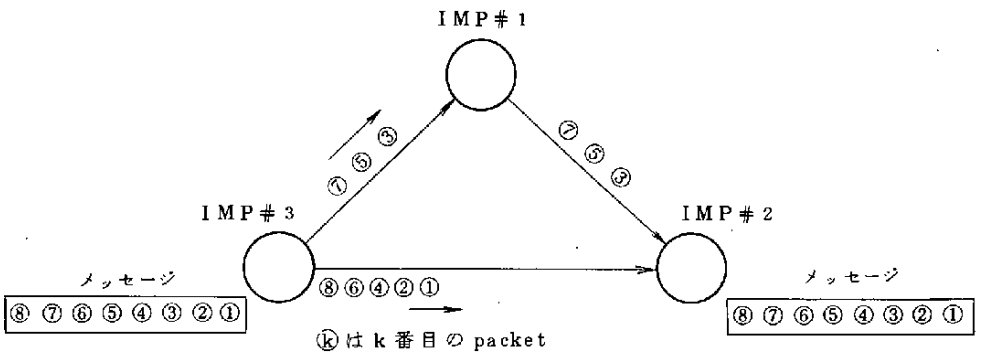


図２－１０　パケットの分流

C　ラウンドトリップ時間

１４４バイトのS－pipe メッセージの場合、１hop のラウンドトリップ時間が３５ミリ秒、２hopのそれが６６ミリ秒という測定結果である。１４４バイトメッセージを４８Kbit／sec の回線で送った場合、オーバヘッド部分も含めたメッセージの伝送遅れは、約２７ミリ秒である。メッセージが１hop経由するということは、メッセージ受信の確認であるreceive も１hop経由することになるわけであるから、メッセージ伝送にあたって１hop増加することに、約３１ミリ秒余分に時間がかかることは理解できる。また、この値は十分に満足できる値である。

測定したデータから、ＩＭＰが１パケットを蓄積交換をするに必要な時間を推定することができる。すなわち（図２－８）の２００m sec に１メッセージを送った場合のパターン１、パターン２の両グラフを比較する。そして、ＩＭＰの内部でどのような経路を通っていたかを調べることにより可能である。（図２－１１、図２－１２）はそれぞれパターン１、パターン２の処理経過を示している。ここでそれぞれの過程でかかる時間を表３－４０のようにおいてパターン１とパターン２の伝送時間の差をT(s)とすれば

$$T(s)=Tmd(s)+Tmr(s)+2\cdot Tmin+2\cdot Tgen+3\cdot Tmout$$

Tmd(s)、Tmr(s)はそれぞれ計算でててくるので、$2\cdot Tmin+2\cdot Tgen+3\cdot Tmout$ を求めることができる。

蓄積交換に必要な時間（Ｔs／ｆ）は本来ならばＴｓ／ｆ＝Tmin＋Tgen＋Tmout であるが、ヌルパケットが発信される可能性もあるのでここでは

$$Ts/f = Tmin+Tgen+1.5\,Tmout$$

とすることにした。これによればＴｓ／ｆ≒1.16ｍsecとなる。

表 2 － 1

general task 走行時間	:	Tgen msec
modem in 〃	:	Tmin 〃
modem out 〃	:	Tmout 〃
Host － IMP 〃	:	Th－i 〃
pipe control 〃	:	Tpipe 〃
message transmit delay	:	Tmd(s) 〃
receive 走行時間	:	Tmr 〃

Tmd(s)のSはmessage長を示す。

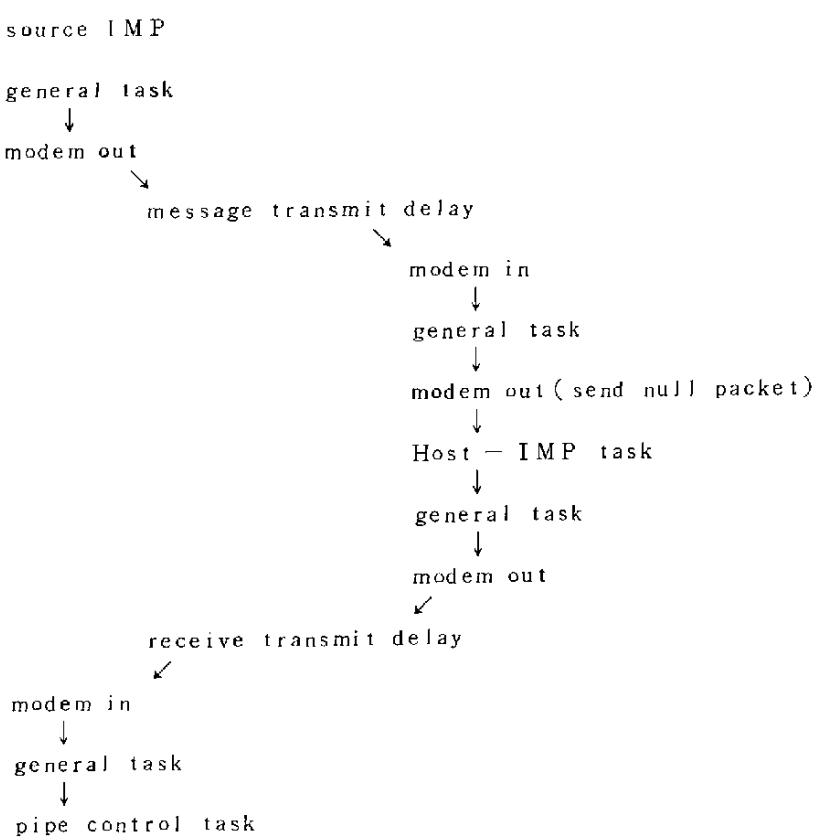


図 2－11　IMP内におけるパケット処理の流れ(1)

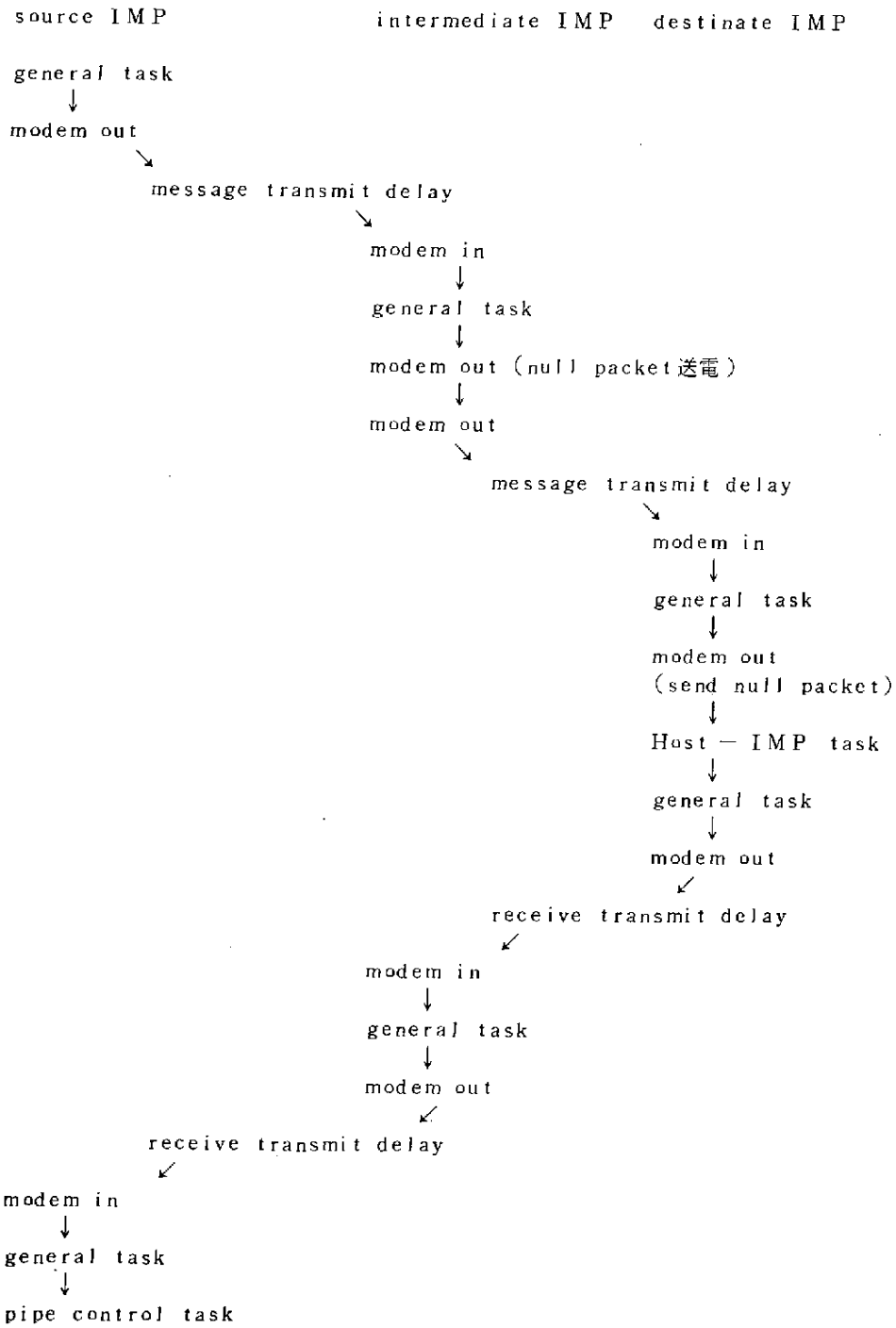


図２－１２　ＩＭＰ内におけるパケット処理の流れ(II)

D　そ　の　他

ＳＣＰ作成にあたって考慮したのは次の点である。

(1)　プログラム構造を単純化する。

(2)　機能の追加修正が簡単である。

(3)　短期間にインプリメントできる。

(4)　プログラムのモジュール化ができる。

(5)　ＨＳＣＬからの入力をタイミングよく処理できる。

この結果としてできあがったのが現在のＳＣＰである。(1)～(4)はプログラムの作成にかかる期間が約４ケ月間であり、出来得る限りシステムの早期安定をはかること、且つ論理的な誤りや不備な点を発見しやすい構造にする必要があっからである。(5)は、高速回線の性能を生かすための要求である。

ソフトウェア構造として、我々がとったようなＣＰＵディスパッチャーが各タスクにＣＰＵをあたえる方式と割り込みを起点として割り込み原因を起こしたタスクにＣＰＵを渡たし、優先度の高いタスクは、低いタスクの割り込み原因をマスクする方式が考えられる。

前者は、大形コンピュータでユーザータスクを管理するときによくもちいられている。

後者はたとえばプロセスコントロールなどのように割り込み原因とタスクの関係がはっきり決っている場合によくもちいられる手法である。

我々が前者を採用したのは、(2)の機能の追加修正がきわめて簡単だからである。もし後者を採用したならば、予定期日にできあがらなかったであろうと確信している。

現在のＳＣＰの欠点としてＣＰＵを使いすぎるのではないかということがある。

たとえばＩＭＰが毎秒２５０パケットの伝送処理を行う場合、ＣＰＵがback groundの処理を行っている時間は測定によれば全体の１６％

である。

しかし、残り７４％のうち、約半分（全体の３５％）はＩＭＰのコントロールプログラムが消費しており、夫々のタスクがＩＭＰ本来の処理を行なうために使用するＣＰＵの利用率は全体の３０～４０％である。

１個の割り込み毎にＣＰＵがback groundタスクに渡たるまでにコントロールプログラムで費される時間は現在のプログラムでは毎回385 μsee である。また１パケットを処理するために発生する割り込みは3回であり、２５０パケットを処理するには毎秒７５０回の割り込みが発生していることとなる。

これらの結果から、現在のシステムは当初の予想以上に割り込みが多発し、且つＣＰＵの有効利用率が低く、コントロールプログラム形式を採用した事は、システムの早期の安定化には極めて有効であったが、効率の点からは、割り込みオリエンテッドにＣＰＵを与え、より低い優先度の割り込みを禁止する方式が望ましいと言える。

2.3 ホスト・ホスト・プロトコル

2.3.1 測定目的と方法

A 測 定 目 的

コンピュータ・ネットワークが実用レベルで利用可能になったとき、ホスト・レベルで利用時の重要な問題と考えられるのは次の２点である。

第１はコンピュータ・ネットワークを介してメッセージを送ることが、ホストのＣＰＵをどれほど使うかということである。

第２はハードウェア環境（ホストの処理能力、サブネットの処理能力etc）に十分見合ったスループットが得られているかということである。

従って、これら２点を吟味することが、ホスト・レベルの性能評価であると考え、ＪＩＰＮＥＴの各ホスト間で次の２項目について測定をおこなった。

(1) メッセージのスループット

(2) メッセージ処理に必要なホストのＣＰＵ時間

B 測 定 方 法

スループット測定の為に、ＩＭＰ内に測定用プログラムを組み込んだ。これは最大４本までのコネクションに対して次の５項目についてミリ秒単位で測定できるものである（図２－１３参照）。

a ＲＴＴの総和：ラウンド・トリップ時間の総和。メッセージ送信側ＩＭＰがＮＣＰよりメッセージ送信要求のリーダを受け取ってから、メッセージ送信終了を意味するreceiveをＮＣＰに通知開始するまでの時間の総和。

b ＤＥＬの総和：ラウンド・トリップ時間の総和。ＲＴＴと異なる点はreceiveをＮＣＰに通知完了するまでの時間である。

c ＢＭＴの総和：receiveをＮＣＰに通知完了してから、次のメッセージ送信要求のリーダを受け取るまでの時間の総和。

d ＲＥＣの総和：受信側ＩＭＰにおいて、メッセージをＮＣＰに渡しはじめてから、receiveを受け取るまでの時間の総和。

e ＭＳＣ：送信側ＩＭＰがＮＣＰより他系へ送るよう依頼されたメッセージ数。

ホストにおけるＣＰＵ時間の測定には、ホストのジョブアカウントリストを用いた。なおアカウントリストの数値の正しさを前もって確認するため、各ホストでダミージョブを実行し、ストップウォチでその時間を計り、ジョブ終了後アカウントリストのジョブ実行時間と比較を行なった。その結果は、両者がほぼ等しかったのでアカウントリストの数値を信用することとした。

ただしＦＡＣＯＭ２３０／７５のＮＣＰはアカウンティングがとれないので（サブモニタであるため）、次のような方法でＮＣＰの使用するＣＰＵ時間の推定を行なった。

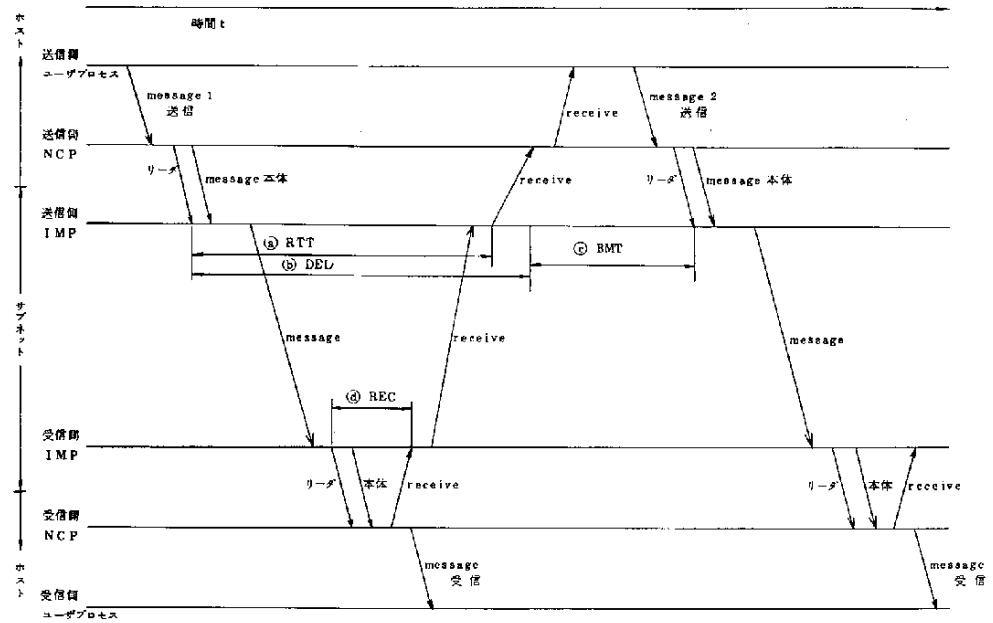


図２－１３　メッセージ伝送手順と測定可能項目図

①　ＣＰＵの利用優先権が低く、ＣＰＵだけを使うジョブを作る。－これをIdle Job と呼ぶ。

Idle Job だけを実行させて、Idle Jobがコア内に存在している時間と、実際に使用したＣＰＵ時間の関係を計測する。これによって、ＯＳに特別なサービスを要求しないときには、ＯＳのオーバヘッドがどのくらいかを知ることができる。

Idle Jobだけがコアにいるときは、Idle Jobのコア占有時間と経過時間は等しいので、

経過時間＝ＯＳのオーバヘッド＋無負荷時のIdle JobのＣＰＵ時間　　(1)

という関係が成立する。

② サブモニタ、サブモニタを使用するユーザ・ジョブ、Idle Jobを実行させる。

このとき、サブモニタのサービス開始、Idle Jobの実行、ユーザ・ジョブの実行の順にジョブを走行させる。ユーザ・ジョブの実行が終了したならば、Idle Jobの実行を終了させて、Idle JobのＣＰＵ時間を調べる。

したがって

経過時間＝ＯＳのオーバヘッド＋サブモニタのＣＰＵ＋ユーザ・ジョブのＣＰＵ＋Idle JobのＣＰＵ　　(2)

となる。

ユーザ・ジョブのＣＰＵ、Idle JobのＣＰＵは、ジョブのアカウンティングから、ＯＳのオーバヘッドは、式(1)より推定できるので、サブモニタのＣＰＵは計算できることになる（図２－１４、図２－１５参照）。

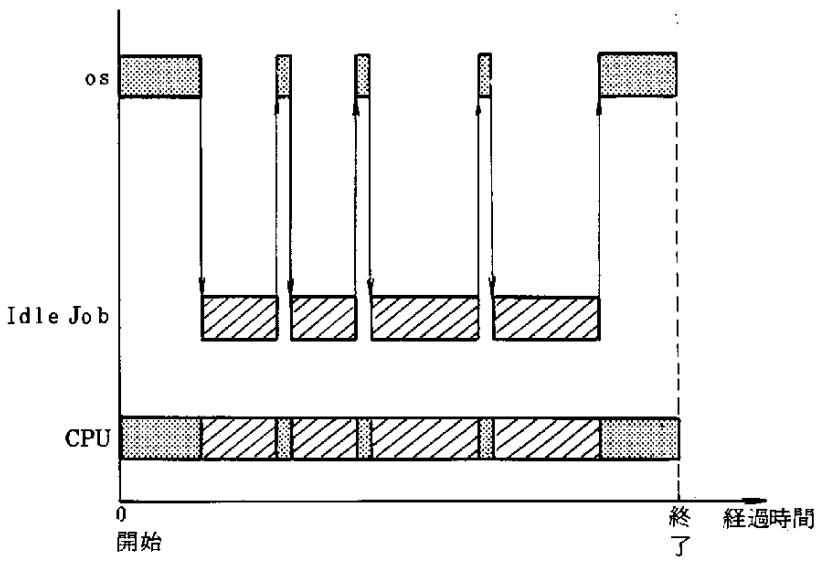


図２－１４　ＯＳのオーバヘッド

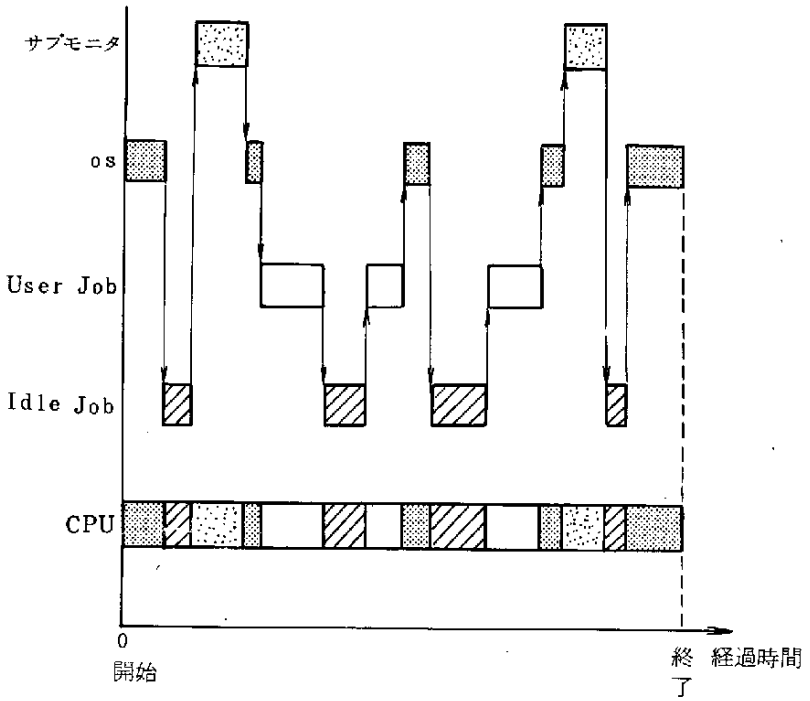
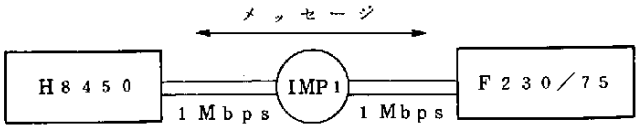


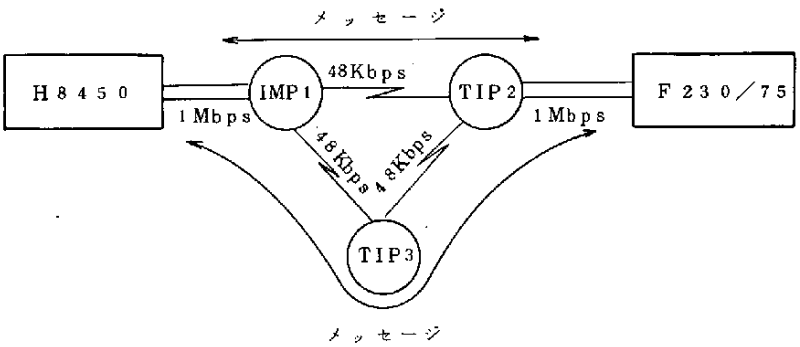
図２－１５　サブモニタのＣＰＵ時間

測定は、次のようにして行なった。指定した長さのメッセージ（Ｓ－pipe メッセージとして１４４バイト、Ｌ－pipe メッセージとして１４４～１１５２バイト）を１０００個ネットワークを介して出力あるいは入力するプログラムをユーザジョブとして実行し、ジョブが終了したのちアカウントリスト、前述のＩＭＰ内の測定用プログラムの統計データによって解析をした。

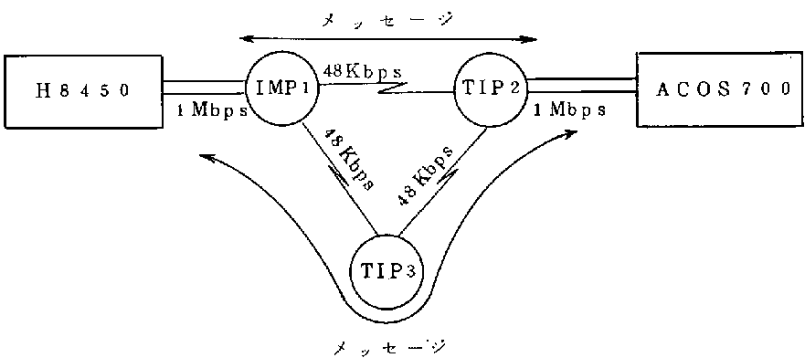
測定パターンは図２－１６に示すとおりであるが、パターン２はパターン１と比較の目的（前者は回線経由、後者は同一ＩＭＰ結合）で、ＦをＡのかわりにつけ変えたものである。



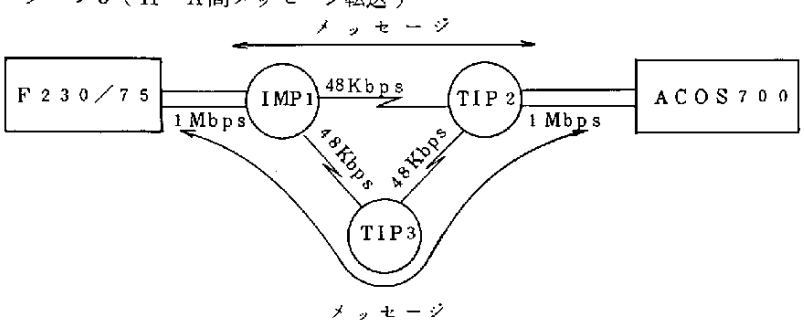
パターン1（H－F間メッセージ転送「同一IMP結合」）



パターン2（H－F間メッセージ転送「回線経由」）



パターン3（H－A間メッセージ転送）



パターン4（F－A間メッセージ転送）

図2－16　ホストレベル測定パターン

2.3.2 測 定 結 果

前述のパターン１～４までの測定結果を表２－２～表２－４に示す。パターン１・パターン２の測定結果は両者を比較しやすいように表２－２として１つにまとめてある。

表の各項目の説明を次に行なう。

(1) 動 作 時 間

測定に要したelapse時間であり、その算定方法は次のとおりである。

ＤＥＬ＋ＢＭＴ の総和 ⟹ 動作時間 の総和

ＤＥＬの総和、ＢＭＴの総和はＩＭＰに組み込んだ測定用プログラムで測った各ＤＥＬ、ＢＭＴの１測定間の総和である。

(2) スループット

各測定で得られたメッセージの伝送スループットであり、その算定方法は次のとおりである。

$$\frac{\text{1メッセージのビット数} \times 1{,}000(\text{メッセージ})}{\text{動作時間} \times 1{,}000} \Longrightarrow \text{スループット (Kbits/sec)}$$

(3) ＣＰＵ時間

ＣＰＵ時間は各ホスト使用ＣＰＵ時間をアカウントリストより導いたものであり、その算定方法は次のとおりである。

ＮＣＰの CPU時間 ＋ ユーザプロセスの ＣＰＵ時間 ⟹ ＣＰＵ時間

なおＨはアカウントリスト上では秒単位でしか表示されないので、その有効桁数は他のホストに比べて低い。またＦのＮＣＰのＣＰＵ時間は前述の方法で算定したものである。

％表示はメッセージ送受が各ホストに与えるＣＰＵ負荷を％で表したものであり、その算定方法は次のとおりである。

$$\frac{\text{CPU時間}}{\text{動作時間}} \times 100 \Longrightarrow \%\text{表示}$$

後述の表２－２のF→H・同一IMP・１４４バイト（S－pipe）のHのCPU時間のように%表示が９５.６ということは、Hの全使用可能CPU時間のうち実に９５.６%がメッセージ受信動作につかわれたことを意味する。

表２－２　パターン１、２（F－H間、１０００メッセージ）の結果

項目		動作時間 (sec)	スループット Kbits／sec	FのCPU時間 sec／%	HのCPU時間 sec／%
F → H １４４バイト (S-pipe)	同一IMP	25.1	45.8	16.5 s／65.7%	24 s／95.6%
	回線経由	61.6	18.7	16.5 s／26.8%	23 s／37.3%
F → H １４４バイト (L-pipe)	同一IMP	30.3	38.1	22.5 s／74.3%	23 s／75.9%
	回線経由	81.4	14.1	22.7 s／27.9%	23 s／28.3%
F → H ５７６バイト (L-pipe)	同一IMP	38.2	120.5	22.7 s／59.4%	23 s／60.2%
	回線経由	146.8	31.4	22.7 s／15.5%	23 s／15.7%
F → H １１５２バイト (L-pipe)	同一IMP	48.9	188.4	23.0 s／47.0%	24 s／49.1%
	回線経由	216.1	42.6	23.0 s／10.6%	24 s／11.1%
H → F １４４バイト (S-pipe)	同一IMP	27.5	41.9	15.8 s／57.5%	17 s／61.8%
	回線経由	61.9	18.6	15.7 s／25.4%	16 s／25.8%
H → F １４４バイト (L-pipe)	同一IMP	33.3	34.6	16.2 s／48.6%	25 s／75.0%
	回線経由	82.1	14.0	16.0 s／19.5%	23 s／28.0%

H → F 576バイト (L-pipe)	同一IMP	41.2	111.8	16.1 s 39.1 %	24 s 58.3 %
	回線経由	143.2	32.2	16.1 s 11.2 %	23 s 16.1 %
H → F 1152バイト (L-pipe)	同一IMP	52.2	176.7	16.5 s 31.7 %	26 s 49.8 %
	回線経由	208.1	44.3	16.5 s 7.9 %	24 s 11.5 %

表2－3　パターン3（H－A、間1,000メッセージ）の結果

項　　目	動作時間 (sec)	スループット Kbits/sec	AのCPU時間 sec %	HのCPU時間 sec %
A → H 144バイト(S-pipe)	83.5	13.8	20.8 s 24.9 %	23 s 27.5 %
A → H 144バイト(L-pipe)	116.8	9.9	25.5 s 21.8 %	22 s 18.8 %
A → H 576バイト(L-pipe)	183.5	25.1	31.9 s 17.4 %	23 s 12.5 %
A → H 1152バイト(L-pipe)	252.6	36.5	40.5 s 16.0 %	24 s 9.5 %
H → A 144バイト(S-pipe)	68.7	16.8	15.1 s 22.0 %	17 s 24.7 %
H → A 144バイト(L-pipe)	88.2	13.1	14.3 s 16.2 %	23 s 26.1 %
H → A 576バイト(L-pipe)	158.5	29.1	20.8 s 13.1 %	23 s 14.5 %
H → A 1152バイト(L-pipe)	234.8	39.3	29.3 s 12.5 %	24 s 10.2 %

表２－４　パターン４（Ｆ－Ａ間、１,０００メッセージ）の結果

項　　目	動作時間 (sec)	スループット Kbits／sec	ＡのＣＰＵ時間 sec / %	ＦのＣＰＵ時間 sec / %
Ａ → Ｆ １４４バイト（S-pipe）	83.6	13.8	20.8 s / 24.9 %	16.0 s / 19.0 %
Ａ → Ｆ １４４バイト（L-pipe）	116.9	9.9	25.5 s / 21.8 %	15.9 s / 13.6 %
Ａ → Ｆ ５７６バイト（L-pipe）	183.6	25.1	32.0 s / 17.4 %	16.3 s / 8.9 %
Ａ → Ｆ １１５２バイト（L-pipe）	255.5	36.1	40.5 s / 15.9 %	16.3 s / 6.4 %
Ｆ → Ａ １４４バイト（S-pipe）	68.6	16.8	14.3 s / 20.9 %	16.6 s / 24.2 %
Ｆ → Ａ １４４バイト（L-pipe）	88.2	13.6	14.5 s / 16.4 %	22.6 s / 25.6 %
Ｆ → Ａ ５７６バイト（L-pipe）	163.5	28.2	21.0 s / 12.8 %	22.7 s / 13.9 %
Ｆ → Ａ １１５２バイト（L-pipe）	244.2	37.7	29.4 s / 12.0 %	23.1 s / 9.5 %

2.3.3　検討および評価

ＪＩＰＮＥＴのＳ－pipe メッセージとＬ－pipe メッセージとではホスト・ＩＭＰ間でもＩＭＰ・ＩＭＰ間（サブネット）においてもメッセージの伝送制御手順が異なるので別々に論ずる。

Ａ　Ｓ－pipe メッセージ

検討に入る前にＳ－pipe メッセージの送信手順を図２－１７によって説明しておく。

①　他系へメッセージを送りたい発信ＮＣＰはＷＤＩ（Write Data Interrupt）コマンドでリーダー（制御情報）をＩＭＰへ送る。

②　①に続いてＷＤ（Write Data）コマンドでメッセージ本体をＩ

ＭＰへ送る。

③　ＩＭＰはリーダーとメッセージ本体からメッセージパケットを作り、他系ＩＭＰへ送る。

④　メッセージパケットを受信したＩＭＰは、ＷＤＩコマンドでまずリーダーを送る。

⑤　④に続いてＷＤコマンドでメッセージ本体を送る。

⑥　メッセージを受け取ったＮＣＰはＷＤＩコマンドで receive をＩＭＰへ送る。

⑦　ＩＭＰは receive をパケット化して発信側ＩＭＰへ送る。

⑧　receive パケットを受け取ったＩＭＰはＷＤＩコマンドで receiveを発信ＮＣＰへ通知する。これにより発信ＮＣＰはメッセージが受信ＮＣＰに着いたことを知る。

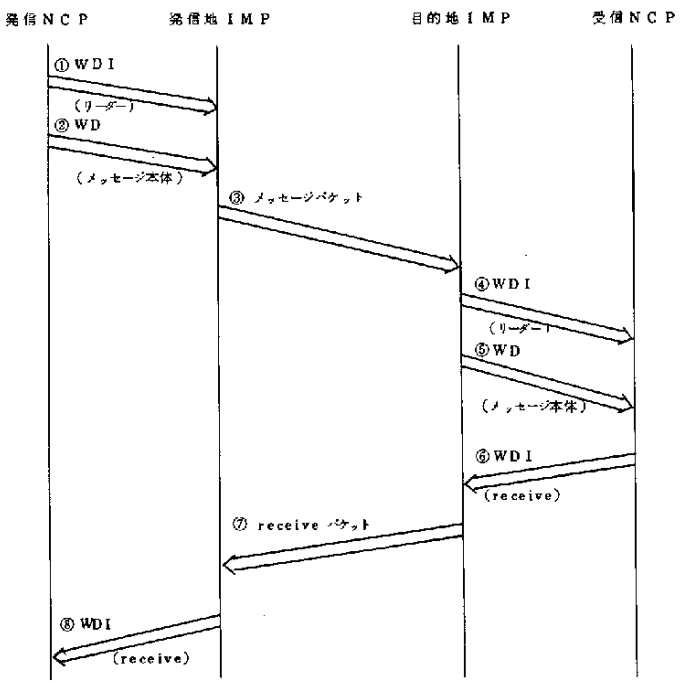


図２－１７　S-pipe メッセージの送信手順

(1)　ＣＰＵ時間の検討

前記の測定結果より、３台のホストがメッセージを送受信するのに使用するＣＰＵ時間を抜き出し、棒グラフにすると図２－１８になる。

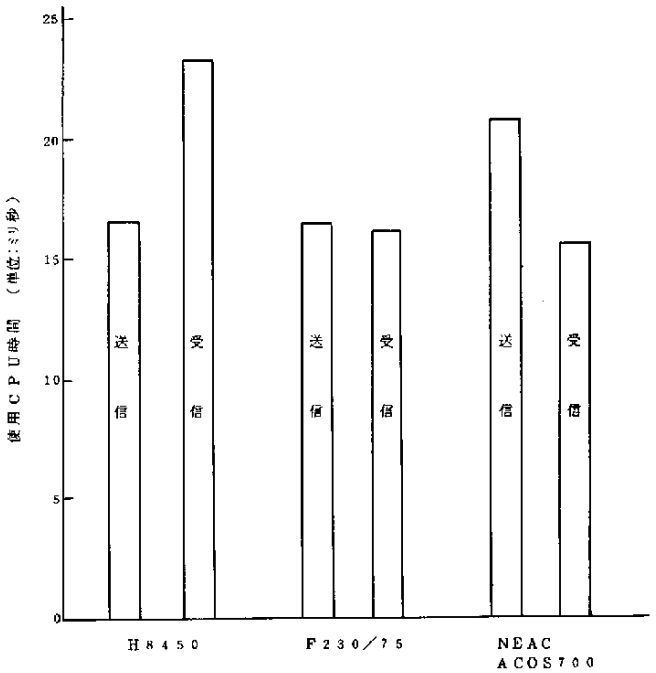


図２－１８　１４４バイトS-pipe１メッセージ当り使用ＣＰＵ時間

図によるとHの場合、送信に１７ミリ秒、受信に２３ミリ秒のＣＰＵ時間を使用しており、受信の方がＣＰＵを多く使用している。この原因は、HではUser ProcessとＮＣＰ間の通信にはＩＰＣ（Inter Process Communication）とＤＩＲＥＣＴ　ＣＯＰＹ（相手領域への直接書き込み、あるいは直接読み出し）を併用しているが、受信の場合の方が１回ＩＰＣが多いことによる（図２－１９、図２－２０参照）。

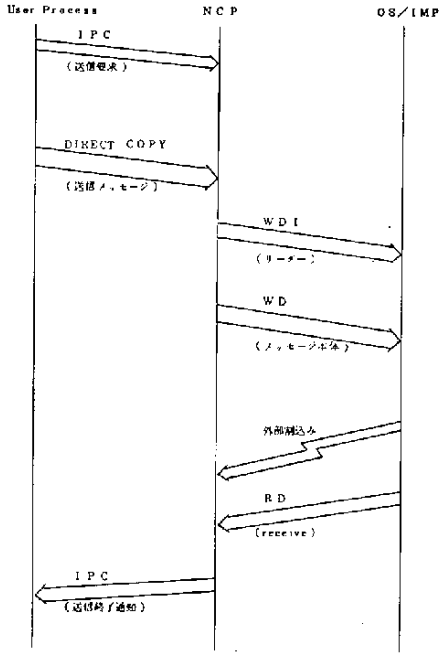


図２－１９　Ｈ８４５０のメッセージ送信手順

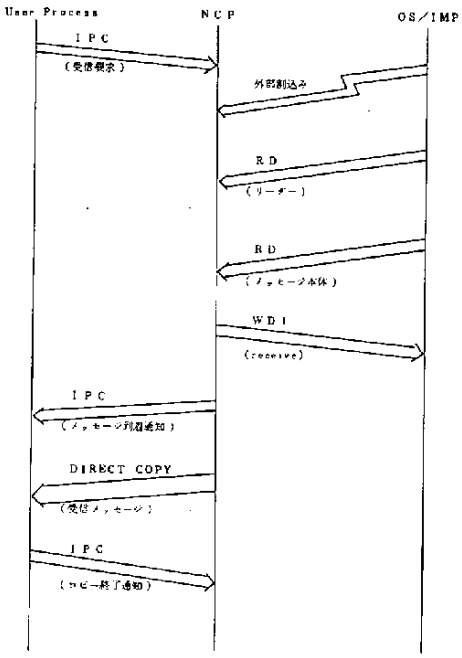


図２－２０　Ｈ８４５０のメッセージ受信手順

Aの場合、Hと逆になるのは、AではＮＣＰがＩＭＰからのメッセージ送信要求割込み（ＩＭＰよりのＷＤＩコマンドによって発生）を受け付ける為、前もって割込みがはいると実行されるＲＤ（Read Data）コマンドを発信しているが、送信のときにはこのコマンドをキャンセルしてのち出力コマンド（ＷＤＩあるいはＷＤ）を発信しなければならないからである。なおUser ProcessとＮＣＰ間の通信手順で送信の場合の方がＤＩＲＥＣＴ　ＣＯＰＹが１回多いが、これはＣＰＵをほとんど使わないことを確認している（図２－２１、図２－２２参照）。

測定結果より、メッセージの送受信には１０数ミリ秒から２０数ミリ秒ものＣＰＵ時間を必要とすることがわかる。またＨ・Ａ・ＦはそのＣＰＵ速度にかなり差がある（約１：３：９）にもかかわらず、メッセージ送受に要するＣＰＵ時間に、それほど差がないのは興味深いことである。

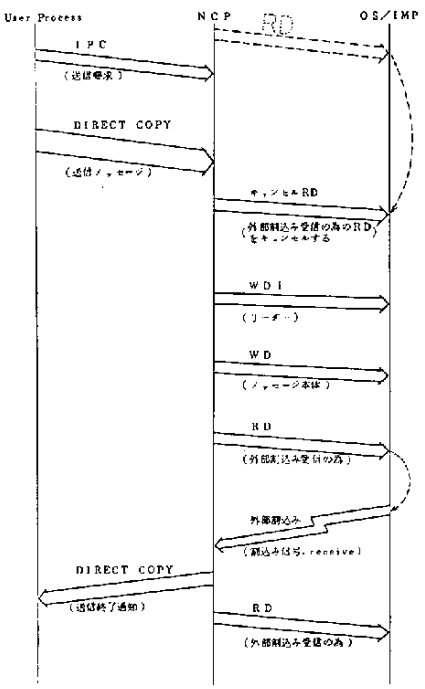


図２－２１　ＡＣＯＳシステム７００のメッセージ送信手順

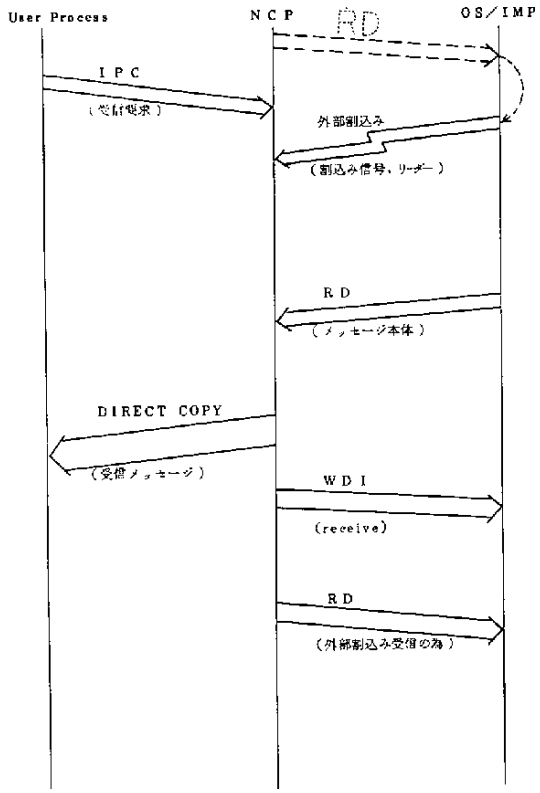


図２－２２　ＡＣＯＳシステム７００のメッセージ受信手順

(2)　スループットの検討

表２－２によると、同一ＩＭＰ結合のＦ・Ｈ間伝送では、明らかにホストのＣＰＵバウンドでスループットがおさえられている（Ｈは使用可能ＣＰＵ時間のうち最高９５.６％も使っている場合がある）。このことはホスト・ＩＭＰ間の高速転送能力（最大１Ｍ bits／sec）が十分に生かされていないことを意味する。

回線経由の場合、ＩＭＰ内測定では５０Kbits／sec以上のスループットがでているにもかかわらず、１４～１９Kbits／sec位しかでていない。このことはホストのＣＰＵ使用時間が大きいことと、メッセ

ージ送信手順として、送信側プロセスが前のメッセージのreceive を受け取ってから次のメッセージを送るという方法をとるので、サブネットに空きの時間が生じるからである。

またF・HではＮＣＰに優先的にＣＰＵが与えられるようＯＳがスケジューリングしているが、Ａではそうなっていない。したがってＮＣＰの応答動作が遅くなり、Ａの場合にはこれもスループット低下の原因となっている。

B　L－pipeメッセージ

検討に入る前にL－pipeメッセージの送信手順を図２－２３によって説明しておく。

① 他系へメッセージを送りたい発信ＮＣＰはＷＤＩでリーダーをＩＭＰへ送る。

② ＩＭＰは受信側ＩＭＰに対して、メッセージ受信の為のバッファアロケートを要求するＲＦＡパケットを送る。

③ 受信側のＩＭＰはバッファアロケートが終了すると発信側へＡＬＬＯＣパケットを送る。

④ 発信側ＩＭＰは発信ＮＣＰへＷＤＩコマンドで、メッセージ本体送信可を意味するＯＫを送る。

⑤ ＯＫを受け取った発信ＮＣＰはメッセージ本体を１４４オクテットのセグメントに区切って、各セグメントをＷＤコマンドで必要回数だけ連続にＩＭＰへ送る。

⑥ ＩＭＰは各セグメントをパケット化して受信側ＩＭＰへ送る。

⑦ 1メッセージを構成する全パケットを受け取ったＩＭＰは、まずＷＤＩコマンドでリーダーを受信ＮＣＰへ送る。

⑧ ⑦に続いてＷＤコマンドで全セグメントを連続に受信ＮＣＰへ送る。

⑨ メッセージを受け取ったＮＣＰはＷＤＩコマンドでreceive をＩＭＰへ送る。

⑩　ＩＭＰはreceive をパケット化して発信側ＩＭＰへ送る。

⑪　receive パケットを受け取ったＩＭＰはＷＤＩコマンドでreceiveを発信ＮＣＰへ通知する。これにより発信ＮＣＰはメッセージが受信ＮＣＰに着いたことを知る。

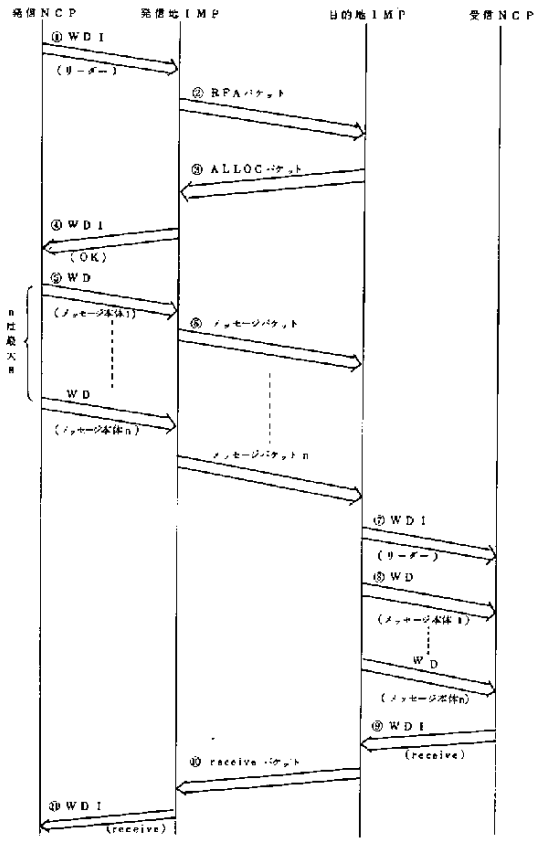


図２－２３　Ｌ-pipeメッセージの送信手順

(1)　ＣＰＵ時間の検討

前記の測定結果より、３台のホストがメッセージを送受するのに使用するＣＰＵ時間を抜き出しグラフにすると図２－２４になる。

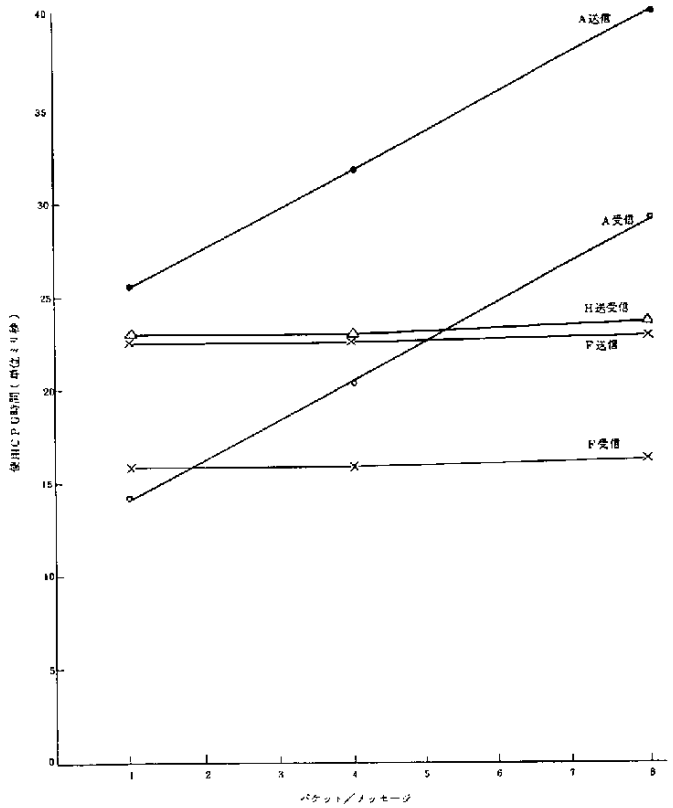


図２－２４　L-pipeメッセージの使用ＣＰＵ時間
（１パケット：１４４バイト）

図によると、Ａ・Ｆは受信よりも送信の方がＣＰＵ時間を多く使用している。この理由は前述のＬ-pipe メッセージの送信手順からわかるとおり、送信の方がＮＣＰとＩＭＰ間の会話手順が１回多いことと、Ａの場合にはＳ-pipe のところで説明したとおり、割り込み待ちのＲＤをキャンセルしてから出力コマンドを出すという手順をとることも原因である（図２－２３参照）。

Ｈの場合は送受信の使用ＣＰＵ時間がほとんど同じである。この理由は、ＮＣＰ・ＩＭＰ間では送信の方が１回会話手順が多いが、User Process ・ＮＣＰ間ではＳ-pipe のところで説明したとお

り、送信の方が1回会話手順が少なく互が相殺された結果である（図2－19参照）。

また、F・Hの場合は使用CPU時間はパケット数に対して一定であるのに、Aの場合はパケット数に比例している。この理由は、F・Hが前述のL-pipe メッセージの送信手順の⑤および⑧のところでコマンドチェーンを使用しているのに対し、Aはコマンドチェーンが使用できないのでnパケットメッセージをIMPと送受する為にn回入出力コマンドを発信しなければならないことによる。

(2) スループットの検討

表2－2にみられるとおり、同一IMP結合の高速伝送を行なったときにはS-pipe の場合に比べて確かに高スループットは得られている。しかし、そのために使用するCPU時間はFで32%～74%Hでは49%～76%にもなっている。また144バイト伝送ではS-pipe の方が高スループットが得られている。これはL-pipeの方が伝送手順が複雑な為と考えられる。

回線経由のスループット測定結果を図示したものが図2－25である。なおA・F間は図に示されていないが、これはA・H間とほとんど同様なので省いた。またF・H間は各パケット長ごとの測定を行ない図示してある。図から明らかなように、L-pipe もS-pipe と同様サブネットのL-pipe 機能を十分に生かせずスループットの大幅な減少という結果になっている。この理由はS-pipe と同じくホストの使用CPU時間が大きいことと、前のメッセージのreceiveを待ってから次のメッセージを送るという手順をとる為、サブネットに空き時間が生じるとともに、サブネットのL-pipe 用の高スループットメカニズムが有効に働かないためである。

図をみるとH→Fのスループットパターンは IMP内測定のスループットパターンと同型であるが、これは両者の測定時のメッセージ伝

送方向が同じ為、サブネット内で同一パターンのフロー制御を受ける為と考えられる。

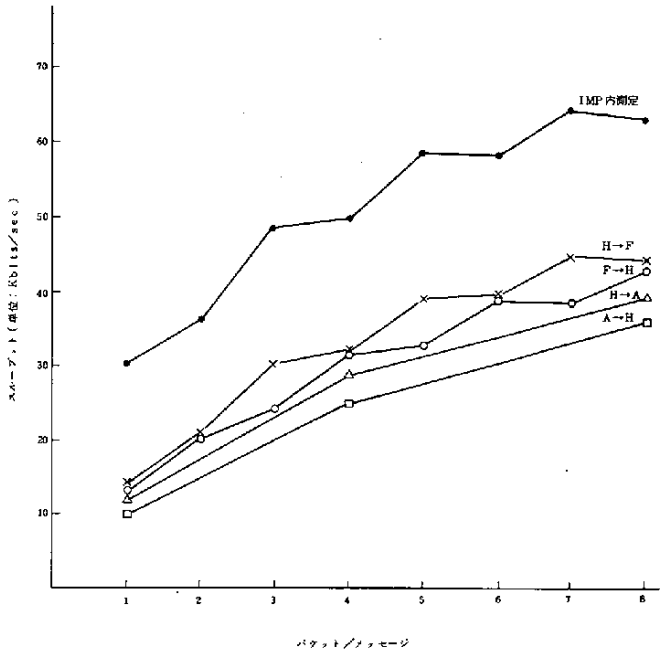


図２－２５　回線経由のL-pipe メッセージにおけるスループット対メッセージ長

C　結　　論

コンピュータ・ネットワークにおいて、ＮＣＰがメッセージ処理に使用するＣＰＵ時間は、今まであまり注目されてはいなかったが、実際には予想外に大きく、これを何らかの方法によって小さくする必要がある。ＪＩＰＮＥＴにおける測定は、ＮＣＰとＩＭＰおよびユーザプログラムとの相互交信の回数がＣＰＵ使用時間に大きな影響をあたえることを示している。このことは、送受信するメッセージ長あるいはパケット長を大きくすることによってスループットの向上、ＣＰＵ時間を減少させることが可能なことを意味する。

しかしながら、根本的にはＮＣＰのハードウェア化、あるいはホストコンピュータ外にＮＣＰを置くなどの方法を検討していかなければならないであろう。

2.4　高位のプロトコル

2.4.1　ファイル転送プロトコル

A　測定目的と方法

ファイル転送プロトコル（File Transfer Protocol ：FTP）は、コンピュータ・ネットワーク内のホスト間でファイルを効率よく転送するための高位プロトコルである。ＪＩＰＮＥＴのＦＴＰもファイルを効率よく転送する意図で設計され、今年度このＦＴＰをＦＡＣＯＭ２３０／７５とＨＩＴＡＣ８４５０の各ホストにＦＴＰプログラムとして実現した。

最初にＦＴＰプログラムの動作概要を図２－２６のモデルに従って簡単に述べる。

① ユーザ（カードリーダ、端末）からの開始指令により、ユーザＦＴＰのユーザＰＩはサーバＦＴＰとの間にＰＩコネクションを確立する。

ＰＩコネクションは、ユーザＦＴＰからはＦＴＰコマンド（ユーザからのＦＴＰサービスの開始・終了そしてファイル転送の開始等をサーバＦＴＰに通知するコマンド）、サーバＦＴＰからはその応答のために使用される両方向のコネクションである。

② ファイル転送の開始指令がユーザより投入されたならば、ユーザＦＴＰはファイル転送用ＦＴＰコマンドをサーバＦＴＰに転送する。その後ユーザＦＴＰのユーザＤＴＰはサーバＦＴＰとの間にＤＴＰコネクションを確立しファイルを転送する。

ＤＴＰコネクションは、ＦＴＰコマンドが“ＳＴＯＲＥ”の場合にはユーザＦＴＰからサーバＦＴＰへファイルを転送するために、またＦＴＰコマンドが“ＲＥＴＲＩＥＶＥ”の場合にはサーバＦＴＰから

ユーザＦＴＰへファイルを転送するために必要な片方向のコネクションである。このコネクションはファイルの転送が終了すると自動的に切断される。

③　ユーザからの終了指令によりＰＩコネクションは切断され、ＦＴＰサービスは終了する。

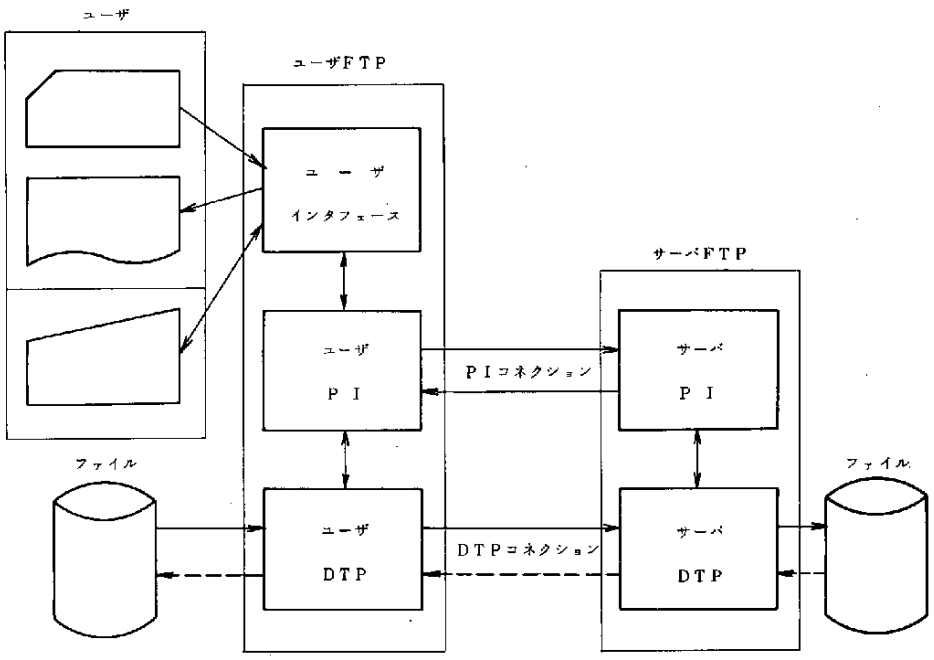


図２－２６　ＦＴＰプログラムのモデル

上記のように動作するＦＴＰプログラムを使用して、２つのホスト間（ＦＡＣＯＭ：Ｆ、ＨＩＴＡＣ：Ｈ）で実際にファイルの転送をおこない、次の項目について測定した。

①　ＤＴＰコネクション上を流れるデータのスループット

②　ファイル転送に必要な各ＦＴＰプログラムの使用ＣＰＵ時間

以上の測定データと前項のホスト・ホストプロトコルの測定データ

を比較検討することにより、ＪＩＰＮＥＴのＦＴＰならびにＦＴＰプログラムを評価し、どのような要因によってこのような性能になったかを明確にする。

測定データの採集方法は、前項のホスト・ホストプロトコルと同様に、測定終了後の各ＦＴＰプログラム・ＮＣＰのアカウントリストを取ることと、測定開始から終了までにＩＭＰ内を流れたメッセージの伝送時間と個数等をＩＭＰ内で計測することである。

測定パターンは次の項目の組み合せである。

① ネットワーク構成

構成としては、図２－２７に示す同一ＩＭＰに２つのホストが結合するコンピュータ・コンプレックス指向の構成（ohop）と、図２－28に示す回線を経由して２つのホストが結合するコンピュータ・ネットワーク指向の構成（full）の２種である。

しかし比較検討の対象としては、コンピュータ・ネットワークの普通の構成と考えられる回線経由の測定結果を用いた。

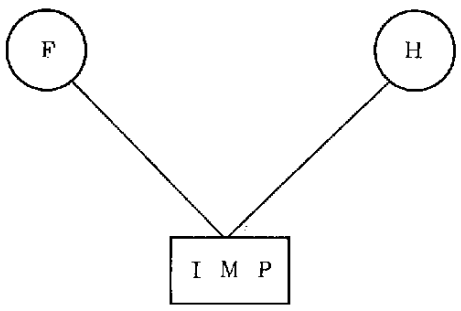


図２－２７　同一ＩＭＰ結合（ohop）

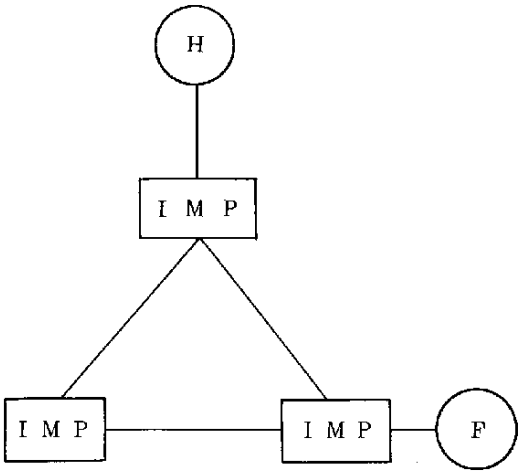


図２－２８　回線経由結合（ｆｕｌｌ）

②　転送レコード長

２０４８バイト ── ＦＴＰで規定される最大レコード長

１２００バイト

③　ファイルの容量

１０００レコード

④　ファイルの属性

ソースプログラム ── コードはＥＢＣＤＩＣまたはＪＩＳである。

データ

⑤　転送方向

ＦＡＣＯＭからＨＩＴＡＣへ（Ｆ → Ｈ）
- Ｆ：サーバＦＴＰ　Ｈ：ユーザＦＴＰ
- Ｆ：ユーザＦＴＰ　Ｈ：サーバＦＴＰ

ＨＩＴＡＣからＦＡＣＯＭへ（Ｈ → Ｆ）
- Ｆ：サーバＦＴＰ　Ｈ：ユーザＦＴＰ
- Ｆ：ユーザＦＴＰ　Ｈ：サーバＦＴＰ

B 測 定 結 果

表２－５　データ型ファイル（２０４８バイト×１０００）の転送

転送方向	構成	送信側の情報			受信側の情報	スループット（Kbit／秒）	備考
		CPU時間（秒）	伝送時間／レコード（m秒）	受信したALLOC数	CPU時間（秒）		
H→F	0 hop	77.0	164.2	6	35.9	99.8	F：サーバ
	full	74.0	485.7	4	35.9	33.7	
F→H	0 hop	53.6	183.0	1009	109.0	89.5	H：ユーザ
	full	47.6	426.4	0	102.0	38.4	
H→F	0 hop	94.0	178.2	0	37.1	92.0	F：ユーザ
	full	92.0	492.7	0	37.1	33.3	
F→H	0 hop	86.2	157.3	2665	126.0	104.1	H：サーバ
	full	69.3	471.8	0	101.0	34.7	

表２－６　データ型ファイル（１２００バイト×１０００）の転送

転送方向	構成	送信側の情報			受信側の情報	スループット（Kbit／秒）	備考
		CPU時間（秒）	伝送時間／レコード（m秒）	受信したALLOC数	CPU時間（秒）		
H→F	0 hop	67.0	132.8	2	34.7	72.3	F：サーバ
	full	65.0	331.3	4	35.7	29.0	
F→H	0 hop	53.4	144.8	1012	87.0	66.3	H：ユーザ
	full	47.4	313.1	0	78.0	30.7	
H→F	0 hop	71.0	130.9	0	37.0	73.3	F：ユーザ
	full	70.0	326.0	0	36.0	29.4	
F→H	0 hop	59.7	106.3	1927	85.0	90.3	H：サーバ
	full	47.7	298.6	0	68.0	32.1	

表２－７　ソース型ファイル（１２００バイト×１０００　　コード：EBCDIC）の転送

転送方向	構成	送信側の情報			受信側の情報	スループット（Kbit/秒）	備考
		CPU時間（秒）	伝送時間/レコード（m秒）	受信したALLOC数	CPU時間（秒）		
H→F	0 hop	58.0	161.6	2	23.7	59.4	F：サーバ
	full	57.0	325.2	2	23.7	29.5	
F→H	0 hop	53.4	680.5	1010	97.0	14.1	H：ユーザ
	full	53.4	673.6	999	96.0	14.3	
H→F	0 hop	59.0	161.7	0	25.9	59.4	F：ユーザ
	full	56.0	378.9	0	25.9	25.3	
F→H	0 hop	59.7	649.9	1998	114.0	14.8	H：サーバ
	full	58.7	642.4	1751	109.0	14.9	

表２－８　ソース型ファイル（１２００バイト×１０００　　コード：JIS）の転送

転送方向	構成	送信側の情報		受信側の情報	スループット（Kbit/秒）	備考
		CPU時間（秒）	伝送時間/レコード（m秒）	CPU時間（秒）		
H → F	0 hop	64.0	169.1	25.9	56.8	F：ユーザ
F → H	0 hop	59.7	652.3	119.0	14.7	H：サーバ

CPU時間：NCPの使用したCPU時間＋FTPプログラムの使用したCPU時間

伝送時間/レコード：1レコードを他系に転送し、その応答が返るまでの時間

受信したALLOC数：受信側から受けたALLOCコマンドの数

スループット：総転送ビット数/総転送時間

C　検討および評価

B.で示した測定結果と前項のホスト・ホストプロトコルの測定データを比較検討する前に、両者で使用したプログラムの構造を述べ相違点を明確にする。

前項で使用したプログラムは、図２－２９に示すように、指定した長さのメッセージ（１４４バイト～１１５２バイト）を１０００個送信あるいは受信するだけのプログラムである。一方ＦＴＰプログラムは、図２－３０に示すように、ディスク上のファイルをレコード単位で読み込み、それをホスト・ホストプロトコルに従ってメッセージとして送信する。あるいは受信したメッセージをレコードに組み立て、ディスク上にファイルとして書き出すプログラムである。さらにＦＴＰプログラムは、コンピュータ・ネットワークに加入するホスト間でファイルを効率よくしかも確実に転送するための手段を全ネットワーク・ユーザに与え、ネットワーク・ユーザがファイルを共有することを可能にするユーティリティ・プログラムであるため、次頁に示す規約（プロトコル）をサポートしなければならない。

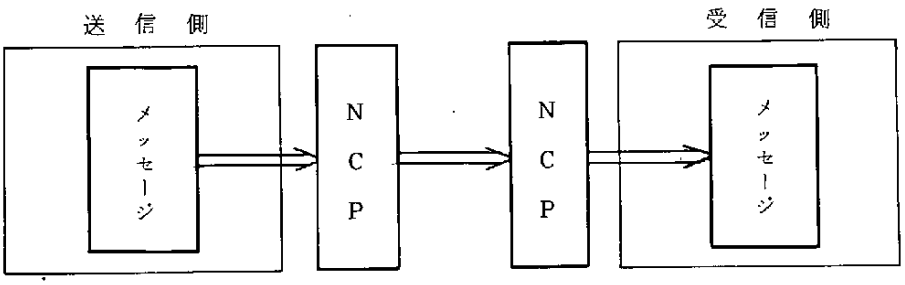


図２－２９　ホストレベルの転送

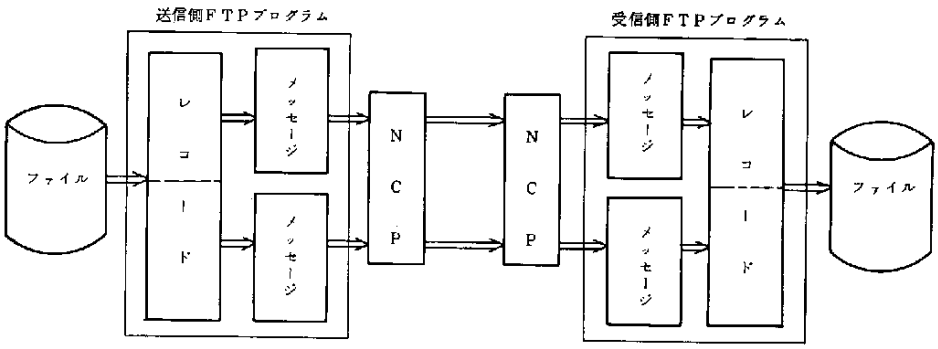


図２－３０　ＦＴＰレベルの転送

①　データの転送方式

ファイルの送信側は、ディスクより読み込んだレコード（ＦＴＰレコード一最大長は２０４８バイトである）をブロックに分割し、このブロックにヘッダを付加しメッセージとして相手側へ送信する。受信側は受け取ったメッセージからヘッダを削除しブロックとする。この時ヘッダ内のDescriptor がＥＯＲ（レコードの終わり）であるならば、今まで受信したブロックをまとめて１つのレコードとしファイルへ書き出す。

②　データの表現形式

転送時のデータ形式には、ビットレベルのトランスペアレンシを保障するバイナリ型とコード系としてＪＩＳかＥＢＣＤＩＣである文字型がある。文字型は図２－３１で示すコード変換が必要になる場合もある。

③　サーバＦＴＰのファイル管理

ファイルの属性には、データ型ファイルとソース型ファイルがある。データ型ファイルは各サーバＦＴＰが属するホストで許される形式で蓄積・管理されるが、ソース型ファイルはライブラリ形式で蓄積・管理される。

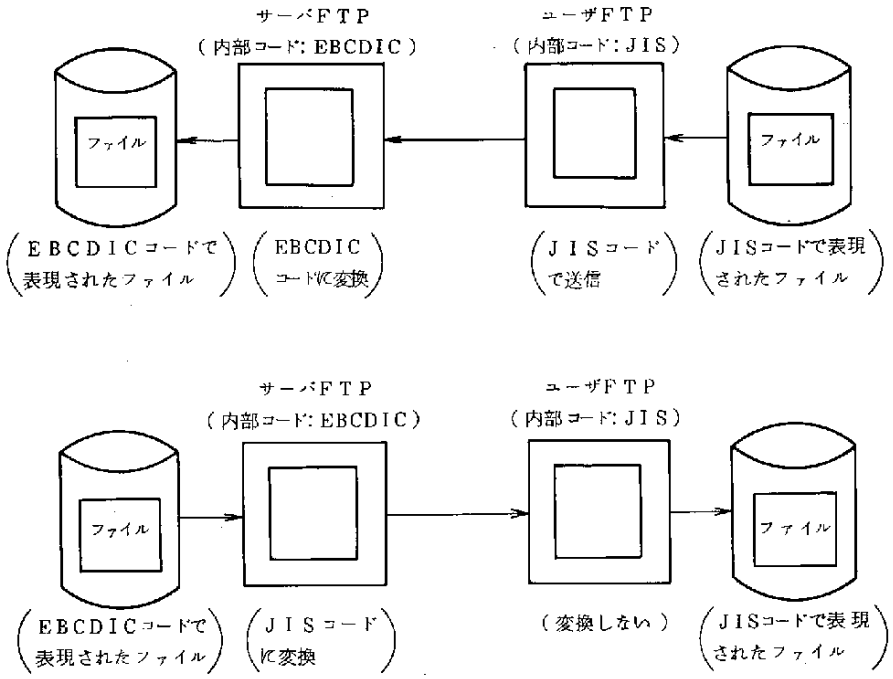


図２－３１　コード変換

④　そ　の　他

コネクション（ＰＩ、ＤＴＰ）の確立・切断とＦＴＰコマンドについては、コンピュータ・ネットワークＪＩＰＮＥＴの研究開発（５０－Ｓ００３）を参照。

以上のことより、ＦＴＰレベルの転送が効率に与える影響としてディスクＩ／Ｏとファイル転送に関係するＦＴＰ独自の処理があげられる。それゆえ、これらについて検討をおこなっていく。

第１番目として表２－５の測定結果を検討する。これはディスクＩ／ＯとＦＴＰで扱うレコード（ＦＴＰレコード）の最大長を２０４８バイトにした影響を調べるものである。

ＦＴＰ独自の処理として次のものがある。

①　ホスト・ホスト間で授受されるメッセージの最大長はホスト・ホストプロトコルにより１１５２バイトと規定されている。そのため１１５２

バイトを超えるレコードをＦＴＰプログラム間で転送する場合には、そのレコードを１１５２バイト以内のブロックに分割しなければならない。

② 分割したブロックにデータの長さそしてレコードの区切り等を示す情報（ヘッダ）を付加する。

表２－５の結果と２０４８バイトをホストレベルで転送した場合のスループット、使用ＣＰＵ時間等を前項のホスト・ホストプロトコルの測定結果より求め比較したのが表２－９である。

表２－９　ＦＴＰレベルとホストレベルの比較

レベル＼	転送方向	構成	FのＣＰＵ時間（秒）	HのＣＰＵ時間（秒）	伝送時間／レコード（ｍ秒）	スループット（Kbit／秒）
ＦＴＰ	F→H	full	47.6	102	426.4	38.4
ホスト			46.1	47	427.8	38.3

この表より伝送時間、スループットがほぼ同じ値になっていることがわかる。この理由を図２－３２を用いて説明する。図２－３２はＦＴＰレベルで１レコードを転送した場合の各種の時間をＩＭＰ内で測定した結果を示している。この図より１メッセージを相手に転送してそのReceive（確認情報）が返ってくるまでの値が約２００ｍ秒と大きいことがわかる。このためＭ１（最初のメッセージ）を送出した後、相手ホストよりReceive が返ってくる間に次のメッセージ（Ｍ２）を送出する準備つまりヘッダの付加そして出力バッファへの転送等が終了する。これによりＢの値は小さくなる。

またＭ２に対するReceive が返ってくる間に次のレコードを読み込んでＭ１'（次のレコードの最初のメッセージ）を送出する準備をするならばＢ' も小さくなる。結局伝送時間／レコード（Ａ＋Ｂ＋Ａ'＋Ｂ'）が

ホストレベルの転送の場合のそれと同じになる。また、伝送時間より導き出されるスループットも同じになることも明らかである。

次にＣＰＵ時間については、ディスクのＩ／Ｏ、レコードの分割、ヘッダの付加、ヘッダの削除、レコードへの組み立て等でＣＰＵを使用する関係上増加している。

（ただし表２－９のＦ側の使用ＣＰＵ時間にはディスクＩ／Ｏの時間は含まれていない。）

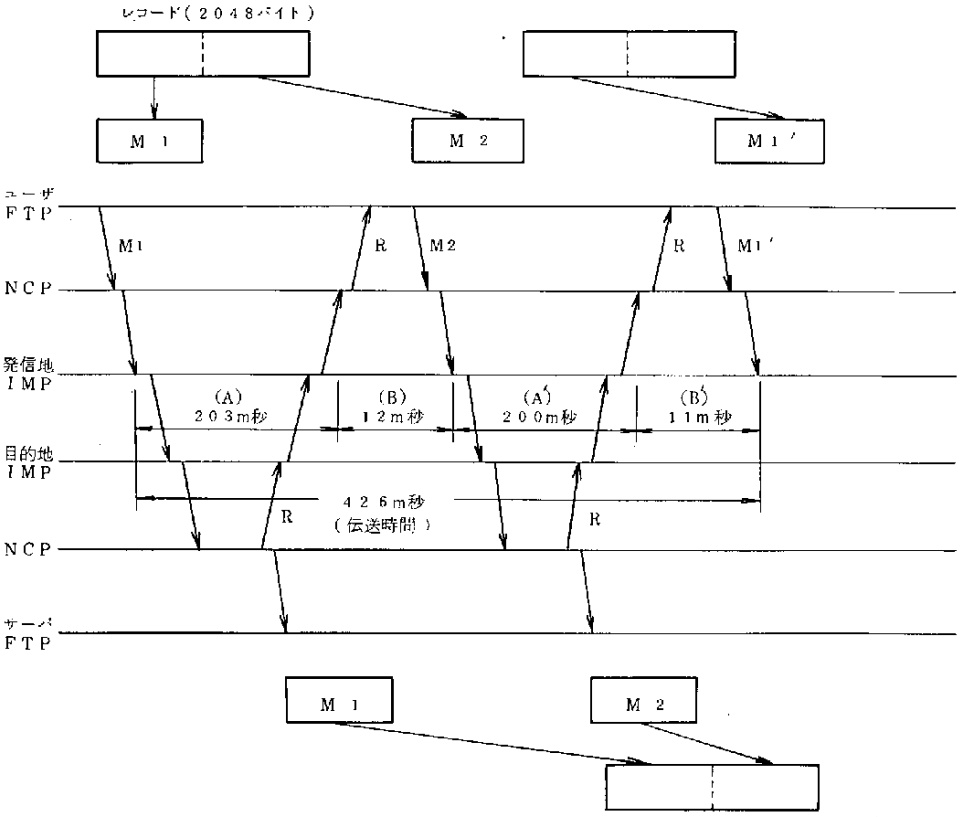


図２－３２　ＩＭＰ内のタイム・チャート１

第２番目として表２－７と表２－８を比較する。これはコード変換が効率に与える影響を調べるものである。結果としてコード変換をする側（Ｈ側）のＣＰＵ時間が増加している。これはコード変換のために必要なステップ数は２０４８バイト×１０００レコードでは１４３４万ステップにもなるため、５秒ほどＣＰＵ時間が増加する。

またスループットにはさほど影響を与えているとは思われない。

第３番目として表２－７を検討する。これはサーバＦＴＰがユーザＦＴＰから受信したソースプログラムをライブラリ形式でディスクに蓄積する場合に、効率に与える影響を調べるものである。表２－７からわかるようにＦ→Ｈ（Ｆ：ユーザＦＴＰ、Ｈ：サーバＦＴＰ）へのスループットは表２－６のそれの半分程に低下し、ＣＰＵ時間も増加している。この原因は、サーバＦＴＰ側（Ｈ側）で転送されてきたデータをライブラリ形式（３２０バイト／ブロック）に編集してディスコへ書き出す（ＥＸＣＰレベルでおこなう）時に時間がかかるため、ＮＣＰに対するＲＥＡＤを発信するタイミングが遅れることである。この結果ホスト・ホストプロトコルで規定されるフローコントロールの適用を受け、スループットに影響を与えている。フローコントロールを図２－３３を用いて説明する。今回の測定では受信側ＮＣＰに２個のバッファを用意している（①）。このバッファには送信側ＮＣＰからのメッセージが蓄えられ、サーバＦＴＰからのＲＥＡＤによりサーバＦＴＰへ転送される。また送信側ＮＣＰは、受信側ＮＣＰのバッファに空きのある状態（②）ではメッセージを送信できるが、なくなると送信側ＮＣＰはユーザＦＴＰの送信を持たせる（③）。そしてこの待ちはサーバＦＴＰがＮＣＰに対してＲＥＡＤを発信し、バッファの内容がＮＣＰからサーバＦＴＰへ転送され（④）、ＡＬＬＯＣコマンド（バッファが空きになったことを通知するコマンド）が受信側ＮＣＰから送信側ＮＣＰへ転送された時に解除される（⑤）。

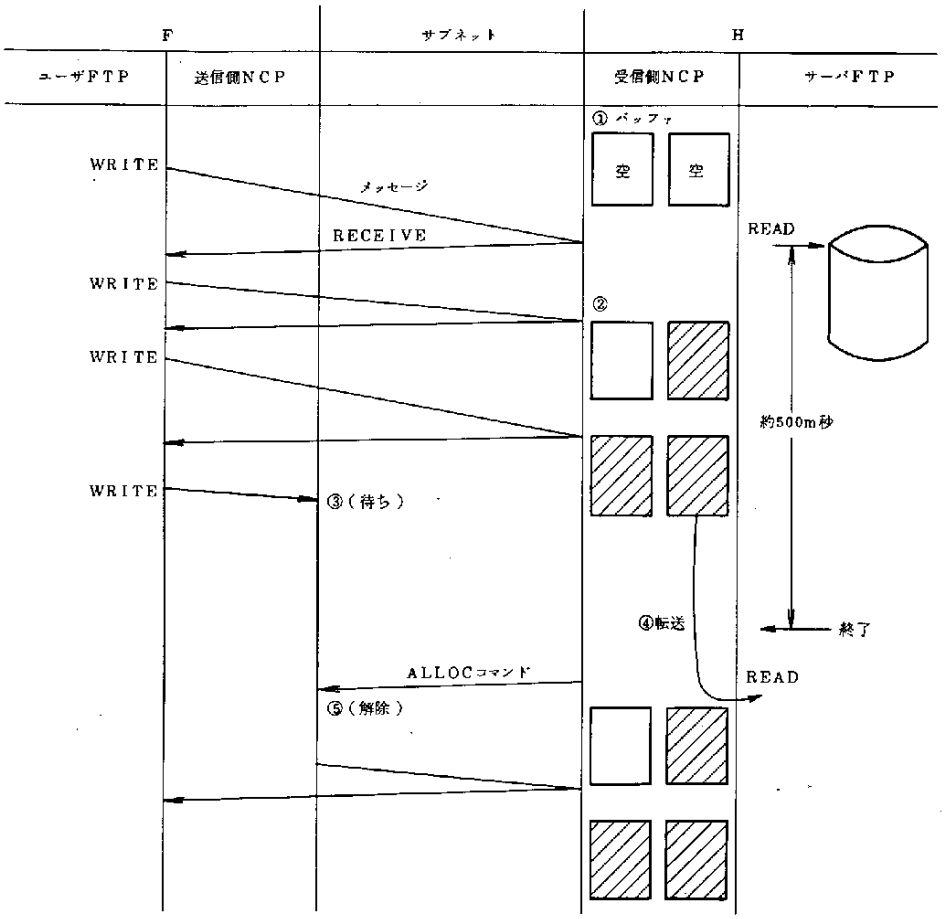


図２－３３　フローコントロール

上記のフローコントロールにより、図２－３４に示すようにＢとＢ'の値が大きくなってくる。このためスループットに多大な影響を与えることになる。

以上３点について、ファイル転送の効率に与える影響を検討してきたが、結論として今回作成したＦＴＰプログラムは、ソースプログラムをライブラリ形式で蓄積・管理する時にスループットの低下をまねくことを除いて、ほぼホストレベルのスループットと同じ位の値は実現できたといえる。これは、送信側ＦＴＰでは１個のメッセージを送出しその

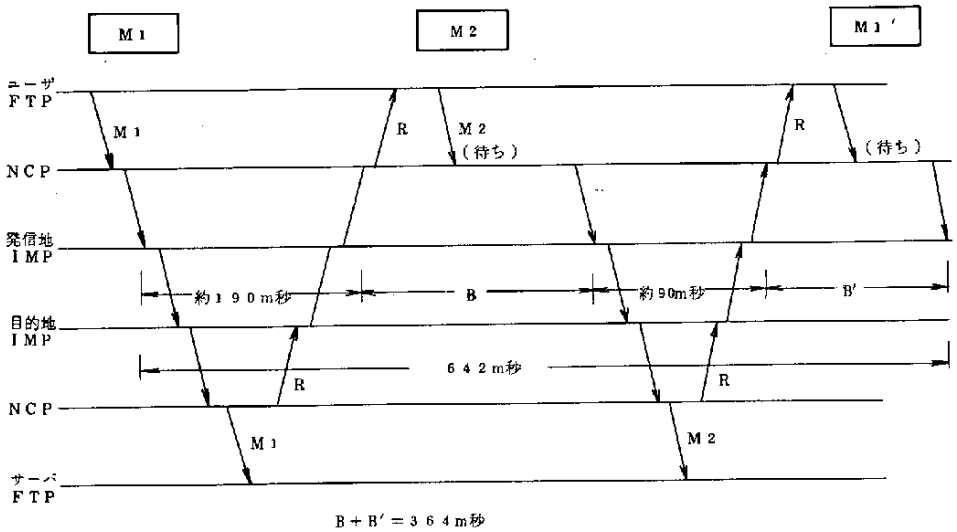


図２－３４　ＩＭＰ内のタイムチャート２

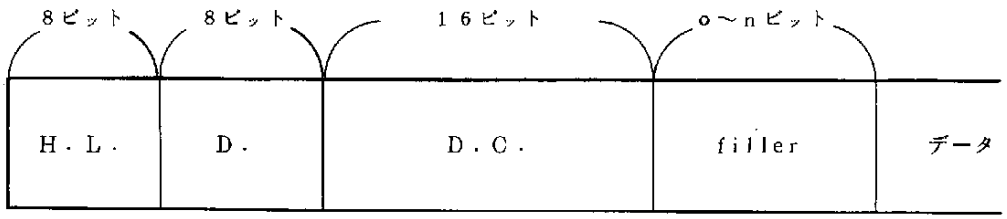
Ｒｅｃｅｉｖｅが返ってくるまでの時間（ラウンドトリップタイム）が大きいため、また受信側では受信するメッセージの間隔が大きいため、この時間を利用してディスクＩ／ＯとＦＴＰ独自の処理が終了するようにＦＴＰプログラムがインプリメントされていることを示している。

また今回の評価により、ＦＴＰ（ただしファイル転送に関係するものだけ）と、ホスト・ホストプロトコル以下のプロトコルに対して次のようなことがいえる。

(1)　ＦＴＰに対するコメント

①　ブロック転送方式の改良

ＪＩＰＮＥＴのＦＴＰは、レコードをホスト・ホストプロトコルに従って分割し、図２－３５に示すようにヘッダ付きメッセージとして転送する方式をとっている。この方式に対する改良点は２つある。



H．L．：Header Length ── ヘッダ部の長さ

D．　：Descriptor ── ヘッダ情報の種類

- EOR：End Of Record ── 1
- EOF：End Of File ── 2

D．C．：Data Count ── データ部の長さ

filler：バウンダリ調整のためのエリアで、各ホスト特有のバウンダリはこのfillerで調整する。

図2－35　ヘッダ付きメッセージ

1つは、アーキテクチャがワードマシンであるホスト（例えばFACOM）の場合、現在のプロトコルではデータの先頭をバウンダリに合わせるために、相手ホストにより1ワード～9ワードのヘッダを作成しなければならない。これはヘッダの分だけ1メッセージで転送できるデータのバイト数が減ることは確実であり、伝送時間においても2.3m秒～20.4m秒のオーバヘッドをもたらす。またData Count（D.C.）の必要性はないと思われる。それは各FTPプログラムあるいは他の処理プログラムのNCPに対するREADの完了時には、実際に転送されてきたメッセージのバイト数がNCPから完了情報と伴に通知されるようになっているためである。これらのことより図2－36で示すようにヘッダ部を削除しメッセージの種類を示すDescriptorバイト（8ビット）をデータ部の最後尾に付加する転送方式にするならば、データの先頭のバウ

ンダリを調整する必要もなくまた1バイトのオーバヘッドですむことになる。

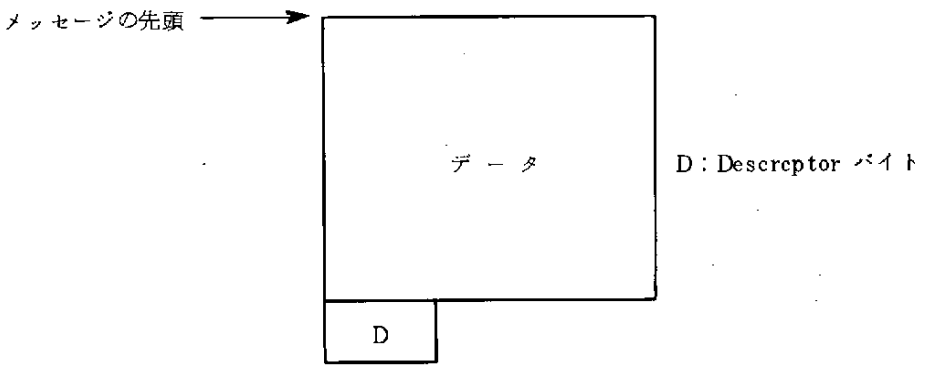


図2－36　ブロック転送方式の改良

いま1つは、データ部の最後も受信側のバウンダリに合わせるように送信側ホストはレコードを分割し送信すべきである。これは先程と同様にワードマシンである場合に、使用ＣＰＵ時間を減少させる有効な方法であると思われる。今回のＦＴＰではこのことについて規定はなく、送信側ＦＴＰは図2－37のようにして送信する場合がある。

この場合受信側ＦＴＰでは受信したメッセージをレコードに組み立てるために、最初のメッセージの最終バイトの次の位置（図中のＡ）から次のメッセージのデータを移動させねばならない。このため1,０００レコードを転送した場合には、受信側でおこなうデータ移動に必要なステップ数は数百万ステップになり、ＣＰＵ時間として数秒程度のオーバヘッドになる。

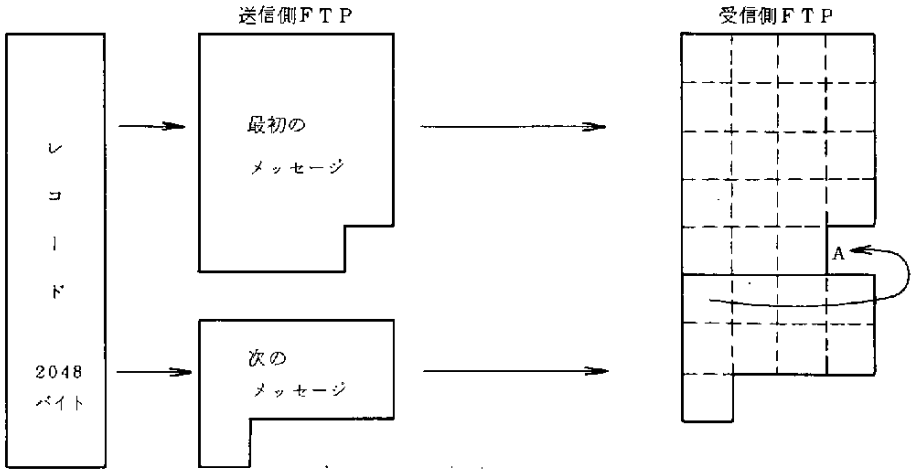


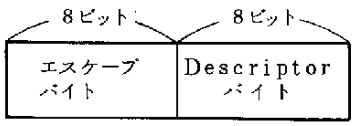
図２－３７　データ移動のオーバヘッド

②　コンプレス（圧縮）転送方式の追加

コンプレス転送方式は、ソースプログラムのように同一コードあるいはブランクが連続するファイルを転送する場合に、送信側ＦＴＰはデータを圧縮して転送し、受信側ＦＴＰはそのデータを元にもどし蓄積するものである。この方式が有効に作動するならば、少ない転送回数で大量のデータを転送できるためスループットが飛躍的に大きくなる可能性がある。以下にコンプレス転送方式の例を述べる。

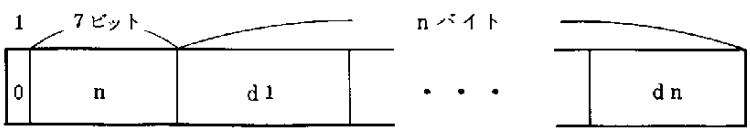
転送すべき情報としては次の３種類がある。

イ　コントロール情報

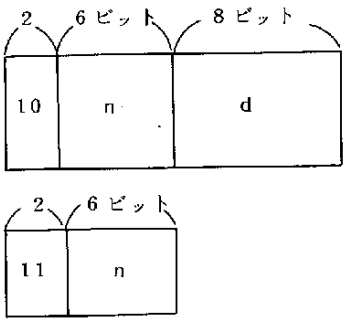


エスケープバイト：all　0のバイト
Descriptorバイト：ＥＯＲまたは
ＥＯＦ等を示す。

ロ　一般データ（圧縮できないデータ）



ハ　圧縮データ



○　同一コード

○　filler

これらの3種類の情報を図2－38のように組み合わせて転送する。

コントロール情報	一般データ and／or 圧縮データ	コントロール情報	…

図2－38　コンプレス転送

(2)　ホスト・ホストプロトコル以下のプロトコルに対するコメント

①　ＮＣＰに対する連続ＷＲＩＴＥ

現在のホスト・ホストプロトコルでは、ＮＣＰに対するユーザプログラムからのＷＲＩＴＥは、前のＷＲＩＴＥが完了しなければ発信できないようになっているため、ＦＴＰのように大きなサイズのレコードを1単位として転送する場合には、レコードを数回に分けて送信しなければならない。このため1レコードの送信完了までの時間（伝送時間）は2つ以上のメッセージを送信するのに要した時間の和となる。しかし、ＮＣＰに対する連続ＷＲＩＴＥが許されるならば、図2－39で示すように各メッセージがオーバラップして送信されるため伝送時間が小さくなりスループットがさらに向上すると思われる。

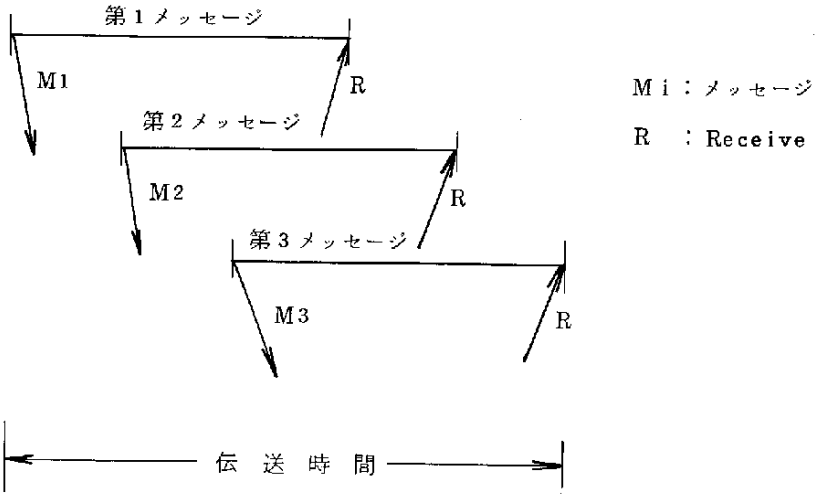


図2－39　NCPに対する連続WRITE

ただし、この場合にはホスト・ホストプロトコルのフローコントロールの適用を受ける可能性があるため、上記のように改良するにはフローコントロールの改良も必要となってくる。

② メッセージ長の拡張

前項のホスト・ホストプロトコルの効率評価において、NCPとIMPおよびユーザプログラムとの相互交信の回数が使用CPU時間に大きな影響を与えているため、送受信するメッセージ長あるいはパケット長を大きくすることによって、スループットの向上そして使用CPU時間の減少が可能であると結論づけた。このことはFTPにおいてもレコードをメッセージに分割しヘッダを付加する、あるいはヘッダを削除しメッセージをレコードに組み立てる作業がなくなることを意味しているため、スループットの向上ならびに使用CPU時間の減少につながることは確実である。

③ 各処理形態に合うプロトコルの設定

表2－6のF→Hへの測定結果に示される同一IMP結合

（o hop）と回線経由結合（full）のデータを比較検討した結果より次のことがいえる。通常のコンピュータ・ネットワークにおいてはホストの処理能力がサブネットの伝送能力を上回っているが、同一ＩＭＰに結合したホスト間ではサブネットの伝送能力がホストの処理能力を上回るのでＡＬＬＯＣコマンドの送出が頻繁に起こりスループットの低下をまねく。このことはスループットの異なる形態が同居するコンピュータ・ネットワークでは、ホスト・ホストプロトコル以下のレベルでそれぞれに異なったプロトコルをサービスする必要があると思われる。

最後にＦＴＰプログラムを利用して、ハードウェア・リソースそしてソフトウェア・リソースを共有した実例を述べる。

① 処 理 概 要

- ＦＴＰプログラムを使用して、ＨＩＴＡＣにある数値計算ライブラリ（ＭＳＬ II）から2つのプログラムをＦＡＣＯＭへ転送し、ＦＡＣＯＭでそれを実行する。
- ＦＡＣＯＭで実行した場合の使用ＣＰＵ時間、コア使用時間そして転送時間等を測定し、それを表２－１０で金額に換算する。またＨＩＴＡＣ内でおこなった場合の使用ＣＰＵ時間そしてコア使用時間も表２－１０で金額に換算する。
- 両者を比較する。

表２－１０ 換 算 表

料金名 / 機種名	ＣＰＵ料金	コア料金
ＦＡＣＯＭ２３０／７５	１６０円／秒	４０円／秒
ＨＩＴＡＣ ８４５０	５６円／秒	１４円／秒

② 使用数値計算プログラム

- ＬＥＳＱ２ ― 未知数の個数より方程式の個数の方が多い連立１次方程式の解を最小二乗法（重みなし）により１倍精度で求めるもので、内部でサブルーチンＬＩＮＳ２を使用する。
- ＬＩＮＳ２ ― 実係数ｎ元１次方程式の解を消去法（ガウス・ジョルダン法）により１倍精度で求める。

③ 処理パターンと結果

- サブルーチンＬＥＳＱ２を用いて未知数３０、方程式の個数４０の連立方程式の解を求める。表２－１１に結果を示す。

表２－１１　処理パターン１

		ＨＩＴＡＣ		ＦＡＣＯＭ	
		CPU TIME	CORE TIME	CPU TIME	CORE TIME
コンパイル	FORTRAN MAIN	1 S	10 S	0.2 S	9.684 S
	FORTRAN SUB	1 1	12 13	0.5	13.293
ＬＩＮＫ		6	28	0.7	12.166
実行		5	6	0.5	3.699
ＴＯＴＡＬ		15	86	1.9	38.842
費用		840 円	1,204 円	304 円	1,554 円
		2,044 円		1,858 円	

ＦＴＰ関係の費用：（７ブロック＋３ブロック）×１１.８１＝１１８.１円

ＦＡＣＯＭで実質的にかかる費用＝1,８５８＋１１８＝1,９７６円

○　サブルーチンＬＥＳＱ２を用いて未知数６０、方程式の個数７０の連立方程式の解を求める。表２－１２に結果を示す。

表２－１２　処理パターン２

		ＨＩＴＡＣ		ＦＡＣＯＭ	
		CPU TIME	CORE TIME	CPU TIME	CORE TIME
コンパイル	FORTRAN MAIN	1 S	10 S	0.2 S	12759 S
	FORTRAN SUB	1 1	11 15	0.5	16489
ＬＩＮＫ		5	28	0.7	17944
実行		22	32	2.7	7019
ＴＯＴＡＬ		31	117	4.1	54,211
費用		1,736 円	1,638 円	656 円	2,168 円
		3,374 円		2,824 円	

ＦＴＰ関係の費用：（７ブロック＋５ブロック）×１１.８１＝１４１.７２円

ＦＡＣＯＭで実質的にかかる費用＝2,８２４＋１４２＝2,９６６円

このように他系のホストにある特殊なプログラム（ただしＦＯＲＴＲＡＮ等で作成されており、どの計算機でも実行できるもの）を自系のホストに転送し実行することも可能であり、また処理速度の速い計算機を持つホストへ転送しそこで実行することも可能である。そして図２－４０で示すようにＲＢＰ（Remote Batch Protocol）等と共同してさらに広範な利用形態も考えられる。

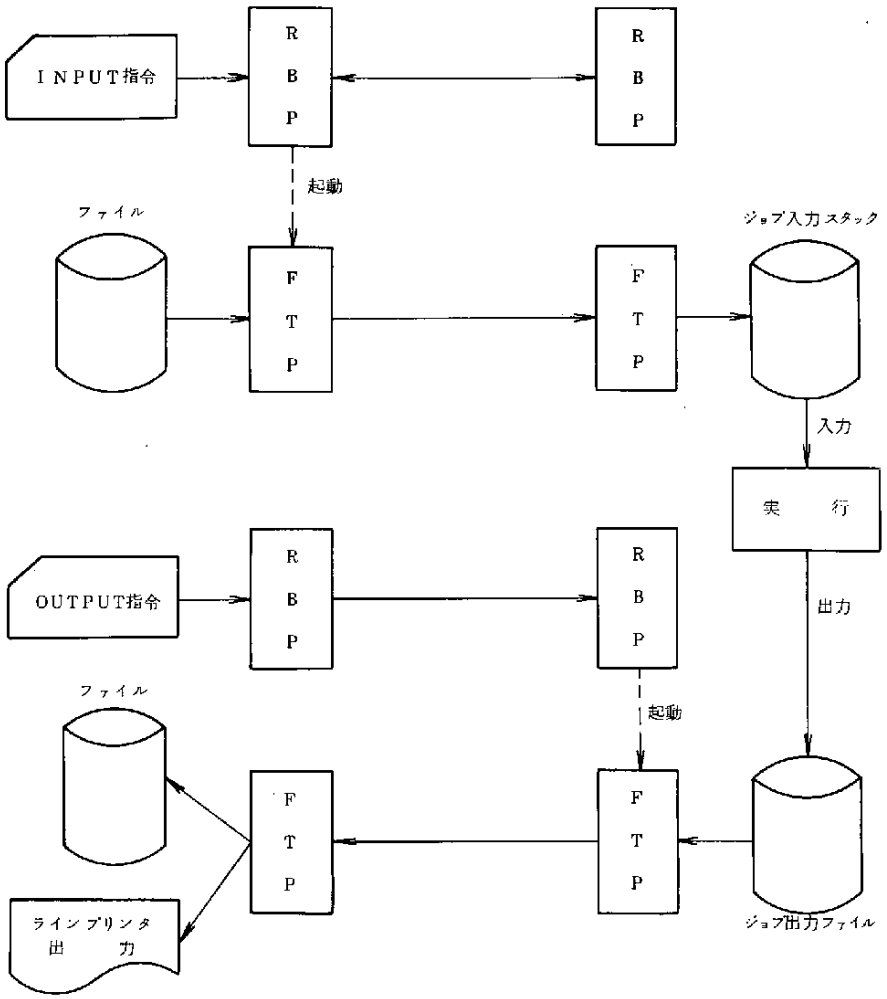


図２－４０　ＦＴＰの利用形態

2.4.2　デマンド・サービス・プロトコル

A　測定目的と方法

コンピュータ・ネットワークにおいてもＴＳＳは、最も一般的な利用形態の一つである。Demand Service Protocol（ＤＳＰ）は最も多く利用される高位プロトコルであり、あるコンピュータに接続された端

末を別のコンピュータのＴＳＳへ論理的に接続し、ユーザにその端末を介して、あたかもそのＴＳＳのローカル・ユーザのようにそのＴＳＳと会話できるようにする。

図２－４１に示すように、ＤＳＰはユーザの端末を制御するＤＳＰユザ・プロセスと、ターゲットＴＳＳへのアクセスのインタフェースを取るＤＳＰサーバ・プロセスから成り、両プロセスの間でのデータの授受の手順、フォーマット、そしてタイミングなどを規定したものである。

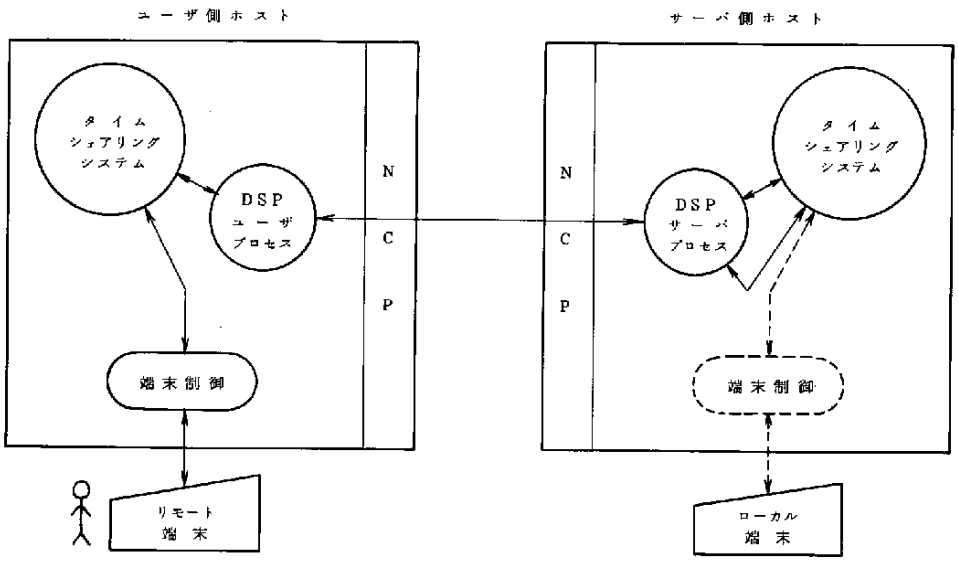


図２－４１　ＤＳＰの実現形態

ＤＳＰサーバ・プロセスは、リモート端末とデータの授受を行なうが、ユーザが使用する端末は多種多様であり、一つ一つの端末の特性をいちいち意識して制御を行うのは大変である。このため、Network Virtual Terminal（ＮＶＴ）と呼ばれるネット・ワーク内での標準的な

機能を具備した端末を定義し、ＤＳＰサーバ・プロセスは実際の端末を意識せずに、このＮＶＴとデータの授受を行ない、一方ＤＳＰユーザ・プロセスがＮＶＴと実端末との変換を行なう。

ネットワークを経由してＴＳＳを使用するユーザは、ローカル端末よりＴＳＳを使用した場合に比べ、ネットワークを経由したためにどの位レスポンスが遅くなったか、どの位余分のコストを取られるかが関心事となる。

ＪＩＰＮＥＴには、ＴＳＳ機能をもつホストとしてはＦＡＣＯＭ230－７５（以下Ｆ７５と略す）とＡＣＯＳ７７ ＮＥＡＣシステム７００（以下Ａ７００と略す）の２台ある。Ｆ７５にはローカル端末が接続されているが、一方Ａ７００にはローカル端末が接続されていない。

サブネットは３台のノード・プロセッサを三角形に結合した形態となっている。その内の２台はＴＩＰ機能を具備し、合計５台の端末が接続されている。

ネットワークＴＳＳとローカルＴＳＳを比較するために、Ｆ７５のＴＳＳを主に測定対象とした。しかし、Ｆ７５の既存のＯＳやＴＳＳそのものの評価には立ち入らず、ネットワーク・サービスを行うために追加された部分のみの評価に止める。

指定した個数の固定長メッセージ（２０字）を出力するユーザ・プログラムを作成し、このプログラムをターゲットのＴＳＳで実行し、次の項目を測定した。

① サーバ側ホストのメッセージの送信に使用したＣＰＵ時間

② レスポンス時間

B 測 定 結 果

測定結果としては、サーバ側（Ｆ７５）の測定プログラムとＤＳＰサーバ・プロセスのＣＰＵ時間はアカウンティング・データを、ＮＣＰのＣＰＵ時間はホスト間通信の測定データを、経過時間に関してはストッ

プ・ウオッチによる計測データをそれぞれ使用した。但し、Ｆ７５のアカンティング・データの精度は0.1秒であり、0.1秒以下は切上げである。

測定に使用した端末は、Ｆ７５に接続されている富士通製の２００ＢＰＳ用のＦ１５９１Ａタイプライタ型端末装置と、ＴＩＰに接続されているＧＥ社製の２００ＢＰＳ用のTermi－Net 300 タイプライタ型端末装置である。

(1) ＣＰＵ時間

ＴＩＰからネットワーク経由してＦ７５のＴＳＳを使用し、測定プログラム、ＤＳＰサーバ・プロセス、そしてＮＣＰのＣＰＵ使用時間を測定した（表２－１３、図２－４２参照）（図ではy_1 で示してある）。図２－４２には、比較の意味でローカル端末よりの測定結果も示してある（図ではy_2 で示してある）。

表２－１３　ネットワークによるＣＰＵオーバヘッド

（単位：秒）

送信メッセージ数（メッセージ長：２０字）	測定プログラムの使用ＣＰＵ時間	ＤＳＰサーバプロセスの使用ＣＰＵ時間	ＮＣＰの使用ＣＰＵ時間	合　計
１４７	0.3	0.2	2.426	2.926
５４７	0.5	0.2	9.026	9.726
１０４７	0.7	0.2	17.276	18.176

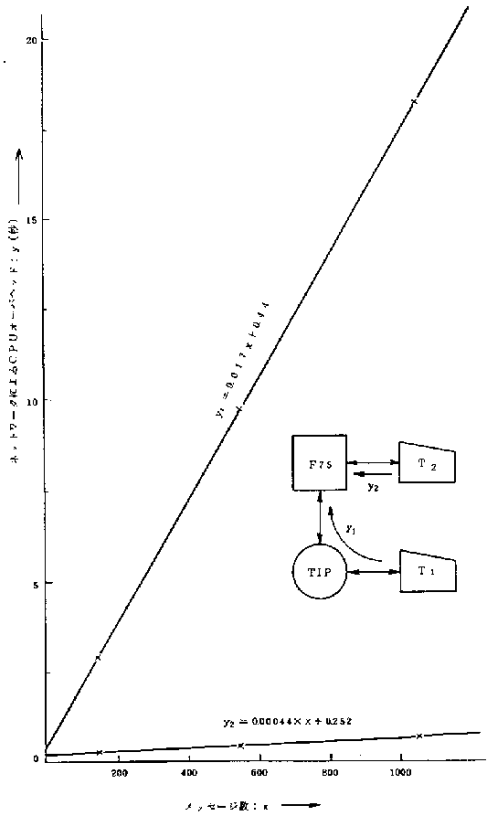


図２－４２　ネットワークによるＣＰＵオーバヘッド

図２－４２から、ネットワークによるＣＰＵオーバヘッドの実験式 y_1 を最小二乗法で求めると、

$$y_1(x) = 0.017x + 0.44$$

である。ここで、ｘは送信メッセージ数である。ｘの係数0.017は１メッセージを送信するために必要なＣＰＵ時間であり、定数項0.44は測定プログラムとＤＳＰサーバ・プロセスの初期設定時に使用されるＣＰＵ時間である。

サーバ側ホストのＣＰＵオーバヘッド時間 y_1 はネットワークを経由しない場合（ y_2 ）に比べて、１メッセージ当り約１７ミリ秒にな

る。この１７ミリ秒の大部分（約１６.５ミリ秒）はＮＣＰのＣＰＵオーバヘッド時間であり、ＤＳＰサーバ・プロセスのＣＰＵオーバヘッドはＮＣＰに比べ無視できる。

(2) レスポンス時間

ＪＩＰＮＥＴには、ＴＳＳ機能をもつホストとしてはＦ７５とＡ７００の２台、そして端末が接続されているホストとしてはＦ７５とＴＩＰである。利用可能なすべての経路を経由してＴＳＳを使用し、レスポンス時間の総和、即ち経過時間を測定した（図２－４３）。

図２－４３から、Ａ７００をＴＩＰに接続されている端末T_1より使用した場合（図ではy_1で示してある）に比べ、Ａ７００をＦ７５経由で同じ端末T_1より使用しても（図ではy_3で示してある）、経過時間には大差がないことがわかる。これは、y_3の場合Ａ７００よりのメッセージに対するＡＣＫはＦ７５で返すので、以後Ａ７００とＦ７５の間でのメッセージの伝送と、Ｆ７５とＴＩＰ及びT_1の間でのメッセージの伝送がオーバラップするためである。

端末T_1よりＡ７００（経路y_1）とＦ７５（経路y_2）を使用した場合、両者は経過時間に関しては大差がない。この測定プログラムに関しては、両ホストとも同じような性能である。

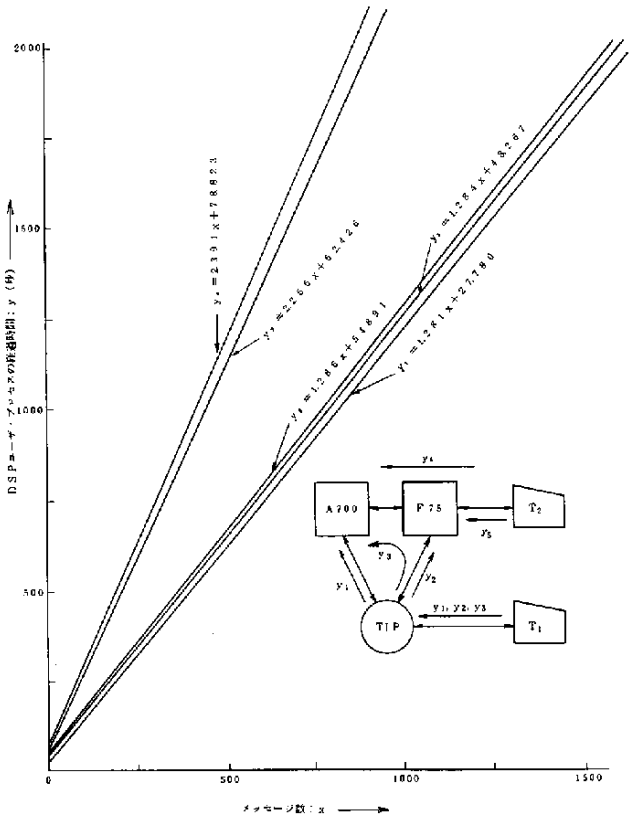


図２－４３　利用経路による経過時間の差違

Ｆ７５に接続された端末T_2 よりＦ７５を使用した場合（経路y_5）に比べ、Ａ７００をＦ７５経由で同じ端末より使用しても（経路y_4）経過時間は大差がない。これは、y_4 の場合Ａ７００よりのメッセージに対するＡＣＫはＦ７５が返すので、以後Ａ７００とＦ７５間でのメッセージ伝送と、Ｆ７５と端末T_2間でのメッセージの伝送がオーバラップするため、Ｆ７５での遅延が有意とならなくなる。

y_1 、y_2 、y_3 のグループとy_4 、y_5 のグループでの経過時間の差は、端末の性能と伝送制御手順の違いであり、ネットワークには無関係である。

サービスする端末数が増えることによるレスポンス時間の増大は、既存のＴＳＳやＯＳの性能にディペンドし、ネットワーク固有の要素ではない。

C　検討および評価

(1)　共通コマンドと最適なＴＳＳの選択

ＴＩＰからネットワーク中の複数のＴＳＳをアクセスする場合、それらのＴＳＳは各々独自のアクセス手順やコマンド体系を採用しており、それらを自由自在に使いこなすには、それらに対する多種多様の異なったアクセス手順やコマンド体系を熟知しなければならない。このことが少なからず複数のＴＳＳ使用の障害となっている。そこで、ユーザがネットワーク中のすべてのＴＳＳに対し、容易にアクセスできるような共通のアクセス手順やコマンド体系を必要とする。

この共通のアクセス手順やコマンド体系をサポートするようにネットワーク中のすべての既存のＴＳＳに対して変更や追加を行うことは現実には困難である。ＴＩＰが各々のホストに代って、この共通のアクセス手順やコマンド体系をターゲットＴＳＳのローカルなそれに変換を行った場合、主記憶容量や補助記憶容量の点で実現不可能である。

あるＴＳＳを使用するのに別のＴＳＳを経由して使用しても、レスポンス時間には大差がないので、この中継用のＴＳＳに共通のアクセス手順やコマンド体系をターゲットＴＳＳのローカルなそれへの変換作業を行なわせることが可能となる。

ＴＩＰからＴＳＳを使用する場合、必ずこの中継用のＴＳＳを経由するようになると、この中継用のＴＳＳがネットワーク中に使用可能なＴＳＳが複数個ある場合、最適なＴＳＳを選択できるし、もし選択

されたＴＳＳに障害が発生した場合、自動的にバック・アップ用のＴＳＳへ切換えることも可能となる。

(2) 仮想端末／実端末変換のオーバヘッド

ここでは、ＴＩＰ上での仮想端末から実端末へ変換するに要するオーバヘッドがどの位になるか検討する。処理に要する時間は推定走行ステップ数より算出した。

① 機能文字の検出

サーバＤＳＰとユーザＤＳＰの間で一度に授受されるデータ・ストリュームは最大１４４文字であり、データ・ストリーム中の任意の位置に機能文字を入れることができる。このため、データ・ストリューム中の機能文字を捜すために、データ・ストリューム全体を調べなければならず、オーバヘッドは数ミリ秒にも達する。

② コード変換

ＮＶＴ（Network Virtual Terminal）コードとしてＪＩＳ７単位でのカナ文字符号も使用可としている。ＴＩＰに接続されているすべての端末は米国製であり、カナ文字の印字ができない。このため、ソフトウェアによってカナ文字／ローマ字変換を行っており、オーバヘッドは約５ミリ秒である。

③ 復改動作

復改（ＮＬ，New Line）動作は、復帰（ＣＲ，Carriage Return）と改行（ＬＦ，Line Feed）の２動作で行なう端末と、どちらか（ＣＲ or ＬＦ）のコードで１動作で行なう端末があり、これらをシミュレートするに要するオーバヘッドは最大１ミリ秒にも達する。

また、復改機能しか持たない端末に対し、復帰と改行を独立した機能としてサービスを行うと、かなりのオーバヘッドを必要とする。

④　タイミングの調節

プロトコルでは、ＣＲ、ＬＦ、ＢＳなどの動作時間に関しては規定されていない。これらの制御文字は、グラフィック文字に比べ数倍の動作時間を必要とする。

表２－１４、表２－１５にＧＥ社のTermi-Net　３００プリンタの動作時間を示す。

表２－１５より３０ＣＰＳ（Characters／Sec.）の場合、復改動作を行うには、ＣＲ、ＬＦの後に６個のｆｉｌｌ　ｃｈａｒ（通常はＤＥＬ文字）を必要とし、この文字を埋込めに必要なオーバヘッドは約５ミリ秒に達する。

Texas Instruments社のＳｉｌｅｎｔ７００　Ｍｏｄｅｌ　７３３カセット付端末は、伝送速度が１２０ＣＰＳ（１２００ＢＰＳ）であり、一方印字速度が３０ＣＰＳである。伝送速度と印字速度の速度差は図２－４４に示すように各文字の後に数個のｆｉｌｌ　ｃｈａｒを埋込んで調節される。この処理に要するオーバヘッドは約５ミリ秒になる。

表２－１４　基本タイミング

	Character Rates Per Second		
	１０	１５	３０
Bits per second	109.95	150.00	300.00
Bit time	9.10 MS	6.67 MS	3.33 MS
Character time	100.05 MS	66.67 MS	33.33 MS
Number of stop bits	2	1	1

表2－15　タイム・ディレー

FUNCTION	INTERVAL BETWEEN	APPROX. INTERVAL IN MILLI-SECONDS	NON-PRINTING FILL CHARACTERS REQUIRED 30CPS	15CPS	10CPS
Normal Line Feed	Commanding last character on old line and first character on new line	300	6	1	0
Consecutive Line Feeds	Consecutive executions of line code	58	2	1	0
Backspace	Printing and then reprinting in the same position	222	6	3	2
Startup*	Commanding motor ON and printing	420	12	6	4
Shutdown**	Last data character and commanding motor OFF	222	6	3	2

*Startup-The TermiNet 300 Printer will normally go from a non-printing or "Standby" state to a printing or "On Line" state under code control. The two-code sequence for Motor On is ESC hor H. A time delay of 420 ms should be allowed before data transmission can begin.

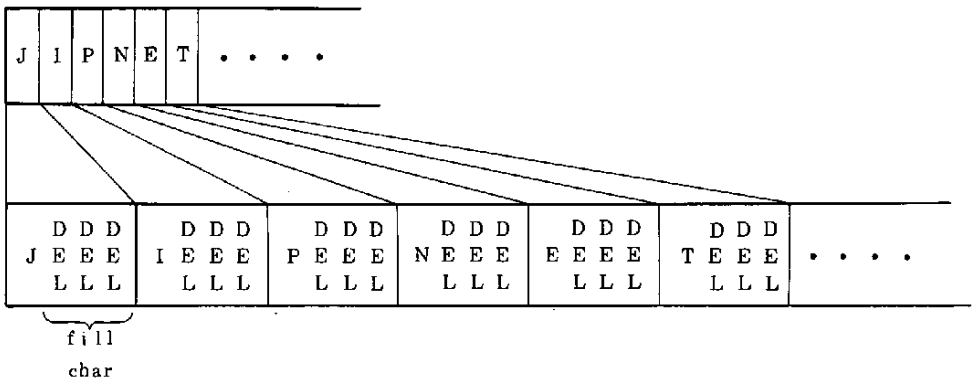
A special version of the printer can be supplied where the printing motor will be commanded "On" whenever the data set CB lead (Clear to Send) is activated. In this case, 420 ms delay is necessary between that signal and data flow.

**Shutdown may be accomplished by either ESC j or J or by EOT.

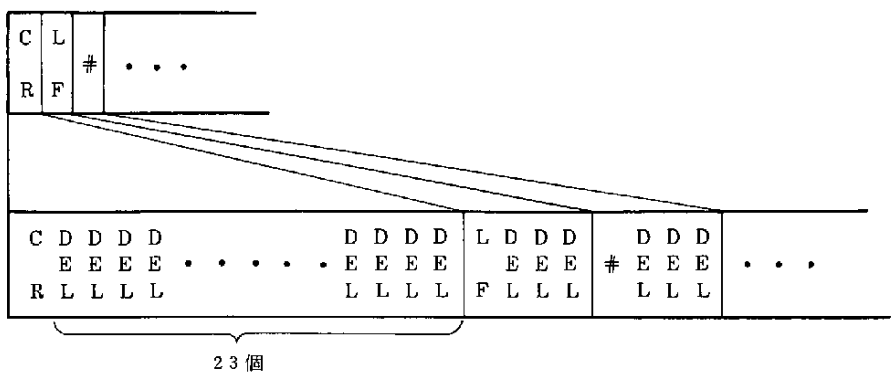
EXAMPLES

FUNCTION	30CPS	15CPS	10CPS
Normal Line Feed	...data...CR,LF, 6 fill char	...data...CR,LF ··fill, data	...data...CR,LF
Consecutive Line Feeds	LF,fill,fill,LF, fill,fill,etc.	LF,fill,LF,fill, etc.	LF,LF,LF
Backspace	BS,6 fill char, ***reprint	BS,3 fill char, ***reprint	BS,2 fill char, ***reprint

***If Backspace code is used more than once, it may take the place of fill characters; e.g., at 30 cps, to type and underscore "AND", send "A, N,D,BS,BS,BS,fill,fill,fill,fill,_,_,_"



(1)　伝送速度（１２０CPS）と印字速度（３０CPS）の速度差の調節



(2)　復改動作時間の調節

図２－４４　伝送速度と印字速度の速度差の調節

以上をまとめると、仮想端末と実端末との変換に要するオーバヘッドは少なくとも７ミリ秒以上になる。

ＴＩＰは、１メッセージを処理するのに全部で１０～１５ミリ秒のＣＰＵを消費している。ＴＩＰはＩＭＰ上に同居しており、ＩＭＰが大量のトラフィック（1.15ミリ秒／パケット）を処理している場合、ＴＩＰのサービス率が低下する。

(3) ホスト－ホスト・レベルのフロー・コントロール

ユーザ・メッセージを送信する場合、送信側ＮＣＰは、受信側ＮＣＰがそのコネクションに対して割当てたバッファが一杯になった場合、受信側ＮＣＰよりのＡＬＬＯＣコマンドを待たなければならない。

ＴＩＰでは、１コネクションに２個のバッファを割付けており、２メッセージ受信する毎にＡＬＬＯＣコマンドを送信している。即ち、サーバ・ホストからＴＩＰへのホスト－ホスト・レベルの回線オーバヘッドは３３％である。サーバ側ＮＣＰはこのＡＬＬＯＣコマンドを受信しなければならず、その分だけＮＣＰのオーバヘッドが増大する。

一方、サーバ側ホストはＴＩＰと同様に１コネクションに２個のバッファを割付けているが、ＡＬＬＯＣコマンドはReceive 信号に便乗させており、ＡＬＬＯＣコマンド単独では送信されない。

2.4.3 リモート・バッチ・プロトコル

リモート・バッチ・プロトコル（ＲＢＰ）は、他系ホストのバッチ処理機能を利用するための高位プロトコルである。ＲＢＰでは、リモート・バッチ特有の動作も数種の標準コマンドとして規定し、各オペレーションは分割して行うことができるよう設定されている。標準オペレーションを行うためのコマンドとしては、入力指令、出力指令、状態問合せ、打切指令終了指令等がある。

A　測定目的と方法

(1)　測　定　目　的

ユーザからみたＲＢＰの処理効率は、自分の依頼したジョブのターン・アラウンド時間がどれだけかということにより、その良し悪しが判断される。しかしターン・アラウンド時間は、ユーザ・ジョブの性格と、ユーザ・ジョブの受付ホストおよびリモートの処理ホストにおけるジョブの混み具合に依存する部分が大きいので、今回の測定ではこのような可変要素を除き、外部環境を単純化することにより、ＲＢＰの処理プロセス自体の処理効率として測定することとした。このような観点に立った場合、測定すべき主たる項目は、ユーザが入力指令によりジョブ・デック送出依頼を行った時、または出力指令により出力結果の返送依頼を行った時の処理効率としてのスループットおよびＲＢＰプロセスの使用するＣＰＵ時間になる。

ユーザから見たＲＢＰ処理のターン・アラウンド時間は、ネットワーク処理特有の要素である入出力指令の処理効率が測定されていれば、ある程度推測が可能であろう。

次に、システム・サイドからみた場合、この測定項目は、前項で記したＦＴＰの測定項目と一致することになったが、ＲＢＰにおける測定値とＦＴＰにおけるそれとを比較することによりインプリメントの妥当性の評価も可能となる。

(2)　測　定　方　法

ＲＢＰプロセスのスループットおよび使用ＣＰＵ時間の測定は、実際のジョブ・デックを受付ホストから投入し、サーバ・ホストから処理結果が返送されるＲＢＰ処理において、入力指令および出力指令実行時のスループットと使用ＣＰＵ時間をジョブ・アカウンティング結果により測定する方法により行った。

今回の測定では、User ＲＢＰが存在する受付ホストがＦ２３０／

７５、Server ＲＢＰが存在する処理ホストはＨ－８４５０であり、両ホストが同一ＩＭＰに結合されている場合と、サブネットを経由して結合されている場合の両者のシステム構成にて測定した。

測定に用いたデータは、カード・イメージのジョブ・デック数種であり、入力カード枚数および出力行数は大・小適当な値を持つものを選択している。

B　測　定　結　果

以下にＲＢＰ処理におけるプロトコル処理プログラムのＣＰＵ時間、ジョブ時間（core time）、スループットの測定結果を記す。

表２－１６は、ジョブ・デック転送の入力指令の測定値を示し、表２－１７は出力結果転送の出力指令の測定値を示している。表２－１６のUser（Ｆ／７５）core time の測定値において、カッコ内の数値はServerホストであるＨ－８４５０でジョブ起動の為に要する時間であり、この間ユーザ・サーバ間のデータ転送は皆無である。これはＣの検討の項で述べるが、Ｈ－８４５０でのインプリメント上の制限によるもので、本来はゼロであるべきであろうが、参考のために記した。同様に表２－１６スループットにおいても、カッコ内はサーバ・ホストにおけるジョブ起動時間を除いた場合の数値を示している。

表２－１６、１７ともServer（Ｈ－８４５０）CPU time において同一ＩＭＰの場合の測定値は記していないが、回線経由の場合とほぼ同一とみなしてよい。

表２－１６　入力指令の測定値

測定項目＼カード枚数（８０字／枚）		５９４枚	1,１０６枚	3,１５４枚	6,２２６枚
User（F/75）	回線経由	0.3	0.4	1.1	2.2
CPU time (s)	同一IMP	0.3	0.4	1.1	2.2
Server（H-8450）	回線経由	8	17	27	49
CPU time (s)	同一IMP				
User（F/75）	回線経由	25.7 (10.5)	43.1 (20.2)	85.9 (31.2)	158.9 (57.7)
Core time (s)	同一IMP	26.3 (13.4)	39.3 (20.5)	74.0 (31.0)	135.9 (57.3)
スループット	回線経由	14.8 (24.9)	16.4 (30.9)	23.5 (36.9)	25.1 (39.4)
（Kbps）	同一IMP	14.5 (29.5)	18.0 (37.6)	27.3 (46.9)	29.3 (50.7)

表２－１７　出力指令の測定値

測定項目＼出力行数（１４０字／行）		９４０行	1,４５０行	3,５１０行	6,５９０行
User（F/75）	回線経由	0.8	1.2	2.6	4.8
CPU time (s)	同一IMP	0.8	1.2	2.6	4.8
Server（H-8450）	回線経由	5	5	10	17
CPU time (s)	同一IMP				
User（F/75）	回線経由	59.9	76.5	145.4	241.8
core time (s)	同一IMP	28.3	34.1	57.1	91.4
スループット	回線経由	17.6	21.2	27.0	30.5
（Kbps）	同一IMP	37.2	47.7	68.8	80.8

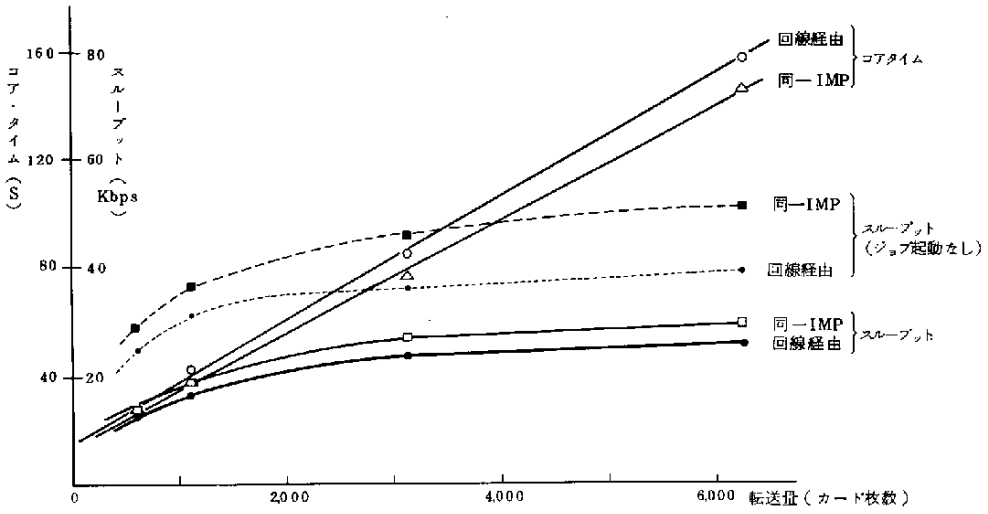


図２－４５　入力指令のスループット

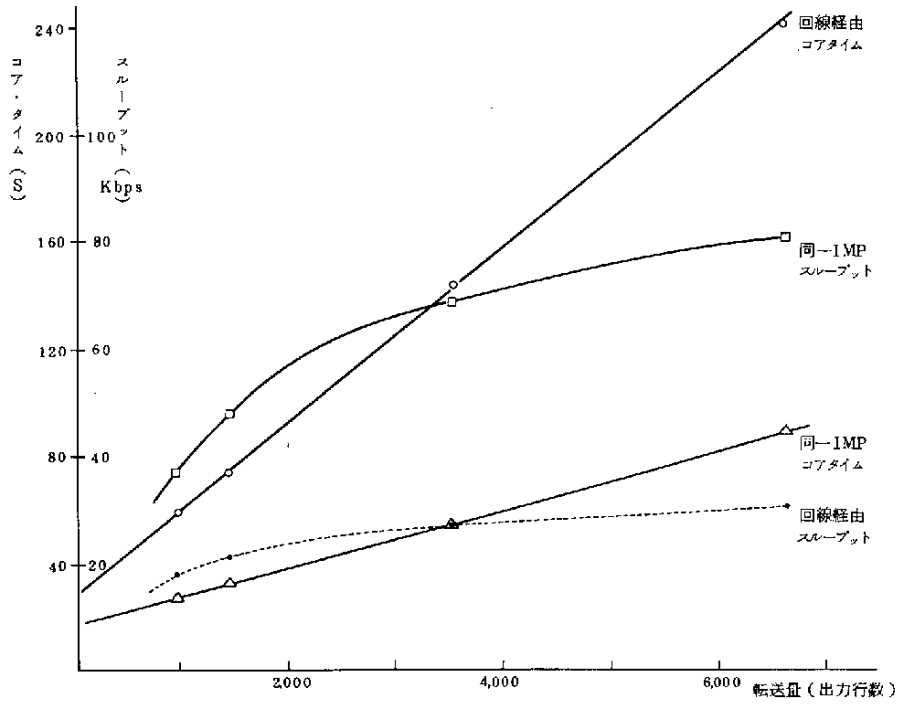


図２－４６　出力指令のスループット

図２－４５に入力指令のコア・タイムおよびスループットを、図２－４６に出力指令のコア・タイムおよびスループットを示す。

C　検討と評価

前記の測定値についての検討評価結果を以下に記す。

(1)　入力指令の測定結果について

入力指令の測定値におけるスループット、コア・タイム、使用ＣＰＵ時間について検討する。

①　スループット

入力指令のスループットは、受付ホストと処理ホストが同一ＩＭＰの場合で約２９Kbps、回線結合の場合は約２５Kbpsである。この値は一見するとかなり低いように思われるが、サーバホストであるH－８４５０側において、受取ったカード・デックをソース・ライブラリ形式でファイル化する処理およびジョブ起動用のマクロ処理にかなりの時間がかかるためである。入力指令のスループットを同様の処理を行っているＦＴＰのものと比較すると表２－１８のようになる。

表２－１８　入力指令スループットの比較

		回線結合	同一ＩＭＰ
ＦＴＰ（ソース型）		14.9 Kbps	14.8 Kbps
ＲＢＰ	ジョブ起動含む	25.1	29.3
	ジョブ起動含まず	39.4	50.7

表２－１８の比較結果により、ＲＢＰにおけるＨＩＴＡＣ側のソース・ライブラリ化のファイル処理は、ＦＴＰのインプリメント経験を通してかなり改善されたと見ることができよう。しかし、サーバＲＢＰ側（Ｈ－８４５０）におけるジョブ起動の処理に数１０秒

も費やしており、この間ユーザ・サーバ間にはデータの転送が行われていないにもかかわらず、次のＲＢＰコマンドの処理が行われない（並行処理機構がうまく働かない）結果になっているのは、インプリメント上、再検討を要する項目である。

次に、同一ＩＭＰ結合の場合と回線結合の場合とで、スループットにそれほど差がないのは、ＦＴＰの場合と同様に、サーバ・ホスト側（Ｈ－８４５０）において受取ったデータの処理が、送り側のデータ転送スピードに追従できないからであろう。

また、図２－４５のグラフにおいてスループットが直線にならないのは、転送データがゼロの時のコア・タイムすなわち入力指令の初期設定、終了処理等のための時間がかなり大きいからである。

② コア・タイム

次に、図２－４５のグラフによりコア・タイム（処理時間）の変化をみると、各測定点はほぼ直線上にあり、転送量ゼロの時の処理時間が１０秒程度になることがわかる。この時間は、前述したように入力指令に固定的に要する時間であるとみなすことができる。

この時間の内訳は、User ＲＢＰとServer ＲＢＰ と間の初期設定（コネクションの確立、オプション会話の大部分）および終了処理（終了コマンド、コネクションの切断）に要する時間が約4.5秒であり、残り約5.5秒は入力指令に要する固定的時間である。5.5秒という時間は入力指令の処理内容からするとかなり大きく、今後はこれを減少させる為に処理方法を詳細に再検討する必要があろう。

③ ＣＰＵ時間

入力指令動作を行うのに使用するＣＰＵ時間は、表２－１６の値を用いて計算すると

User ＲＢＰ（Ｆ／７５）：3.5ｍｓ／カード（８０字）

Server ＲＢＰ（Ｈ－８４５０）：７８.７ｍｓ／カード（８０字）

である。

ＦＴＰにおける同様の処理ではUser ＦＴＰ（Ｆ／７５）で約１１.８ms、Server ＦＴＰ側（Ｈ－８４５０）で約４７ms程度であるのと比較するとサーバ・ホスト側のＣＰＵ時間がかなり大きい。

これは、サーバ・ホストの処理能力がかなり小さいことと、ソース・ライブラリ型ファイルの作成およびジョブ起動の処理等にかなりのＣＰＵ時間を消費しているからであると思われる。

(3) 出力指令の測定結果について

出力指令の測定値におけるスループット、コア・タイム、使用ＣＰＵ時間について検討する。

① スループット値

出力指令のスループット値は、同一ＩＭＰの場合で約８０Kbps回線結合の場合で約３０Kbpsである。

この値は、ＦＴＰのデータ型ファイル転送におけるスループット値と比較するとほぼ等しい値を示している。ＲＢＰのインプリメント方法は、ＦＴＰの場合とそれ程異なっていないので、この結果は当然の値であると言える。ただ、図２のグラフからも明らかなように転送量が増えるにしたがってスループットが漸次増加しているのは、入力指令の場合と同様に転送データがゼロの時のコア・タイムがかなり大きいからである。

② コア・タイム

図２の出力指令のコア・タイムで転送データがゼロの時の値が、すなわち出力指令処理の固定的時間が約２５秒程を示しているのは、あまりにも大き過ぎると考えられる。入力指令の場合と同様にUser ＲＢＰとServer ＲＢＰ間の連絡に最低必要な時間約4.5秒を差引いても約２０秒を要しており、この時間についてもより詳細な検討

が必要となろう。

③　ＣＰＵ時間

出力指令動作を行うのに使用するＣＰＵ時間は、表２－１７の値より計算すると

UserＲＢＰ（Ｆ／７５）：7.3ms／行（１４０字）

ServerＲＢＰ（Ｈ－８４５０）：２５ms／行（１４０字）

である。この値はＦＴＰにおける同様の処理にUserＦＴＰ側（Ｆ／７５）で約１５ms、ServerＦＴＰ側（Ｈ－８４５０）で約３１msの時間がかかっていることをみると、ＦＴＰの場合とほぼ同程度のＣＰＵ消費時間であると考えられる。前述したようにインプリメント方法がＦＴＰとそれ程違わないことを考えた場合、これは当然の結果であると言える。

以上、ＲＢＰの入力指令と出力指令の処理効率測定結果について検討したが、インプリメント上で今後検討しなければならないのは、サーバ・ホスト（Ｈ－８４５０）におけるジョブ起動の処理方法および入力指令、出力指令の処理における固定的な処理時間の詳細な解析と減少手段の検討であろう。

3．ＮＡＰの開発

3．ＮＡＰの開発

3.1　開発の背景

独自のホスト・コンピュータは持たぬが、端末を通してネットワークを使用したいユーザのために既にＴＩＰ（Terminal Interface Message Processor）を開発した。ＴＩＰはＩＭＰ（Interface Message Processor）と同一プロセッサ内にインプリメントされ、サブネットの一部であるため、個人向けの修正や改良を行なうことは不可能である。

2.4.2で述べたように、ＴＩＰはＤＳＰ（Demand Service Protocol）などのハイレベルプロトコルをサービスするために大量のＣＰＵを消費するので、ＩＭＰが大量のトラフィックを処理している場合、ＴＩＰのサービス率が低下する。また、ＴＩＰは多種・多様な端末をサポートしており、多種類のプログラムが存在することによる安定度の低下により、サブネット自体の信頼性を低下させる原因となる。

ＴＩＰには、ディスク装置などの補助記憶装置がないので、ログオンしたユーザに対して正当性や予算などのチェックを行なうことができない。また主記憶容量の点から、多様な端末や周辺装置を多数接続するのが困難であり、ＲＢＰやＦＴＰなどのハイレベル・プロトコルの全機能のサポートすることがむつかしい。

またユーザがネットワーク中のホスト・コンピュータ・システムをアクセスする場合、ユーザはそれに対する多様なアクセス・プロトコルや手順を熟知しなければ、自由自在に使いこなすことができない。なぜならば各ホスト・システムは、イニシャル・コネクションの確立／切断、ログオン／ログオフ、アプリケーション・システム（サブシステム）の呼出しなどに対し異なった手順を採用している。そのため、特定のホスト・システムに依存しない仕事をやろうとしても、ユーザは特定のホストを選択したり、選択され

たホストに対する個別のアクセス手順やそのレスポンス・メッセージの意味などに精通しなければならない。

したがって、ネットワーク全体で共通の標準コマンドを設定し、すべてのホスト・コンピュータに対してこの標準コマンドで使用できることが望ましい。しかし、これから新規に作成されるアプリケーション用のコマンドを除き、現実には、多種多様なホストのオペレーティング・システムやアプリケーション・システムが既に存在しており、これらに対して標準コマンドをサポートするように変更や追加を行うことは、現実には困難であろう。

これに対処する例としてはNetwork Access Machine がある。これはユーザとホスト・システムの間にミニコンピュータを設置し、このミニコンピュータが、ユーザが入力した標準コマンドを受け取り、ターゲット・ホスト（Target Host）のローカル・コマンドに変換し、そのローカル・コマンドをターゲット・ホストに送り、それに対するローカル形式のレスポンス・メッセージをターゲット・ホストから受け取り、標準形式のレスポンス・メッセージに変換して、ユーザに渡すものである（図３－１参照）。

ＪIPNETに於て開発したＮＡＰ（Network Access Processor）は、前述のようなＴＩＰの欠点を改善し、ユーザがネットワーク使用を容易にするためのインテリジェンス機能を実現する手段を与えるとともにＴＩＰ以上の多種多様な端末や周辺装置の制御を可能とし、且つ異なったハイレベル・プロトコルのサポートも含めシステムの変更を容易にすることを目的としたネットワーク・アクセス用のプロセッサである。

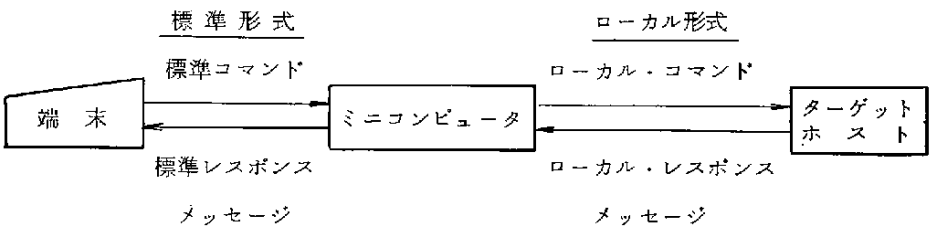


図３－１　標準コマンド／レスポンスの使用

以下に、ＮＡＰのハードウエア構成、ステーションの概念とＮＡＰコマンドの外部仕様、内部仕様、そしてリモート・ホスト—ＩＭＰプロトコルについて述べる。

3.2 ＮＡＰハードウェア構成

ＪＩＰＮＥＴのＮＡＰはＦＡＣＯＭ Ｕ−２００ 上で実現されており、そのハードウェア構成を図３－２に示す。

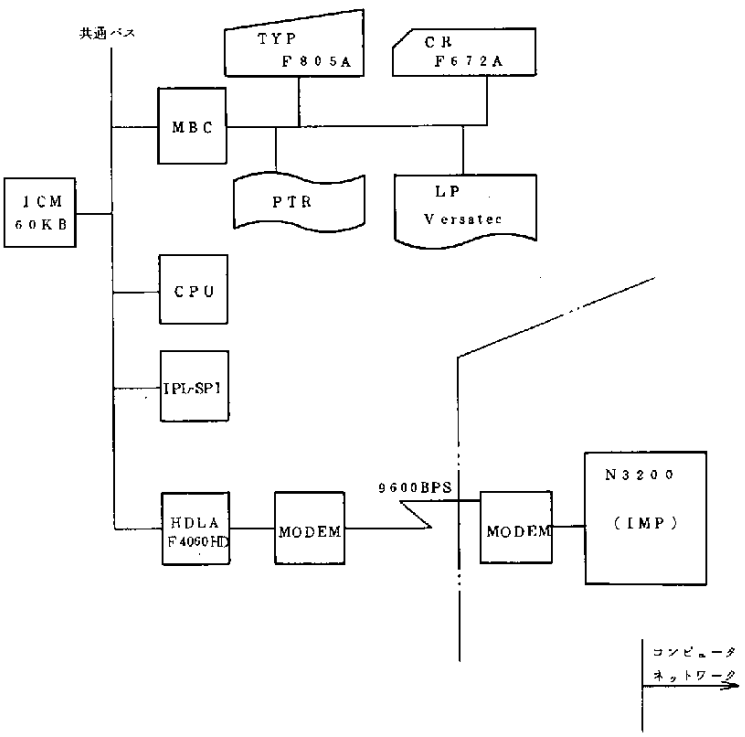


図３－２　Ｕ－２００システム構成

図にも見られるとおり、ＦＡＣＯＭ Ｕ−２００の大きな特徴は共通バスを持つことである。中央処理装置（ＣＰＵ）、主記憶装置および各種入出力装置は共通バスとインタフェースを持ち、装置間データ転送や割込み情報の転送は

すべてこのバス経由で行なわれる。また共通バス方式の採用によってアドレス体系は一本化され、各種レジスタも主記憶装置と同様に各々固有のアドレス（例えばステータスレジスタはE_{16}およびF_{16}の２番地）が割当てられており、その番地によって参照される。

本ＮＡＰシステムのハード構成を説明する。主記憶装置は６０ＫバイトのＩＣメモリである。周辺機器（デバイス）としてはカードリーダ、タイプライタ端末、紙テープリーダ、ラインプリンタが一種のマルチプレクサ・チャネルであるＭＢＣ（マルチプル・ベーシック・チャネル）を介してバスに結合されている。ＮＡＰとＩＭＰ（Ｎ３２００）は９６００ＢＰＳの全２重回線で結ばれており、この回線の伝送制御手順はＨＤＬＣである。なおＮＡＰの現バージョンでは紙テープリーダはデバイスとしてサポートしていない。

3.3 ＮＡＰ外部仕様

ＮＡＰは回線経由および直接に結合されたデバイスを持ち、これらのデバイスからＮＡＰのサブシステム（DSP、RBP etc）経由でＪＩＰＮＥＴのリソースを利用する。ＮＡＰはデバイスをステーションという単位で管理しており、これがＮＡＰのサービス単位である。

ＮＡＰのサブシステムは、ステーションと結合され、ステーションに対してハイレベル・プロトコルのサービスを提供する。ＮＡＰサブシステムとしては次のものがある。

① ＤＳＰ……User Demmand Service Protocol Process
これにより他系ホストのＴＳＳをステーションから使えるようにする。

② ＲＢＰ……User Remote Batch Protocol
これにより他系ホストのリモートバッチをステーションから使えるようにする。

③ ＮＪＣ……User Network Job Control Language Process

これによりネットワーク全体を１つの仮想ホストとしてステーションからジョブ投入をできる。

ＮＡＰサブシステムとステーションの結合は、サブシステム選択コマンド「＠ＯＰ△サブシステム名」で行なわれ、ＮＡＰサブシステムとステーションとの切断はサブシステム切断コマンド「＠ＣＬ」によって行なわれる。１つのサブシステムが１つのステーションに結合されているとき、別のサブシステムを同一ステーションに結合することはできない。

3.3.1　ステーション

ステーションはＮＡＰがデバイスを管理かつサービスする単位である。ステーションを構成するデバイスは物理的には次のものから構成される。

- １つの入出力可能デバイス
- ０あるいは１つの入力専用デバイス
- ０あるいは１つの出力専用デバイス

なお、これらのデバイスはＵ－２００と直接にチャネル経由で結合されていてもよいし、回線経由で結合されていてもよいが、現ハードウェア構成では回線経由のデバイスはない。

ステーションの物理構成は前述のとおりであるが、ＮＡＰのサブシステムは、その物理的なステーション構成を意識して制御しているのではなく論理的なステーションを対象に制御している。サブシステムが意識している論理ステーションは次のものから構成されている。

- １つのＮＶＴ（コントロール端末）
- １つの入力専用端末（ＮＶＴの入力部のみからなるデバイス）
- １つの出力専用端末（ＮＶＴの出力部のみからなるデバイス）

ＮＶＴとはNetwork Virtual Terminal の略で、ネットワークにおいて、標準となる機能をもつものとして設定されたタイプライタ様の端末装置である。

論理ステーションという概念を導入することにより、サブシステムを変更することなく、機能の大きいデバイスを用いて強力なステーションを構成することもできる。またタイプライタ端末1台でステーションを構成したとしても、リモートバッチのサービスが受けられないというようなことは起こらない（図3－3参照）。

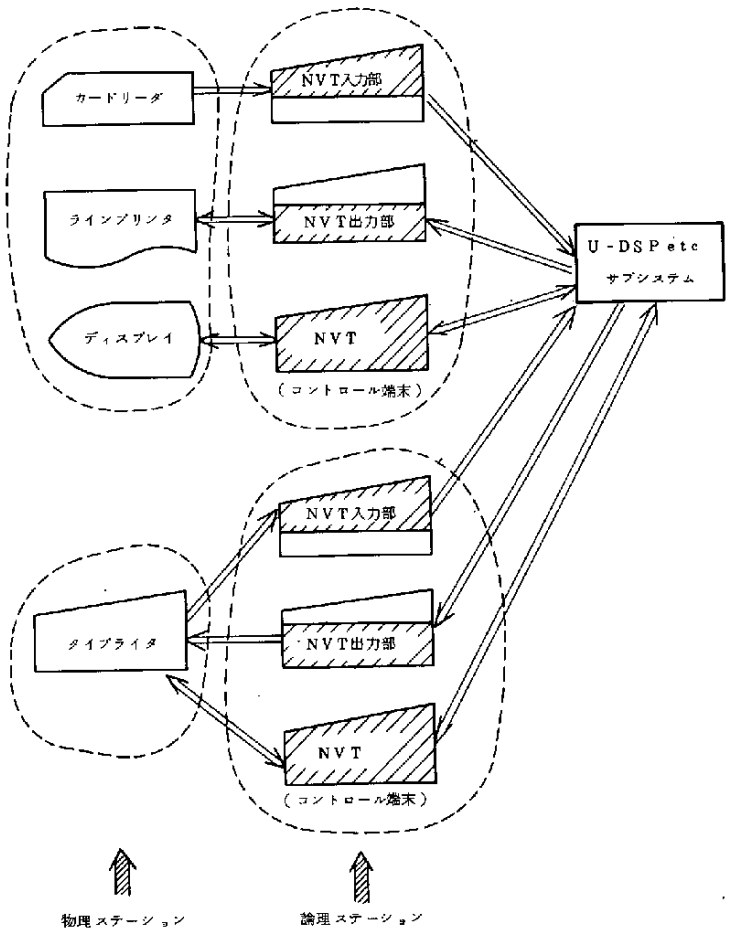


図3－3　物理ステーションと論理ステーション

ＮＡＰのステーションには一般ステーションとマスターステーションがある。マスターステーションはＮＡＰ1システムに1つだけ存在し、NAP

システム全体を管理するものであり、一般ステーションの機能もあわせ持つ。

3.3.2 NAPコマンド

NAPコマンドはステーションよりユーザがNAPに対して種々の指示を与えるのに用いコントロール端末より入力する。コマンドの種類はマスターコマンドと一般コマンドに分けられ、マスターコマンドはマスターステーションのみからしか入力できない。

A 一般コマンド

① ログオンコマンド

@ON△#u

パラメータ

#u：ユーザ番号を16進6桁で書く。

機能

ステーションログオン。該当ステーションから#uで示したユーザがNAPを使うことを宣言する。このコマンドは全てのNAPコマンドに先行してタイプインする。

② ログオフコマンド

@OF

パラメータ

なし

機能

ステーションログオフ。該当ステーションからのNAPアクセスを終了することを宣言する。

③ サブシステム選択コマンド

@OP△サブシステム名，#H〔，#i〕

パラメータ

サブシステム名：サブシステム名を3桁で書く

ＤＳＰ……ユーザＤＳＰ

ＲＢＰ……ユーザＲＢＰ

ＮＪＣ……ユーザＮＪＣＬ

＃Ｈ：相手サーバの存在するホスト番号を１６進２桁で書く。

＃ｉ：ポート識別番号を１６進２桁で書く、指定しない場合００が指定されたものとする。

機能

指定したサブシステムと結合される。以後の制御はサブシステムに任される。

④　サブシステム切断コマンド

@ＣＬ

パラメータ

なし

機能

結合中のサブシステムとの結合を解除する。以後の状態はログオンコマンドをタイプイン後と同じ状態となり、サブシステム選択コマンドを入力できる。

⑤　入出力キャンセルコマンド

@ＣＣ△$\begin{matrix}\mathrm{I}\\ \mathrm{O}\end{matrix}$

パラメータ

出力専用端末に対するものならばＯ、入力専用端末に対するものならばＩを指定する。

機能

指定されたデバイスに対して、そのとき実行中、実行待ちのＩＯリクエストを終了させる。

⑥ 割込みコマンド

@BR

パラメータ

なし

機能

結合中のサブシステムに割込む。

⑦ ステーション・ステータス表示コマンド

@ST

パラメータ

なし

機能

自ステーションの状態をコントロール端末に表示する。

B マスターコマンド

① タームコマンド

@TM

パラメータ

なし

機能

NAPのサービスをやめる。現存結合中のユーザが無くなればNAPサービスを中止する。NAPは再開待ちとなる。

② ブロードキャスト

@BC△メッセージ

パラメータ

メッセージ：同報通信メッセージ。

機能

現存ログオン中のステーションに、@OPコマンドを入れるたびに同報通信メッセージを送る。

同報通信メッセージ送信をやめるにはメッセージなしの@ＢＣコマンドをタイプすればよい。

③　フェッチコマンド

@ＦＥ△ address , length

パラメータ

address：読み出しスタートアドレスを１６進で指定する。

length　：読み出し長さを１６進で指定する。

機能

指定されたアドレス以降、指定レングスだけコントロール端末にコア内容を１６進で表示す。

④　パッチコマンド

@ＰＡ△ address , image

パラメータ

address：書き込みアドレス。

image　：書き込みイメージを１６進４桁で指定する。

機能

指定アドレスへ指定内容をコアへ書き込む。

⑤　コンソール通信コマンド

@ＣＳ△ ＃$_{H}$, message

パラメータ

＃$_{H}$：ホスト番号を１６進２桁で書く。

message：通信メッセージ。

機能

指定されたホストのコンソールへメッセージを送る。

⑥　ステータス表示コマンド

@ＤＳ

パラメータ

なし

機能

ＮＡＰ全ステーションのステータスをコントロール端末に表示する。

⑦　ステーション切断コマンド

@ＤＣ　#s

パラメータ

#s：ステーション番号。

機能

指定したステーションを強制的にログオフする。

⑧　ＮＡＰリスタートコマンド

@ＲＳ

パラメータ

なし

機能

ＮＡＰが再開まちのとき、このコマンドを入れると、ＮＡＰサービスを再開する。

3.4　ＮＡＰ内部仕様

ＮＡＰを構成するソフトウェアは、ＮＡＰの為に開発されたＮＡＰモニターおよびその管理下にあって各種機能（デバイスコントロール、サブシステムの実現etc）を遂行するタスク群と、タスク間あるいはＮＡＰモニターが参照するテーブルより構成されている。タスク間の通信は主としてＳＥＮＤ・ＲＥＣマクロによってメッセージを送受することによって行なわれる。タスク間でのテーブルの競合整理やバッファ管理の為にキューＯＮ・キューＯＦＦというマクロも用意されている。

3.4.1　ＮＡＰタスク群

ＮＡＰを構成するタスクは４つの群に分かれる。次にそれぞれの群のタスクについてその概要を示す（図３－４参照）。

A　ステーション管理系タスク群

ステーション管理系タスク群とはＮＡＰの論理ステーションを物理端末にマッピングし、サブシステムタスク（Ｕ－ＤＳＰetc）とステーションユーザ間のメッセージ交信をつかさどるタスク群である。以下にそれぞれのタスクの機能を概説する。

(1)　ステーション管理タスク

ステーション管理系タスク群の中枢であり、以下の機能を遂行する。

①　サブシステムタスクとステーション間のメッセージ・フローのコントロール。

②　論理ステーションの物理ステーションへのマッピング、その他のステーション管理。

③　ステーションから投入されたＮＡＰコマンドの処理。

④　ＮＡＰシステム全体の管理。

(2)　ＴＣＰ１～３

ステーション管理からの要請を受け、実際のデバイスに対する入出力を行なう。ＮＶＴと実デバイスの変換はこれらのタスクで行なう。ＴＣＰは各デバイスタイプごとに１つあり、ＴＣＰ１～３はそれぞれタイプライタ端末、ラインプリンタ、カードリーダをサポートする。

B　ＮＣＰ系タスク群

ＮＣＰ系タスク群とはサブシステムタスクに他系ホストのプロセスとのプロセス間通信機能を提供する為のタスク群であり、以下にそれぞれのタスクの機能を概説する。

(1)　ＮＣＰタスク

ＮＣＰ系タスク群の中枢であり、ホスト―ホストプロトコルに従っ

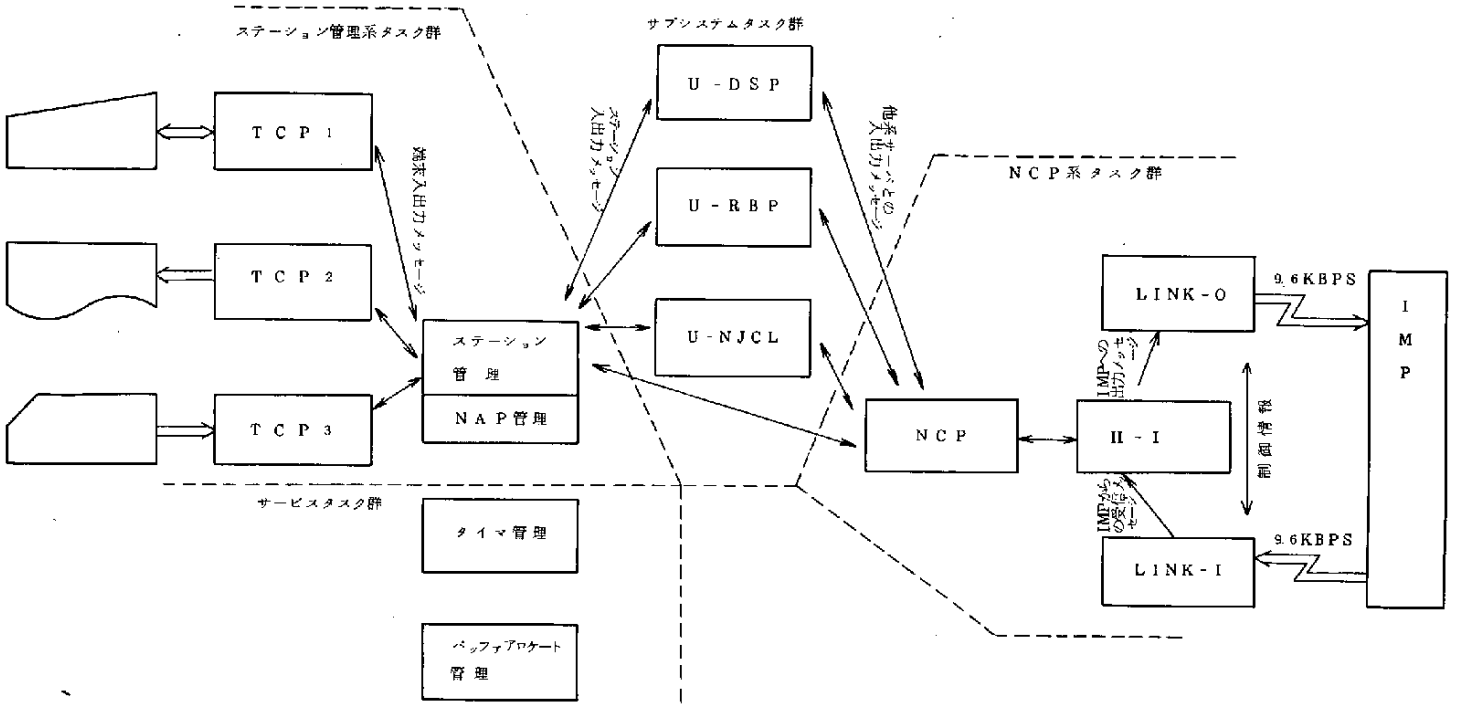


図 3 — 4　タスクおよびデータ・フロー　（タイマ管理、バッファアロケート管理は他の全てのタスクと通信する。またタスク間の通信はキュー経由で行なわれている。）

て他系ホストのプロセスと自系サブシステム間のメッセージ通信を制御する。

(2) Ｈ－Ｉタスク

リモート・ホスト―ＩＭＰプロトコル（後述）に従ってＩＭＰとのメッセージフローを制御する。

(3) ＬＩＮＫ－Ｏタスク

ＨＤＬＣ手順に従ってＨ－Ｉタスクより受け取ったＩＭＰあてメッセージを送信する。

(4) ＬＩＮＫ－Ｉタスク

ＨＤＬＣ手順に従ってＩＭＰより受け取ったメッセージをＨ－Ｉタスクに渡す。

C サービスタスク群

サービスタスク群とは、各タスクが動作する上で必要な時間情報、バッファなどの提供サービスを行なうもので、現在次の２タスクが用意されている。

(1) タイマ管理タスク

ＮＡＰを構成する各タスクからの要請を受け、一定時間ごとにタイマ情報を各タスクに送る。

(2) バッファアロケート管理タスク

各タスクよりの要請によりバッファを確保し、確保し終ったら要求タスクにバッファを渡す。このタスクはバッファ確保待ちで待ち状態に入れないタスクの為に用意されているものである。

D サブシステムタスク群

サブシステムを実現するタスク群で、ステーションよりの@ＯＰコマンドでステーションと結合され、ステーションユーザにサブシステムサービスを行なう。

(1) U－DSPタスク

DSPプロトコルに従い、ステーションから他系のTSSを使えるようにする。

(2) U－RBPタスク

RBPプロトコルに従い、ステーションから他系のリモートバッチを使えるようにする。

(3) U－NJCLタスク

NJCLプロトコルに従い、ネットワーク全体を1つの仮想コンピュータとしてステーションからジョブを投入することを可能にする。その詳細は第6章で述べる。

3.4.2 NAPモニター

NAPモニターはU－200ハード上に、マルチタスキング構造を実現するものである。タスク間の通信と同期はメッセージのキューイングとセマフォーによって行なっている。

A モニター構造

NAPモニターは次の4部分からなっている。

① イニシエータ

各種テーブルの初期設定等、NAPシステムのイニシャライズを行ない、終了後CPUディスパッチャーに制御を渡す。

② SVCモジュール

各タスク、割込みハンドラー間の通信・同期を行なうためのSEND・RECマクロ、キューON・キューOFFマクロを実行する。このモジュールからのreturnには次の2種類がある。

(a) マクロの呼び出し元に直接returnする。

(b) ディスパッチャー経由でreturnする。

③　外部割込みハンドラー

外部割込みが発生したとき、制御が渡され、割込みに対応した処理（タスク起動、Ｉ／Ｏの後処理etc）終了後ディスパッチャーに制御を渡す。

割込み処理はスタック方式で処理され、低レベルの割込み処理中に高レベルの割込みが発生したときは、低レベルの処理は中断され（プッシュダウン）、高レベルの割込み処理がなされたのち、低レベルを再開（ポップアップ）する。現在割込みレベルとしては４レベルある。またハンドラーではタスクに情報を渡すのに、タスクと同様ＳＶＣマクロを利用している。

④　ＣＰＵディスパッチャー

各タスクに対するＣＰＵの割当てを行なう。ディスパッチャーは４つのディスパッチキューを持ち、プライオリティの高いキューのタスクからファーストイン・ファーストアウトでＣＰＵを与える。

図３－５にタスクとモニターの制御および情報の流れを書いたものを示す。

B　バッファ

ＮＡＰでタスク間の通信などに用いるバッファにはコマンド・バッファとメッセージ・バッファと呼ばれる２種類のものが用意されている。両者とも固定長（前者は１要素：２４バイト、後者は160バイト）でありバッファプールに各々数１０個プールしておき、各タスクで必要に応じて確保し、不要になったらプールに返却する。コマンド・バッファとメッセージ・バッファの相違はその用途にあって、前者はタスク間での制御情報のやりとりに用い、後者は通信データをセットするのに用いる。

C　タスク間通信とマクロ

タスク間の情報交換や同期をとる為に２組のＳＶＣマクロが用意されている。

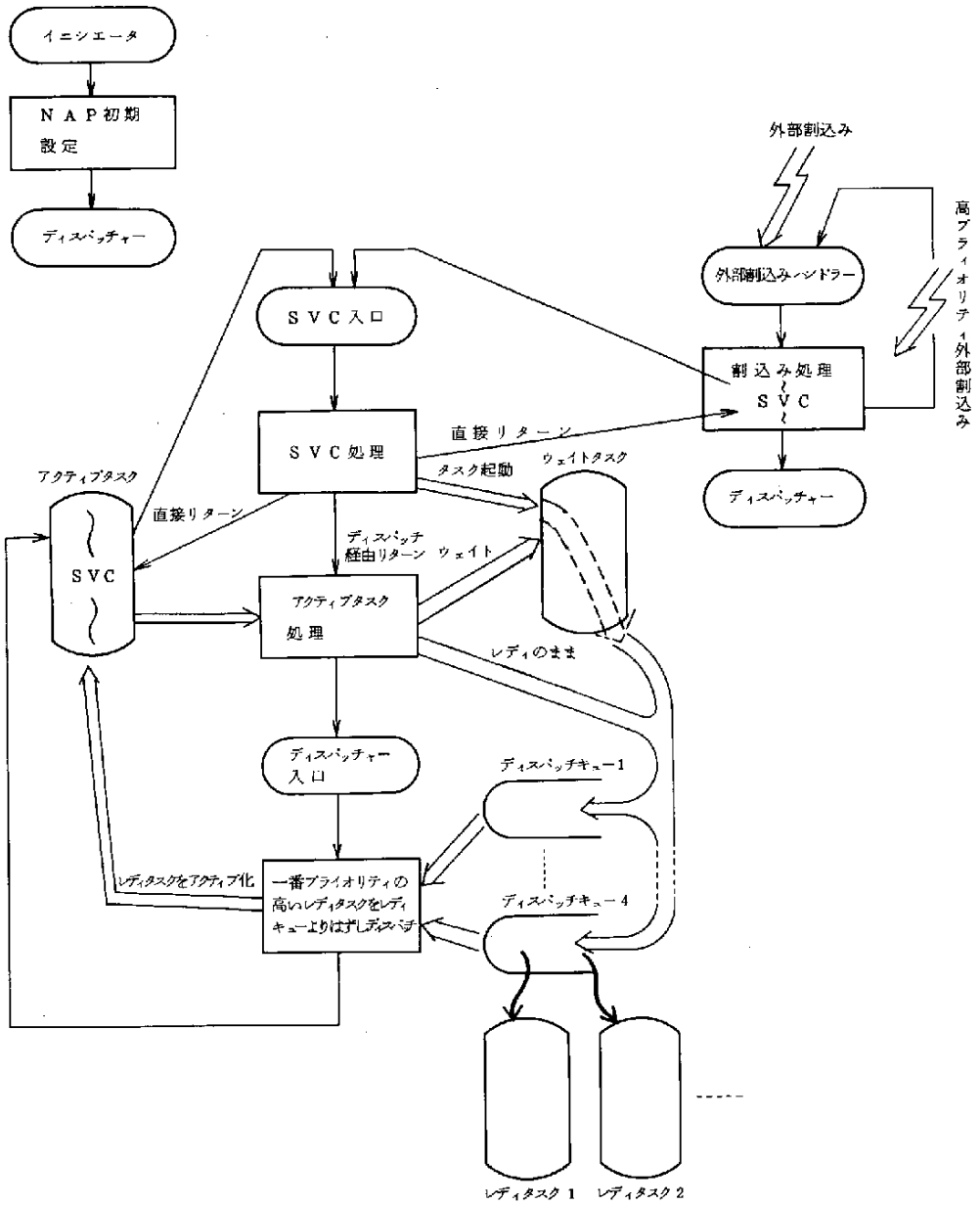


図３－５　タスク・モニター関連図（←制御の流れ、⇦ステータス・情報の流れ）

(1)　ＳＥＮＤ・ＲＥＣマクロ

ＮＡＰの各タスクはそれぞれ４個ずつ個有の入力キューを持ってい

る。SEND・REC マクロはそのキューを介してタスク間でメッセージを送受することによりタスク間の情報交換を行なうためのものである。

あるタスクが別のタスクにメッセージを送りたいときは、相手タスク番号とキューの番号を指定してＳＥＮＤマクロを発する。メッセージが指定したキューに乗せられると制御は直ちに発信タスクに戻ってくる。

他タスクからのメッセージを受け取りたいタスクは、メッセージを受けたいキューを指定して（指定により全てのキューを対象とすることもできる）ＲＥＣマクロを発する。指定したキューにメッセージがあれば、それを１個キューよりはずしタスクにリターンする。なければメッセージが送られるまでタスクはウエイトする（図３－６参照）。なおメッセージがないときに直ちにタスクに制御を戻すよう指定することもできる。

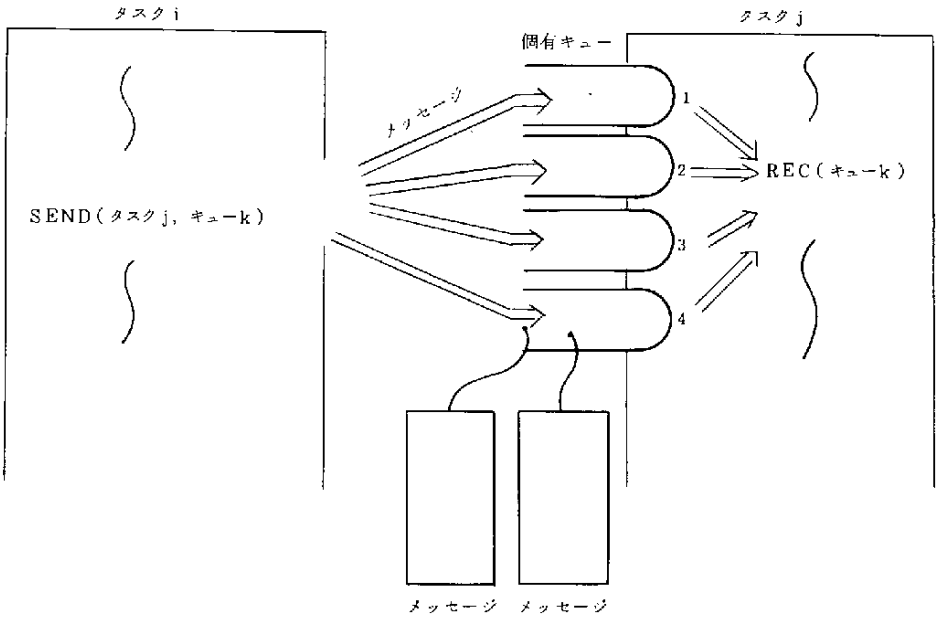


図３－６　ＳＥＮＤ・ＲＥＣマクロによるタスク間通信（ｋ：１～４）

(2) キューＯＮ・キューＯＦＦマクロ

ＮＡＰモニターでは、コアリソースの管理やタスクの同期をとるためにキューセマフォーを用いることとし、それを取り扱うためのマクロがキューＯＮ・キューＯＦＦマクロである。

キューセマフォーとは図３－７の構造を持つものである。以下に各項目の説明を行なう。

① セマフォー

正の値のとき：空きリソースの数を示す。

負の値のとき：リソース待ちのタスクの数を示す。

② リソース chain head・tail

セマフォーが正で head が正の値のときは最初のリソースへのポインタ。

セマフォーが正で head が０のときは、リソースの実体が無いこと（セマフォーカウントだけが増減する）を示す。

tail は最後のリソースへのポインターである。

③ リソース待ちタスク chain head・tail

セマフォーが負の値のとき、リソース確保待ちの最初および最後のタスクのＴＣＢ（Task Control Block：タスクを制御する為の情報がセットされている領域）へのポインターが入っている。

セマフォー
リソース chain head
リソース chain tail
リソース待ちタスク chain head
リソース待ちタスク chain tail

図３－７　キューセマフォー

ＮＡＰの空きバッファ管理（バッファプールの管理）にはキューセマフォーが用いられている。図３－８は空きバッファが２個あることを示している。バッファを確保したいタスクはバッファ管理用キューセマフォーと欲しいリソース数（バッファ数）を指定してキューＯＦＦマクロを発する。もし空きリソースが要求されただけなかったならば図３－９のようにタスクは要求が満たされるまでウエイトに入る。しかる後、別のタスクがキューＯＮマクロでバッファを返却し、ウエイト中のタスクの要求が満たされると、ウエイト中のタスクは再びレディとなる。

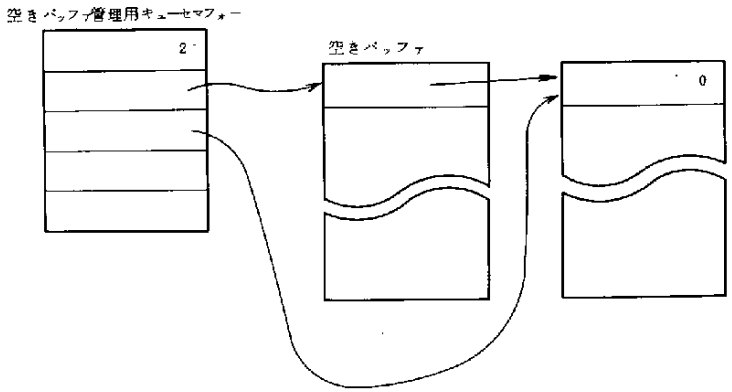


図３－８　空きバッファが２個ある例

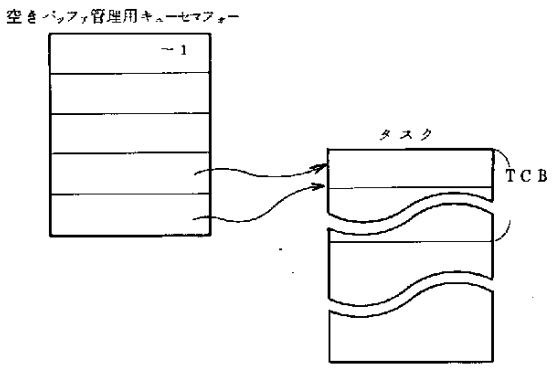


図３－９　リソース確保待ちタスクが１個ある例

また２つ以上のタスクが１つのテーブルを共用するとき排他制御をしたいならば、キューセマフォーとテーブルをペアにしておき（キューセマフォーのセマフォーは１、リソースチェインは不用）、各タスクでこのテーブルをさわるとき、次に示す手順で行なえばよい（図３－１０参照）。

① キュー０ＦＦ発信……テーブル確保

② テーブルアクセス

③ キュー０Ｎ発信……テーブル解放

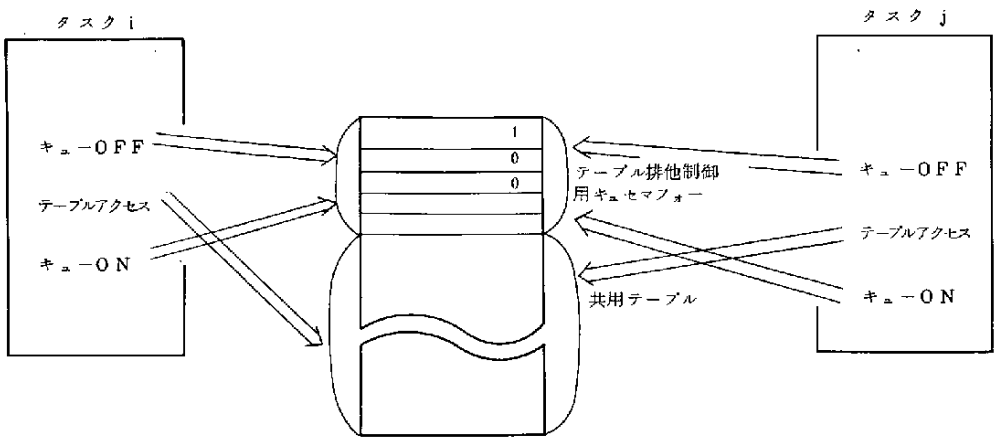


図３－１０　キュー０Ｎ・キュー０ＦＦによるテーブル排他制御の例

以上のべたキュー０Ｎ・キュー０ＦＦマクロの用法はＮＡＰシステム内で多数用いられている。

3.5　リモート・ホスト－ＩＭＰプロトコル

本プロトコルは高速通信回線で結合されたホスト－ＩＭＰ間のデータ転送手順について規定するものであり、チャネル結合ホスト－ＩＭＰプロトコルをその土台としている。

リモート・ホスト—ＩＭＰ間のデータ転送の一番ベーシックなレベルはホストを１次局、ＩＭＰを２次局としたＨＤＬＣダブルナンバリングによって行ない、そのインフォーメーションフレームを介してホスト—ＩＭＰ間のフローコントロール情報・データ本体を送受する（ホスト－ＩＭＰプロトコルレベル）、２層構造よりなり立っている（図３－１１参照）。

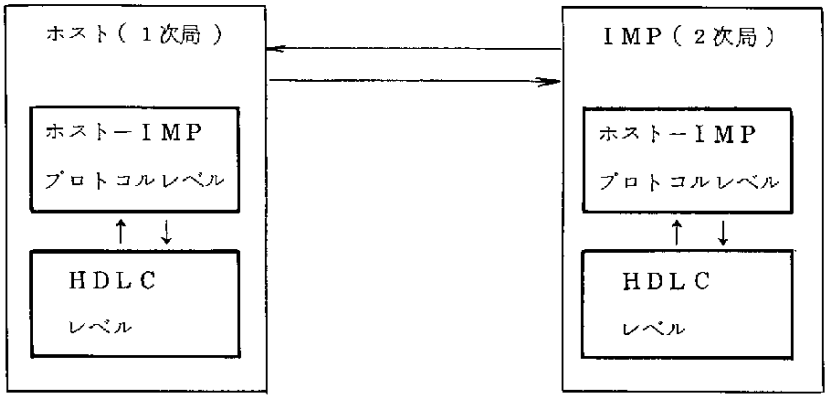


図３－１１　プロトコル構造

3.5.1　データ形式

ホスト—ＩＭＰ間でやりとりされるデータ（ホスト—ＩＭＰプロトコルレベル）は全てインフォーメーションフレームとして送受され、その内容として

①　制御情報（リーダー）のみ

②　制御情報＋一般データ部……メッセージ種類０のみ

の２種類があり、そのレイアウトを図３－１２に示す。

制御情報は４バイトで構成され全てのインフォーメーションフレームに存在し、プロトコルに規定された種々の制御情報を含む（図3－13参照）。一般データ部はユーザプロセスが送信したメッセージ（最大１１５２バイト）あるいはホスト—ホスト間での通信情報等が入り、最大144バイトからなる。なおメッセージが１つのインフォーメーションフレームで送信できな

いとき（144バイトを越えるとき）は複数個（2～8）のフレームとして送られる。その場合最後のフレームのみがトランケートされたバイト数のデータを含む。

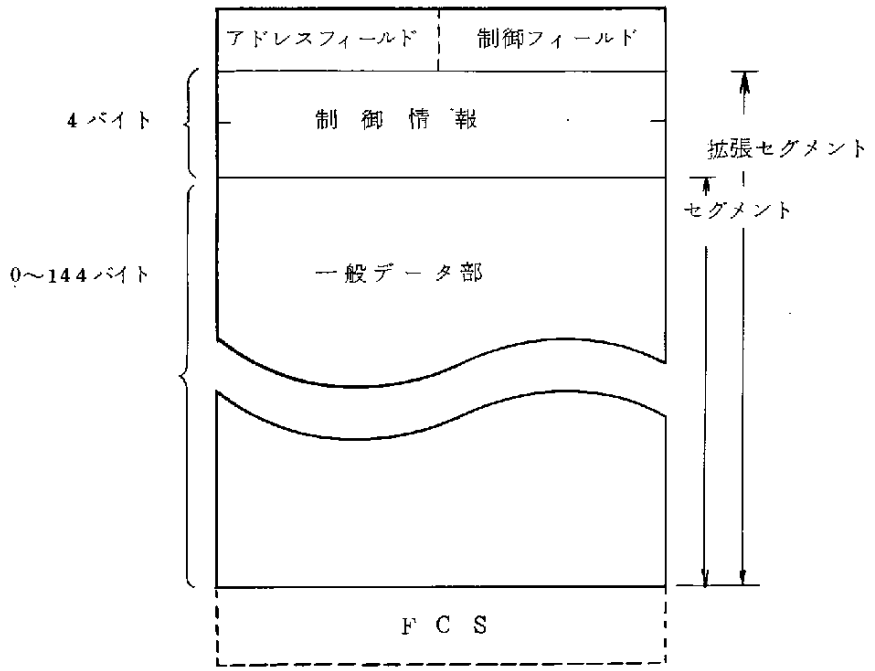
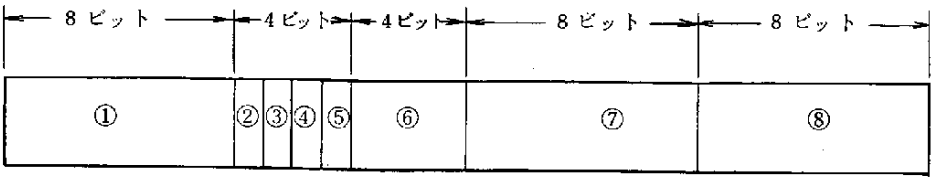


図3－12　インフォーメーションフレームのレイアウト



① メッセージ種類に対応した情報（８ビット）

⑥のメッセージ種類により使用法が異なる。（表３－３）参照のこと。

② ＯＫ、リトランス応答（１ビット）

ＯＫ、リトランスミット（＝12、＝13）の応答としてホストがＩＭＰに送るロングメッセージのセグメントであることを示す。（メッセージ種類＝０の時のみ有効）

③ コネクション種類（１ビット）

１のときこのコネクションは多重パケットメッセージを送り出すことを示す。このビットはコネクション確立時の最大メッセージサイズにより、メッセージの転送ごとにセットされる。

すなわち、最大メッセージ長が複数セグメントになる場合には、たとえ転送メッセージが１セグメントであっても１にされる。

④ 未使用（１ビット）

⑤ 未使用（１ビット）

常に０とする。

⑥ メッセージ種類（４ビット）

制御情報が何を意味するかを示す。（表３－１）、（表３－２）を参照のこと。

⑦ ホスト番号（８ビット）

ホスト→ＩＭＰ制御情報の場合には、目的地ホスト番号を示す。

ＩＭＰ→ホスト制御情報の場合には、発信地ホスト番号を示す。

⑧ コネクション番号（８ビット）

プロセス間コミュニケーションにおいてＮＣＰがプロセス中のｐｏｒｔを識別するための番号。

図３－１３　制御情報

表３－１　ＩＭＰ→ホスト制御情報

メッセージ種類	意　　味
0	通常メッセージ送信要求（Regular message） 他ホストより、そのホスト宛の一般データを転送する要求があった時、プレフィックスとして使用される。制御情報に続いて、データ部が存在する。
1～2	未　使　用
3	データ受信完了（RECEIVE） 他ホストに発信したメッセージが正常に送られたことをＩＭＰが確認した時、発信する。
4～11	未　使　用
12	複数セグメント受け入れ準備完了 ホスト側から複数セグメントの転送要求があった場合に、受け入れ準備ができたら、これを発信する。これによりホスト側から第１拡張セグメントから最後の拡張セグメントまでの１メッセージがインフォメーションフレームとして送られてくる。
13	メッセージ再送要求 サブネットの障害などでパケットが消えた場合に、ホストに対してメッセージの要求をする。これを受け取ったホストはRECEIVE まちのメッセージを再送しなければならない。この応答は、多重パケットメッセージの場合だけ発信される。

表３－２　ホスト→ＩＭＰ制御情報

メッセージ種類	意　　味
0	通常メッセージ送信要求（Regular message） 他ホストに一般データを送信したい場合にプレフィックスとして使用される。１セグメント・メッセージの場合には、全コネクションを通してRECEIVEまちの１セグメントメッセージが、４未満ならば、直ちに送信される。複数セグメント・メッセージの場合には、ＩＭＰ→ホスト制御情報の複数セグメント受入れ準備完了を受け取ると、メッセージを転送する。
１～２	未　使　用
3	データ受信完了（RECEIVE） 目的地ホストがメッセージを受け取った時に発信する。
４～15	未　使　用

表３－３　制御情報のメッセージ種類に対応した情報部分の用途

メッセージ種類	メッセージ種類に対応した情報の用途
0	送信メッセージのセグメント数（１～８）
１～２	未　使　用
3	RECEIVEの状態を数字で入れる。（下位４bit） ＝１の時正常受信（上位４bitをALLOC カウンターとして用いる） ＝２の時　バッファ余裕なし　discard. ＝４の時　コネクション　close discard. ＝５の時　目的地ＩＭＰ　disconnect or ホスト down. ＝６の時　目的地ＩＭＰなし
４～15	未　使　用

3.5.2　初期状態処理手順

ホストーＩＭＰの初期状態処理手順はＨＤＬＣのＳＡＲＭ、ＵＡコマンドの送受によって開始される。

ホスト側

送受信回線をレデイにし、ＳＡＲＭコマンド（ＨＤＬＣ）を送りＩＭＰよりのＵＡコマンド（ＨＤＬＣ)を待つ。ＵＡコマンドを受信したらＩＭＰと結合されたものとする。

ＩＭＰ側

送受信回線をレデイにし、ホストよりＳＡＲＭコマンドが送られてくるのを待つ。ＳＡＲＭコマンドを受信したらＵＡコマンドを送信しホストと結合されたものとする。もしＳＡＲＭコマンドを受信したときＩＭＰがレデイになりたくなければＤＭコマンド（ＨＤＬＣ)を返す。ＤＭコマンドを受け取ったホストは、一定時間後再び初期状態処理手順に入る。

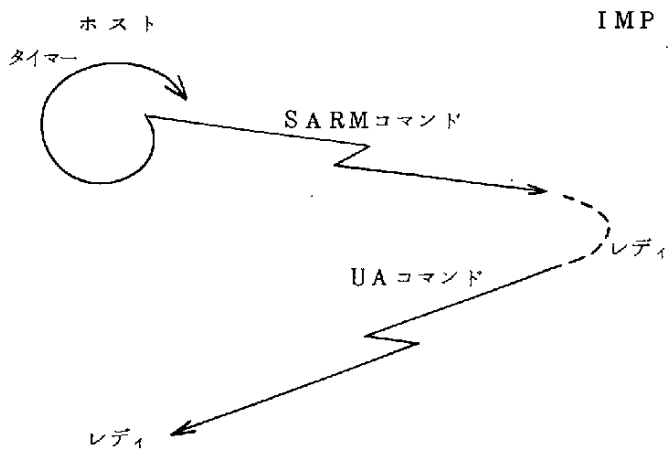


図３－１４　初　期　手　順

3.5.3　データ転送手順

ホスト—ＩＭＰ間で転送されるデータ（インフォーメーションフレームとして送られる）にはホスト—ＩＭＰ間であらかじめ決められた制御情報のみを含むものと、制御情報と一般データ部とを含むものがある。後者はホスト—ホスト間の通信、ホスト内のプロセス間の通信メッセージを含むものであり。メッセージの長さによって単一セグメント・メッセージと複数セグメントメッセージに分けられ、それぞれホスト—ＩＭＰ間での送受の手順が異なる。

(1)　単一セグメントメッセージの送信手順

①　ホスト　→　ＩＭＰ

既にＩＭＰに送りRECEIVEを受け取ってない単一セグメントメッセージの総数が4未満ならば直ちに、4ならばRECEIVEを受け取ってからメッセージを拡張セグメントの形でＩＭＰへ送信する（図３－１５参照）。

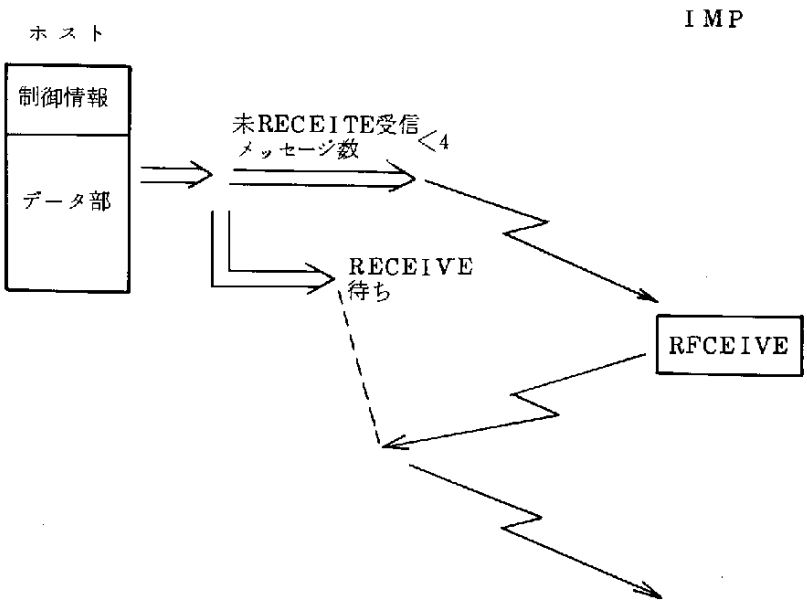


図３－１５　単一セグメントメッセージ送信手順（Ｈ→Ｉ）

② ホスト ← IMP

拡張セグメントの形でホストへ直ちに送信する。メッセージを受け取ったホストはできるだけはやくRECEIVEをIMPに送る（図3－16参照）。

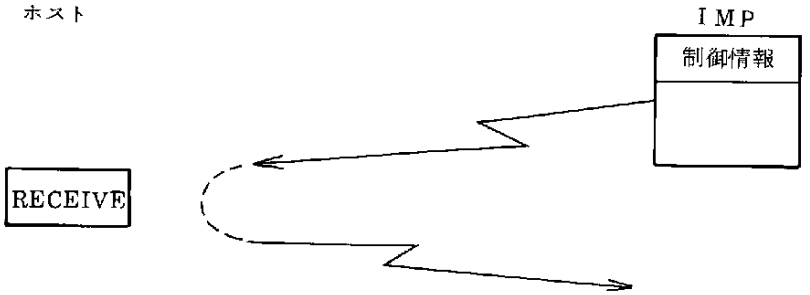


図3－16　単一セグメントメッセージ送信手順（H←I）

(2) 複数セグメントメッセージの送信手順

① ホスト → IMP

制御情報（メッセージ種類＝0）のみからなる拡張セグメントをIMPに送り、IMPから対応する制御情報（メッセージ種類＝12）が送られてくるのを待つ。なおこの間に他のメッセージを随意に送ってよい。ただし複数メッセージの制御情報は32個までとする。

IMPから対応する制御情報が送られてきたならば、ただちにそのメッセージを構成する拡張セグメントを順番に第1番目から連続に送る（図3－17参照）。

なお複数セグメントメッセージはIMPより再送要求される（メッセージ種類＝13の制御情報をIMPより送られる）かもわからないのでRECEIVEを受け取るまで保持しておく。再送要求された場合もメッセージ種類＝12を受け取ったと同様にメッセージを構成する拡張セグメントを順番に第1番目から連続に送る。

② ホスト ← IMP

メッセージを構成する拡張セグメントを順番に第1番目から連続に送

る。メッセージを構成する全セグメントを受け取ったならばホストはできるだけ早くそのRECEIVEをＩＭＰへ送信する（図３－１８参照）。

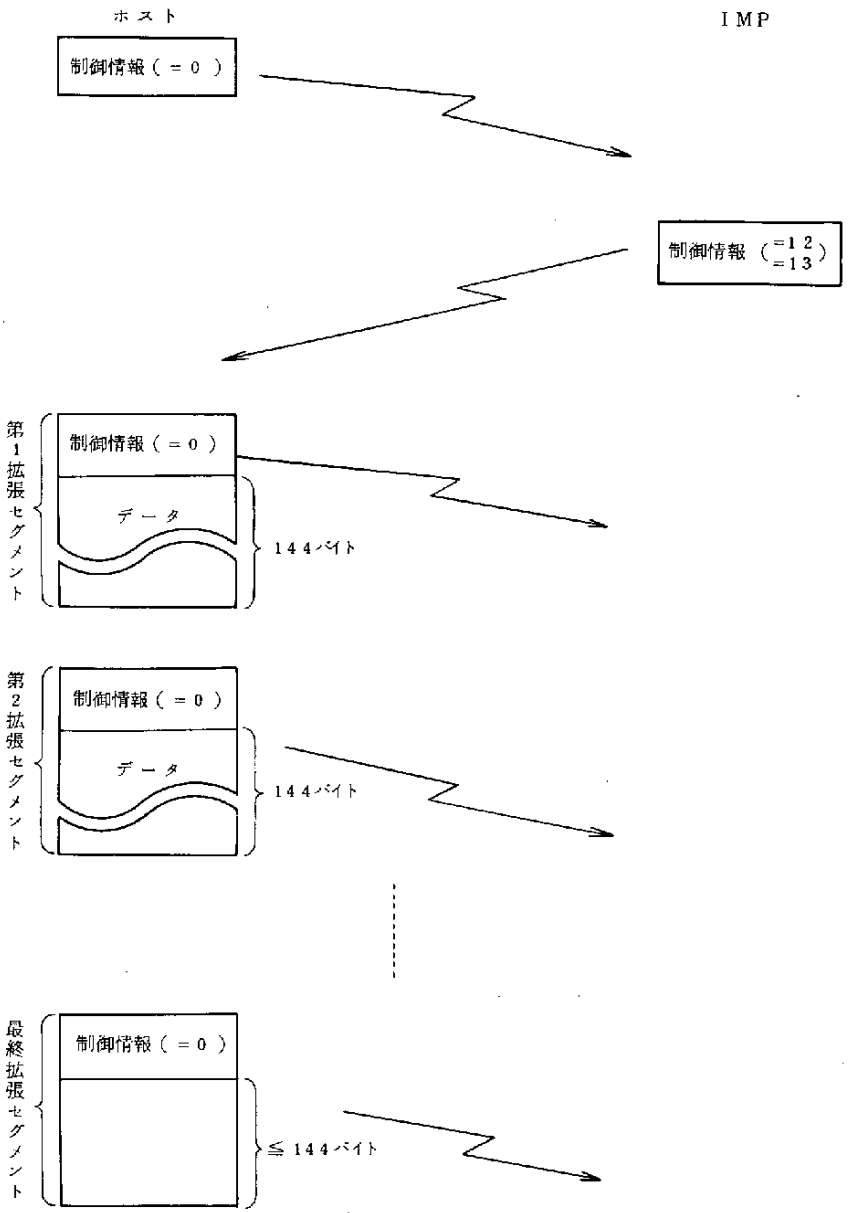


図３－１７　複数セグメントメッセージ送信手順（Ｈ→Ｉ）

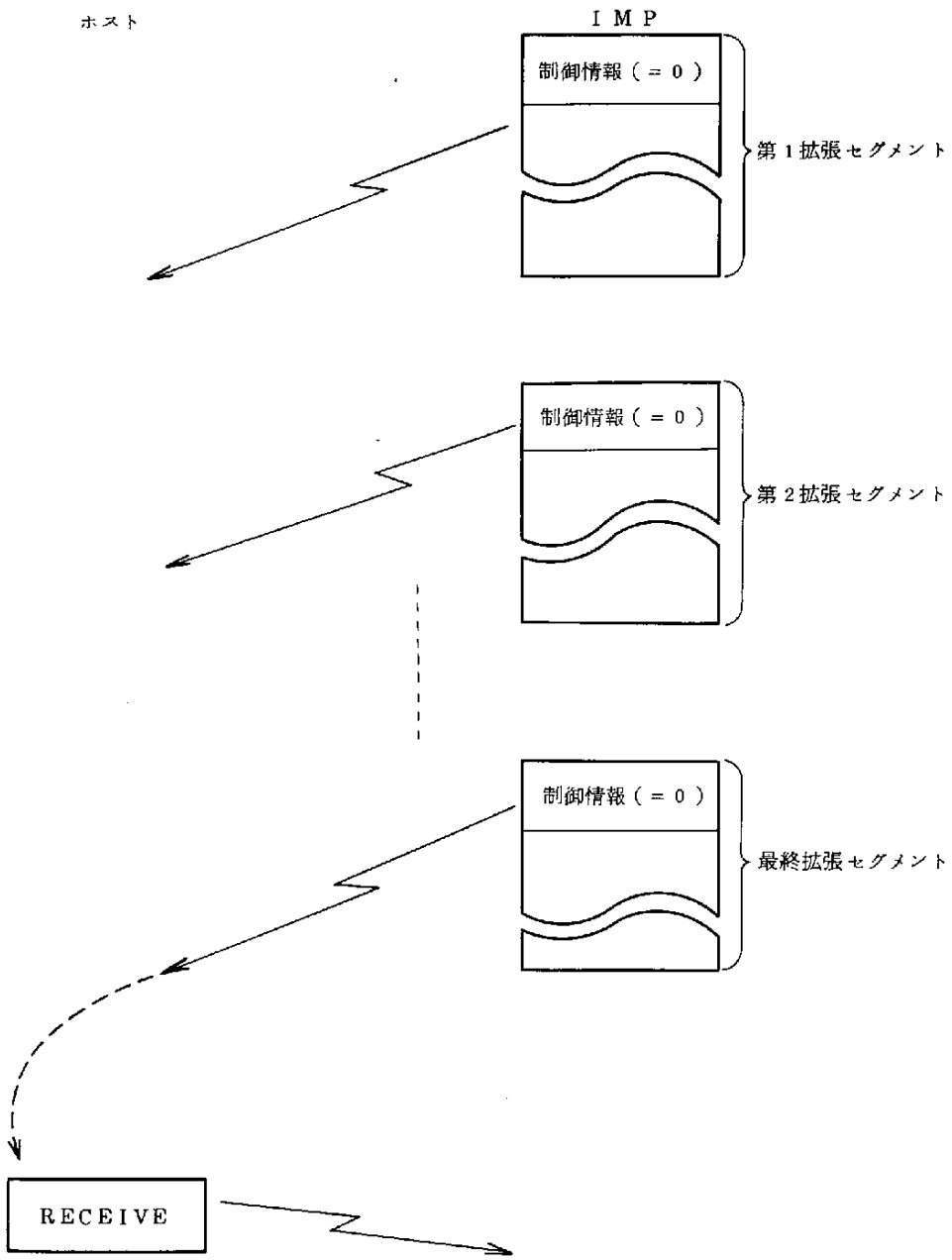


図３－１８　複数セグメントメッセージ送信手順（Ｈ←Ｉ）

3.5.4　終了処理手順

①　正常終了処理手順

ホストよりＤＩＳＣコマンド（HDLC）を送りＩＭＰよりＵＡコマンドで答える（図３－１９参照）

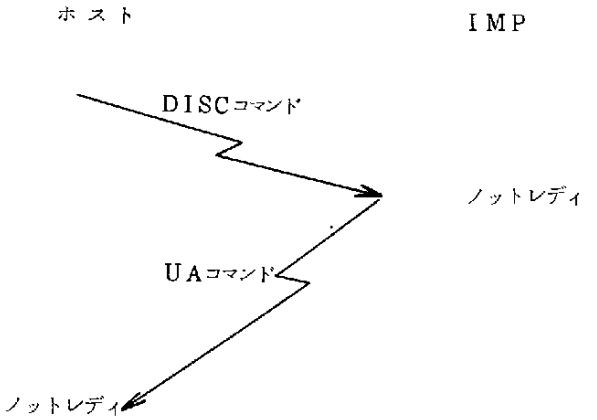


図３－１９　正常終了手順図

②　異常終了処理手順－１

ＩＭＰがホストとの接続を自分から切断したいときに用い、ＩＭＰよりＲＤコマンド（HDLC）を送る。ＲＤを受け取ったホストは①の手順をふむ（図３－２０参照）。

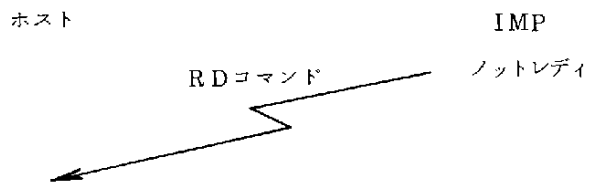


図３－２０　異常終了手順図

③　異常終了処理手順－２

ハードエラー発生の場合は互に初期手順に入る。

4．ＮＦＥＰの開発

4．ＮＦＥＰの開発

コンピュータ・ネットワークの有効性は、参加するリソースの数と大いに関係がある。すなわち、有用なリソースを持つホストの参加は、コンピュータ・ネットワークの価値の向上をうながす。このためホストコンピュータをどのような形でネットワークに参加させるべきかが問題であり、できるだけ多くのホストが容易にコンピュータ・ネットワークへ参加できるような技術の開発が必要となる。

NFEP（Network Front End Processor ）はこの様なニーズに適応すべく考えられたもので、ホスト・コンピュータとサブネットの間に存在し、コンピュータ・ネットワークに参加する為の処理のほとんどを行なうものである。NFEPはホストとの間で複数の論理的なリンクによって結合され、このリンクを通じてプロセス間通信を始めとして、ファイル転送、ＴＳＳ、リモードバッチその他の高位のプロトコルをサービスする事を目的とする。この目的実現のために、NFEP内においては、NCP 、サブネットとの通信、ホストとの通信を基礎として、プロセス間通信、高位プロトコル処理機能の大部分を持ち、ホストの負荷の軽減をはかるものである。本章ではＮＦＥＰ が考えられた背景及び有効性について論じ、ＮＦＥＰ の機能について概説する。最後にある特定のホストを対象としたＮＦＥＰの論理設計を述べる。

4.1　ＮＦＥＰ開発の背景

ホスト・コンピュータがネットワークに参加する形態としては次のものが考えられる。

① サブネットとホスト・コンピュータをチャネル結合する。

② サブネットとホスト・コンピュータを回線結合する。

③ マイクロ・コンピュータを用いてサブネットと結合する。

④　ＦＥＰを経由してサブネットに結合する。

⑤　ＴＩＰと同様な端末制御を行ないかつ、ホスト特有のインターフェースに対しての結合を行なう。

コンピュータ・ネットワークはホストとサブネットから構成されるという考え方がＡＲＰＡ ネットワーク以降の主流をなしていた。①②がその代表であり、ＵＣＳＤのＢ６７００のような③の形態も出現した。その後、ホスト・コンピュータとサブネットの間にミニコンピュータを導入した④の形態のシステムがＮＡＳＡ－ＡＭＥＳ のＩＬＬＩＡＣ－ＩＶ、ＲＡＮＤのＩＢＭ 370／158等で出現し、さらにＡＮＴＳでは⑤の形態で接続されている。この様に、ホストとサブネットの間に何らかのコンピュータを介してネットワークにアクセスする事が考えられて来た背景には以下の理由がある。

①　ミニコンの価格低下

ミニコンのハードウェア購入料金が非常に安くなってきており、しかも性能の向上がみられる。この為大型ホストからサテライトへの機能移行が採算のとれるものとなっている。

②　運用時のコスト

ホストのＯＳにかかる負荷（ＣＰＵ、コア ）をできるだけ少なくしてネットワークと結合する事が望まれる。また、ＮＣＰ、高位プロトコルのホストの負担をユーザに対するコンピュータ使用料金として計算すると、ユーザがネットワークを通じてコンピュータを利用する時非常に高価なものとなってしまう。

③　インプリメント時のコスト

ホスト内に内蔵する形態のＮＣＰ、高位プロトコル処理等は、ＯＳとのインターフェース部分が非常に複雑で、その調査、及び作成に期間とコストがかかる。

④　標準システムの作成

ＮＣＰ、高位プロトコル処理等のかなりの部分は標準的なソフトウェア

として整備する事が可能である。これらのものをホスト内に持てば、ホスト特有の処理が介在するが、ミニコンに持たせれば、ローカルホストとのインターフェース部分だけの検討で、ネットワークへの参加が可能となる。

⑤ リモート・ターミナル化

ホストとサブネットの間にミニコンを置く事で、遠隔地のホストからも容易にネットワークへ参加できる。

以上の事よりサブネットとホストの間のインターフェースの有効性が指摘され、サブネット、ホスト・サブネットインターフェース、ホストの3要素間で機能分担をすることにより、充来のサブネット、ホストの機能分担よりもソフトウェアの作成の容易さ、コストパフォーマンスの向上、ホストコンピュータの負荷の低減が期待でき、ネットワーク全体からみた時有益である事が推測される。図4－1に充来の機能分担と、インターフェース部分を考えたネットワークの機能分担を示す。図4－1の斜線部分は、ＮＣＰ、高位プロトコル処理に要する領域である。

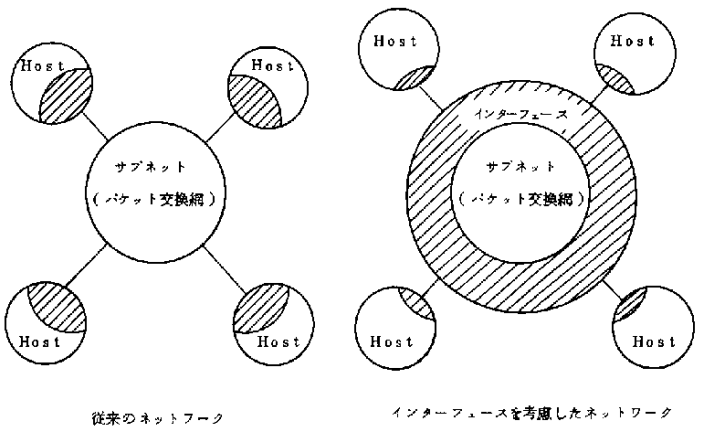


図4－1　ネットワーク機能

ホストとサブネット間のインターフェースをＮＦＥＰとした時考えられる利点は次の通りである。

① ホスト側の既存ＯＳとの関係を少なくする事により、ＮＦＥＰインプリ

メント時のＯＳの調査及び作成コストを減少することができる。

② ＮＦＥＰ内にＮＣＰ、高位プロトコルサービスのソフトウェアを持ち込み、このソフトウェアを共通にする（標準化）ことができるので、ホストをネットワークに参加させる際のコストを減少することができる。

③ ホスト側のリモート・サポート機能のエントリを利用する事により、既存のオンライン・アプリケーションをネットワーク下で利用できる。

④ ＮＣＰ、高位プロトコルサービスのソフトウェアがホスト内に存在しないので、ホストのＣＰＵ、コアのオーバヘッドが減少する。

ＪＩＰＮＥＴ の開発と評価より、コンピュータ・ネットワークのソフトウェアは次のような条件を満たす必要があることが判明した。

- ホスト・コンピュータの持つ標準的なＯＳに対する機能追加、修正などは少ない事が望ましい。
- ホスト・コンピュータがネットワークのために使用するリソース（ＣＰＵ、コア、チャネル）が予想外に大きく、ユーザにかかるネットワークコストも大きくなっている。このためできる限りホスト内でのネットワーク処理を少なくする事が望ましい。

この事から、ＮＦＥＰを設定する場合次の事に留意する必要がある。

① ＴＳＳ機能を持ついかなるホストにも接続可能なＮＦＥＰを想定し、その際ホストにdependする機能を明確にしておく。

② ホストの持つ標準的な計算機間リンクを用いて、ＮＦＥＰとホストの間の論理リンクを確立する。

③ ホストとＮＦＥＰ間の論理リンクは複数本必要とされるが、物理リンクは１本である事が望ましい。

④ ホスト・コンピュータのソフトウェアの変更又は、追加するソフトウェアは最小限におさえる。

⑤ プロセス間コミュニケーションをサービスできるように考える。

4.2 NFEPの有効性

コンピュータ・ネットワークの構成要素が、サブネットとホストであるという考え方から、FEPの導入が考えられてきた背景には、ホスト・コンピュータの負荷の軽減、データ交換網の処理効率と信頼性の質との間のトレードオフ等の問題の折衷案となるものが研究されてきたためであり、コンピュータ・ネットワークの総合的な経済性を高める事がねらわれている。以下に、現在までに発表されているFEPに対する有効性の賛否の論文を述べる。

A Retzの見解〔1〕

既存のホストをネットワークに加えるためにホストのオペレーティングシステムの変更を最小にする事が望ましい。また信頼性、柔軟性、ホストシステムのパフォーマンスの向上のためにネットワーク機能をホストから明確に分離する事が望ましい。

上記の場合に、しばしばネットワークとホストの間にフロント・エンド・システムを付加することによって対処される。フロント・エンド・ホストは種々の端末や装置のハンドラーばかりでなくネットワーク・プロトコルをサポートするオペレーティングシステムを利用して、小型コンピュータにおいてインプリメントされる。

フロント・エンド・システムのARPAネットワークにおける最も一般的な応用は、ネットワーク・コンピューティング・リソースに対して利用者が端末や装置からアクセスすることの必要性からきている。

TIP、ANTS、ELFシステムは、図4－2に示してあるようなユーザ・フロント・エンドの例である。

上記のシステムにおいては、たとえば、端末の利用者は、フロント・エンドにLoginし、ネットワークの種々のサーバ・サイトに結合するために、TELNETプロトコルを使用する。

ファイル転送プロトコルを使用することによって接続してある周辺装置からあるいはそれに向ってデータのやりとりがなされてもよい。

もう1つのタイプのフロント・エンド・システムは、コンピューティング・リソース（サーバ・ホスト）をネットワークに結合することを容易にする。

このアプローチは、ホスト・ホスト・プロトコルのようなネットワークの通信の仕事をサーバ・ホストからとり除くことを目的としており現存するホスト・システムのソフトウェア修正が禁止されているときには非常に有効である。

サーバ・フロント・エンドにおいて、オペレーティングシステムの機能として要求されるのは、ユーザ・フロント・エンドのそれと同じである。事実、システムはユーザおよびサーバ・フロント・エンドの両者として働くであろう。

図4－3は、サーバとサーバ・フロント・エンドの結合方法を示す。サーバ・フロント・エンドのプロセスはネットワークからのリクエストに応答し、サーバ・フロント・エンド・ポート（ハードウェア結合）を通してサーバへアクセスする。

たとえばフロント・エンド・システムは会話形ターミナルアクセスをサポートするサーバ・システムに属するターミナル・インターフェース・ポートに結合される。サーバとコネクションをとりたいというネットワーク・メッセージがフロント・エンドに受け取られたとき、フロント・エンドで1つのプロセスが作られる。このプロセスは使われていないサーバ・フロント・エンド・ポートを1つアロケートし、リモートのネットワークプロセスとサーバ・フロント・エンド・ポート間での全二重のデータ転送の為の要求を作る。

このようなアプローチは、サーバ・ソフトウェアの変更をせずにネットワークからのサーバ・アクセスを可能にする。しかし、これは一般性に欠け、サーバの機能はターミナルからのそれに限定される。

サーバにソフトウェアの追加が許されるならば、ハードウェア・コネク

ションを経由した通信を許すのにサーバ・フロント・エンド・プロトコルが使える。しかしサーバ内のプロセスによるネットワーク・リソースへの一般的アクセスを許すためにはサーバのＩＰＣ機能が若干必要である。

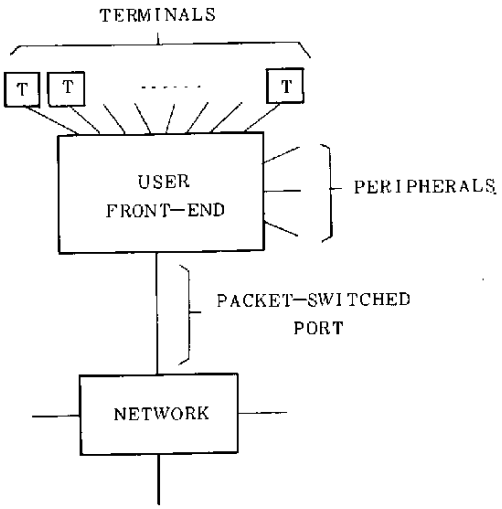


図４－２　ユーザ・フロントエンド

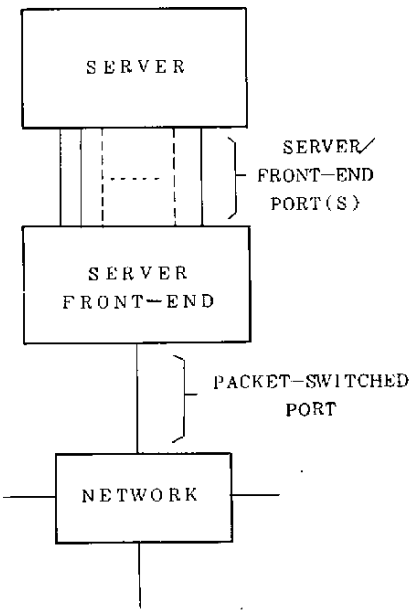


図４－３　サーバ・フロントエンド

B　Woodと Trehanの見解〔３〕

実用の体制におけるパケット交換網の発展についての直接の経験とてらし合わせると、図４－４に示すようなネットワーク・アーキテクチャにすべきであることになる。

このアプローチにおいては、ネットワークに関係したすべてのホスト・ソフトウェアは、ホスト・コンピュータ自身ではなく、Network Front End Processor（ＮＦＥ）に存在する。この方法では、ネットワーク全体に対するホスト･ネットワーク・ソフトウェアは標準ＮＦＥの中で統合され、ネットワークを通してのこのソフトウェアの統一性が達成され、ホス

ト・コンピュータ上に何の影響もなしに維持されるソフトウェアの再設計に対しての柔軟性をあたえる。ネットワークにおけるＮＦＥ間のバリエーションはホスト・コンピュータとの結合である。

ホスト・コンピュータとの結合は、"デバイス・シミュレーション"、フロント・エンド・プロトコルに対する特別なホストのインプリメンテーションのような種々の方法によっておこなわれる。

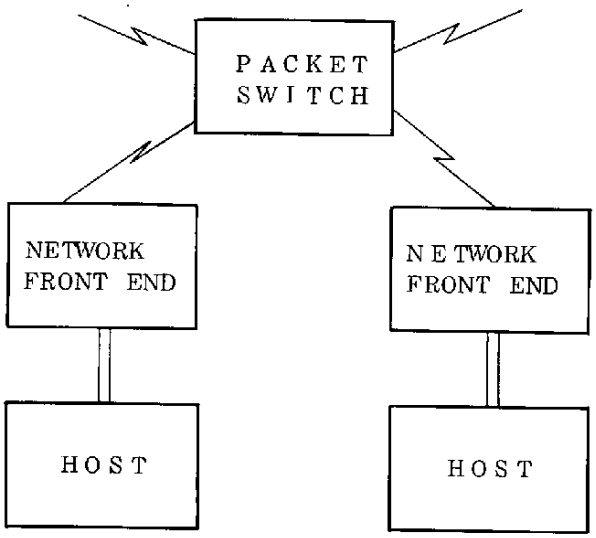


図４－４　新しいネットワーク・アーキテクチャ

C　StokesとHigginsonの見解〔２〕

ＡＲＰＡコンピュータ・ネットワークにホストを結合するためには多量のソフトウェアを開発しなければならない。これはいくつかの不利益を持っている。このうちで重要なのは、処理サービスに対する影響である。

University College London（UCL）のＴＩＰとRutherford Laboratory(ＲＬ)のＩＢＭ３６０／１９５、Cambridge Computer Aided Design Center（CADC）のＡＴＬＡＳを結合する方法として次のような方法を採用した。

すべてのネットワークに従属するソフトウェアは、小型コンピュータ

（ＰＤＰ－９、２４Kwords：１８ビット、２５６Kドラム）の中でインプリメントされ、ホストの中のソフトウェアを変更しないというものである。

ＰＤＰ－９の位置づけは、次のようになる。

ＡＲＰＡネットワークからみたとき、それは標準ホストであって、すべての低位のプロトコルがその中でインプリメントされている。

ＡＲＰＡネットワーク・ユーザからみたとき、それは、トランスペアレントなフロント・エンド機能であり、高位プロトコルのサーバ機能のほとんどがインプリメントされている。ＩＢＭ３６０からみたとき、PDP－９は単に標準リモート・ジョブ・エントリ・ステーションである。IBM360のユーザからみたとき、それはＡＲＰＡネットワーク・プロトコルのユーザ機能をサービスするものである。

彼らの経験では、Telnetにおいて多少問題があった。またＦＴＰにおいては、これにぴったりあてはまるプロトコルがＩＢＭ３６０にはなかったために、種々のものが検討されているところである。特にＦＴＰで問題になるのは、ターミナル・ターミナルのトラフィック程度でよいかということ、ＦＴＰジョブをホスト内に起動したときに時間がかかって（たとえば数分）、ユーザＦＴＰがタイムアウトするのではないかということ、ファイルの形態およびステータス情報などである。

このような問題点があるにもかかわらずＰＤＰ－９によってホストを結合したメリットは大きいと言っている。

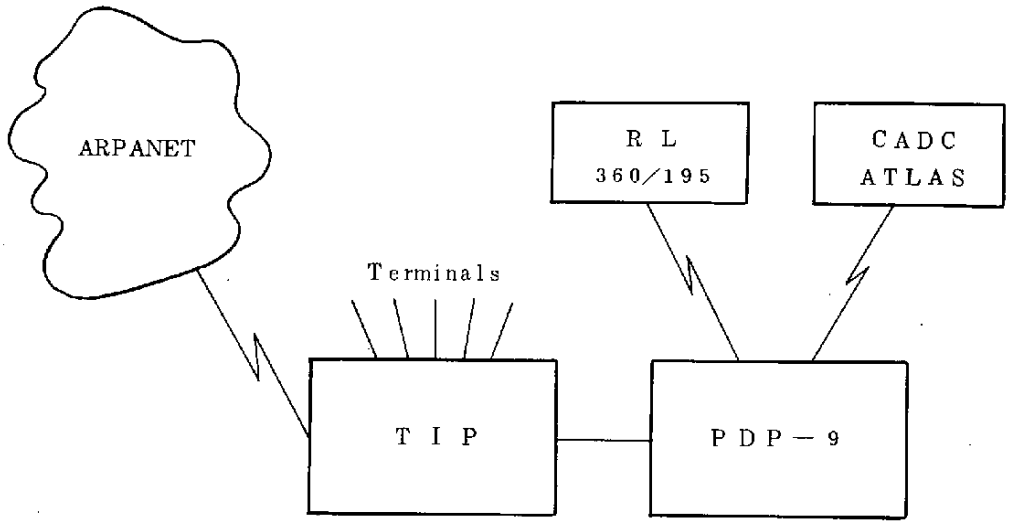


図４－５　ＵＣＬの構成図

Ｄ　du Masleの見解〔４〕

ＩＢＭ３６０／６７は１９６７年に２７０３（調歩同期１００回線とＢＳＣ８回線）と２７０１（ＰＤＰ－８に対する１ＰＤＡとＢＳＣ１回線）をつけたものが導入された。

２７０３は、ＢＳＣポートとして4.8Kbitまでに制限されていた。これは受け入れられるスピードではなかった。唯一の解決方法は、２７０１のＢＳＣアダプタを半二重モードで使用することであった。ＩＢＭでは9.2Kbitまでしか保守していないが、危険をかくごで１９.２Ｋｂｉｔで使用した。

これは現時点においては、Cyclades における唯一の半二重のリンクである。我々はこの解決方法に満足しているわけではない。

いくつかの代替の方法が可能である。

○　ネィティブ・モードでプログラムされたＩＢＭ３７０４／５

○　Ｓ／３６０とＣＴＣリンクでつながったフロントエンドコンピュータ

○　Ｓ／３６０チャネルに直接結合されたマイクロプログラム化された装置

○　ＩＭＰとターミナルコンセントレータが一緒になっている働くフロント・エンド（ＩＲＩＳ４５において使用され、将来は、ＩＲＩＳ８０においても利用される）。この方法はＩＢＭ３７０５で容易にインプリメントできる。

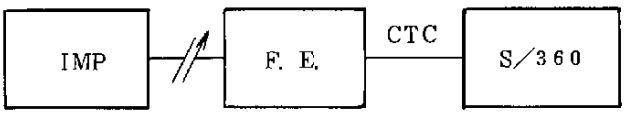


図４－６　ＦＥによる結合

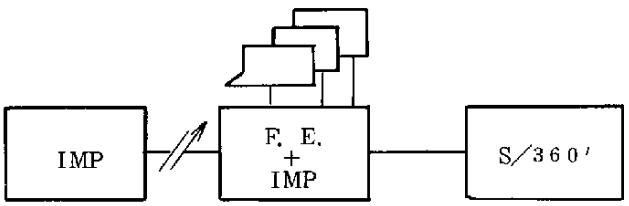


図４－７　ＩＭＰとＦ．Ｅ．が一緒になったもの

Ｅ　ＢＢＮグループの見解〔５〕

ある場合においては大型のホストをネットワークに結合するときにその間にミニコンピュータの挿入をしたいことがある。このマシンの中にホスト・ホスト伝送手順がおかれ、ホストがこれらの仕事に対処することからのがれさせる。この概念において言われていることは、

○　ホストのモニタを変更することは困難であり、コンピューメ・メーカによる新しいモニタのリリースは永続的なコンパティビリティの問題を持ち出す。

○　ホストにおいてコアやタイミングの制限がある。

○　新しいＩ／Ｏ装置の設計や調達をさけるために、すでに存在するかあるいは挿入するミニコンとホストの間で利用できるＩ／Ｏ装置を使うことは望ましいことである。

このアプローチは原則的には正しいようにみえる。事実ある場合には唯一

の可能なアプローチかも知れない、しかししばしば問題がでてくる。

まず第一に、ホストと既存の周辺プロセッサ間のＩ／Ｏ装置はネットワークの結合のために設計されていない。そしてタイミングとバンド巾の制限は大巾なパフォーマンスの低下をきたす。

第二番目に、周辺プロセッサとそのソフトウェアはしばしば外部グループによって提供される、そしてホスト本体はホストについて知っているよりは少なくしか知らないであろう。このような状況で、ネットワークのプログラムは保守され、改良されるが、周辺プロセッサのそれは、そうできないであろう。

このような問題は、世の中に広くネットワークが使用されるようになるまで解決できないものである。

初期のネットワークにおいては、ソフトウェアを直接ホストにいれる方法を試みた。これは、ユーザがネットワークに対してどのような対価をはらうかということである。ネットワークをカード・リーダないしそれに類似したものをシミュレートするというのは現実的ではない。

F　利用研の見解〔６〕

「電子計算機利用に関する技術研究会」のＲＳＳ研究班においてNFEP経由でホストコンピュータをネットワークに参加させる為の提案がコストを基準として述べられている。

コンピュータ・ネットワークの価値は、有用なリソースを保有しているユーザ数に比例するといわれている。このようなユーザを増加させるためには、まずホストの数を増やさなければならない。次に、ネットワークを通じての他ホストのリソースの利用は、直接そのホストのリソースを利用する場合の料金にくらべて、あまり高くなってはならない。

ホストが、ネットワークに参加するためには、少なくとも６～２４人月の工数をかけて、プログラムの開発をしなければならないと言われている。これを金額になおすと、７５０万～3,０００万円の費用がかかることにな

る。この数字はＢＢＮによるものであって、CYCLADES、ＪＩＰNETの場合から考えてみると、３人年～５人年、3,000～5,000万円の人件費がかかるものと考えられる。これらのプログラムを開発するには、ホストのＯＳに対してかなりの知識が要求される。このために、各ホストのサイトで独自に開発するのは、参加するサイトの数を制限することになってしまう。したがって、ネットワークを構築しようとするとき、前もって最少の努力でそのネットワークに参加できるような手段を考えておかなければならない。

現在考えられているホスト・ホスト・プロトコルが汎用的で、プロセス間コミュニケーションを主体としたものである以上、ホストにおいてある程度の量のプログラムを用意せざるを得ない。さらにユーザが実際に使うレベルの高位のプロトコルをもプログラムしなければならない。

このように考えてゆくと、ＮＣＰ、高位のプロトコルを一括してミニコンピュータでサービスし、ホスト特有のインターフェース部分だけをミニコンおよびホストの両側のプログラムにより協力してインターフェースをとるようにした方がよいと思われる。すなわち、全体として１００万ステップになるようなＯＳを理解してホスト内にＮＣＰ、高位のプロトコルを実現するよりも、ミニコンによりインターフェースをとった方がはるかに安価にできるはずである。

実際ホストの中にＮＣＰを組み込んだときに、そのプログラムの大きさは、６８～８６ＫＢになり、これが占有した主記憶装置を買い取るとすれば、1,500～2,200万円に相当する。しかも、高位プロトコル、NCPの両者がホスト内で働くとすると非常に多くのＣＰＵを使用することになる。ＪＩＰＮＥＴを例にとった測定結果では、１４４バイトのメッセージ（ラインプリンタ１行）をＦＡＣＯＭ２３０／７５からＡＣＯＳ７００に送るのに、１メッセージあたりＦＡＣＯＭでは14.5ミリ秒、ＡＣＯＳでは14.0ミリ秒のＣＰＵを使っている。これを料金になおすと、それぞれ

２円５０銭、１円３９銭となる。これをもとに、磁気テープ1巻をFACOMからＡＣＯＳに送ったらどうなるかを計算してみよう。2,４００フィートの磁気テープに1,６００bpiで記憶させると、1,４４０バイトのブロックが１9,２００はいることになる。これを、ＦＡＣＯＭからＡＣＯＳへ送るとすれば、計算機使用料だけで、ＦＡＣＯＭが１５万円、ＡＣＯＳが９万７千円となり、合計２４万７千円となる。

このように高いコストがかかるのは、ＪＩＰＮＥＴのＮＣＰなどが特に効率が悪いからではない。他のネットワークでも同様のことが報告されている。たとえばＡＲＰＡネットワークにおいて、ＵＣＬＡとＵtah大学で行なった測定では、ＵtahのＰＤＰ－１０からＵＣＬＡのＩＢＭ３６０／９１にネットワークを通じて１メガビットのデータを送った場合、伝送のためのコストは３０セントくらいであるが、ＣＰＵコストは１０ドル位になるという〔ＪＩＰＤ７２〕。このデータをもとに、磁気テープ一巻を送るとした場合のＣＰＵコストを推定すれば６６万円ということになる。

このように高いコストがかかるのは、ホストの汎用のＯＳの機能を利用しているからにほかならない。現在６５Ｋバイトの記憶装置を持ったミニコンピュータは約９００万円で買い取ることができ、サブネット、ホストとのインターフェース装置をあわせたとしても合計2,０００万円以下で買い取ることができる。この値段はちょうどホスト内でＮＣＰが占有する主記憶装置の買い取り価格に匹敵する。すなわち、いままでホスト内で処理していた部分をサブネットとホストのあいだにＮＦＥＰをいれることによって、ホストの負担を軽くするだけではなく、全体的なコストおよびネットワークの利用料金も軽減することができる。

ホストとサブネット間に、ＮＦＥＰという一つの要素を入れることによって、故障する要素が一つ増えたことになるが、ＮＦＥＰの故障は特定のホストのみをネットワークから切り離すだけであること、ホスト内にNCPなどを組み込むことによって発生するインターフェース・ミスによるソフ

トウェア故障が減少することを考えれば、無視できる故障率の増加と考えることができる。

またホストのＣＰＵは、非常に高速であると考えられがちであるが、実際には、ＯＳが汎用性を求められるために種々のオーバヘッドを生み、かえってミニコンの方が高速であるという場合がおこる。ＪＩＰＮＥＴにおけるＮＣＰの効率測定でもこの事実が実証されており、ホスト間のデータ転送においてある程度のスループットをだすとホストがＣＰＵバウンドになってしまう状態が発生する。このために、よりスループットをあげるためには、ＣＰＵの高速化複数化あるいは他のコンピュータにＮＣＰ機能をはたさせる等の手段をとる事が痛感されている。この中で最も安価に実現し得る方法はＮＦＥＰの導入であろう。

4.3 ＮＦＥＰの概要

ＮＦＥＰはホストコンピュータとパケット交換網の間のインターフェース機能を持ち、このソフトウェアの多くの部分を標準化し、各ホストの開発するソフトウェアをできるだけ少なくするように考慮されたものである。

ＮＦＥＰはホストコンピュータと複数の論理的リンクによって結合され、このリンクを通じて、プロセス間通信やファイル転送、ＴＳＳ、リモートバッチその他の高位のプロトコルをサービスする事を目標とする。この目標実現の為に、ＮＦＥＰ内では、ＮＣＰ、パケット交換網との通信は勿論の事、プロセス間通信、高位プロトコルのサービスに必要とされる処理の大部分（できるならば全て）を処理する機能を持ち、ホストの負荷の軽減をはかるものである。図４－８にＮＦＥＰとホスト及びパケット交換網の関係を示す。

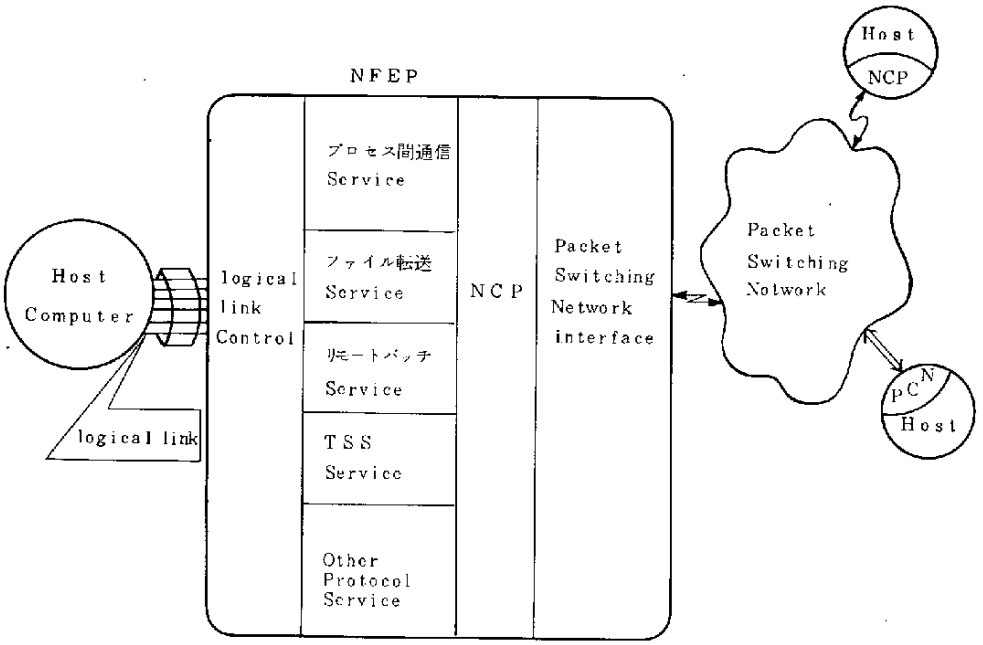


図４－８　ＮＦＥＰの概要

ＮＦＥＰのサービスはプロセス間通信サービスと、高位プロトコルサービスに大きく分けられるが以下に、プロセス間通信及び高位プロトコルについて簡単に述べる。

ＮＦＥＰにおけるプロセス間通信とは、ホスト内のプロセスがＮＦＥＰのＮＣＰを介して他系ＮＣＰのもとのプロセスとの間でコネクションの確立、コネクションを通したデータの授受、コネクションの開放を行なうものである。プロセス間通信をホストのユーザにサービスするためには、コネクションの確立、開放、コネクションを通したデータの送信、受信、状態情報の通知を行なうためのサービス体系を用意する必要がある。ここでコネクションはその両端となるプロセスの合意の上で確立される。通常のユーザが両者の合意の上でコネクションを確立する場合にはこのまゝでよいが、ユーティリティ・プロセスのように相手側のポートが連絡された時コネクションを確立する必要がある場合には、相手からのコネクション確立要求を１つのイベントとしてこれに対する待ち状態をポストするコマンドを必要とする。また、

コネクションを通る通常メッセージとは別に特別な状態情報を通知するためのサービスを処理するためのコマンドも考慮しなければならない。これらを総合すると、ホストのユーザに以下の種類のコマンドをサービスしなければならない。

- コネクションの確立コマンド（ OPEN ）
- メッセージ受信コマンド（ READ ）
- メッセージ送信コマンド（ WRITE ）
- コネクションの開放コマンド（ CLOSE ）
- コネクション待ちコマンド（ LISTEN ）
- 状態情報通知コマンド（ POST ）
- イベント待ちコマンド（ WAIT ）

ネットワークの高位のプロトコルであるＴＳＳ、リモートバッチ、ファイル転送は、他系ホストの要求によってサービスする場合と、ホスト側から利用要求を出す場合がある。ＮＦＥＰは他系ホストの要求によって高位のプロトコルをサービスする場合にはサービスを行なうためのサーバを持つ。TSS及びリモートバッチのサービスはホストの持つＴＳＳ機能、リモートバッチ機能を直接利用することによって実現し、ファイル転送のサービスは、ファイル処理用のプロセスをホスト内に持つことによって実現する。ＮＦＥＰ側のサーバは、要求元のユーザとの間でネットワークの高位のプロトコルの処理を行ない、必要に応じてホストに対しサービスの要求を出す。

ホスト側からネットワークにつながっている他系のＴＳＳ、リモートバッチ、ファイル転送を利用しようとする時、プロセス間通信を利用して行なうことができる。ＴＳＳ端末から他系ＴＳＳを利用する場合には、ホストのＴＳＳインターフェースでＴＳＳジョブと会話し、そのＴＳＳジョブがプロセス間通信を行なうユーザプログラムとなる。

4.3.1　ＮＦＥＰの機能

ＮＦＥＰはホストとサブネットの間にあってホストがネットワークに参加するために必要な機能を処理するミニコンピュータであり次の様な機能を持つ。

① ホストとのインターフェース

② ＮＣＰ

③ リモートＨＩ

④ サブネットとのインターフェース

⑤ プロセス間通信サービス

⑥ ＴＳＳサービス

⑦ リモートバッチサービス

⑧ ファイル転送サービス

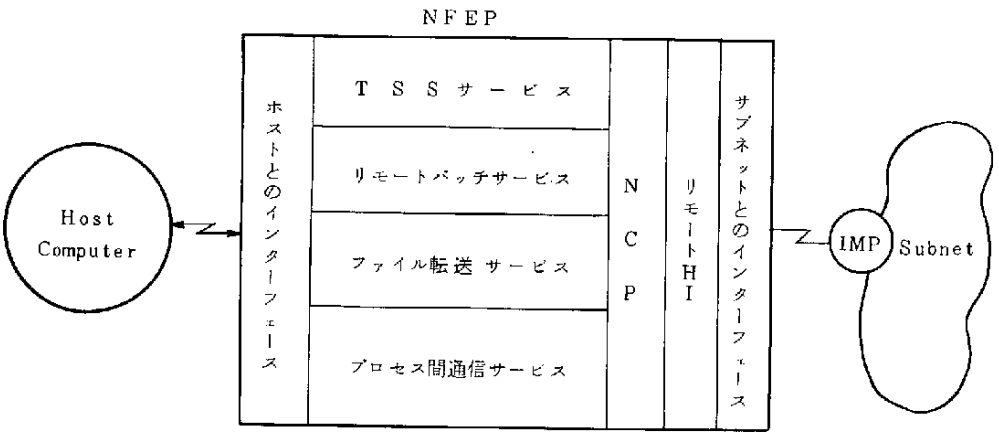


図４－９　ＮＦＥＰの機能

以上の機能のうち必須機能は①～④迄であり、残りの⑤～⑧迄のうちいずれかが満足されればＮＦＥＰとなり得る。⑤～⑧迄の機能のうちいずれを選択するかは、ホストの機能（例えばＴＳＳ、リモートバッチ機能のサポート形態）及びＮＦＥＰの能力（コア、ＣＰＵ）によって異なってくる。

以下に各機能についてさらに詳しく述べる。

(1) ホストとのインターフェース

ＮＦＥＰとホストの間は論理的な複数のリンクで結合され、ホストの持つ通信制御装置またはチャネルと物理的に結合されている。ＮＦＥＰ内のホストとのインターフェース機能は、論理リンクを用いたデータ転送を制御する機能である。この機能はホスト特有の計算機間リンク又は端末アクセス方にあわせて設定される。図４－１０にホストとのインターフェース機能を示す。

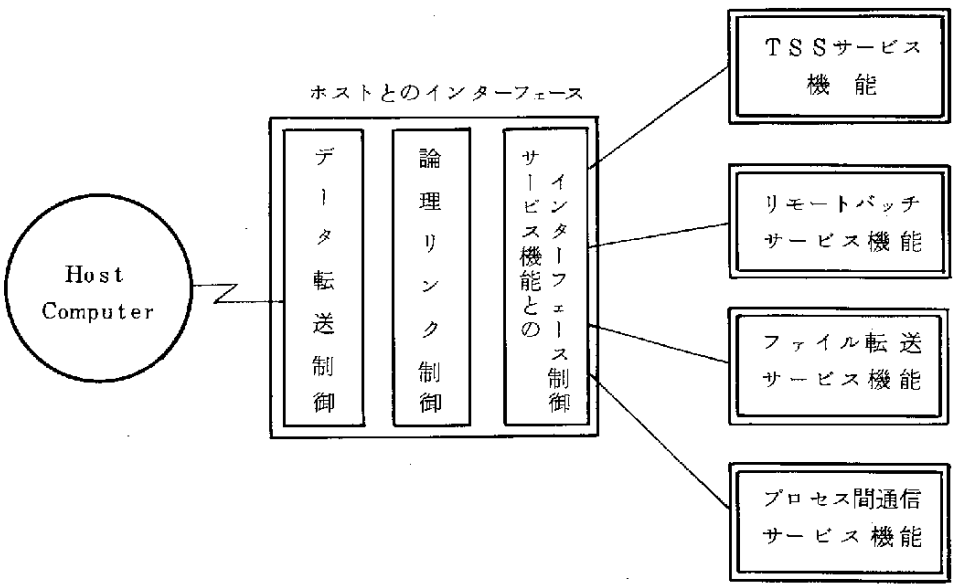


図４－１０　ホストとのインターフェース機能

① データ転送制御

ホストとの間でデータ転送をおこなう入出力制御処理で、データ伝送手順などのリンク制御を含む。チャネル結合の場合にはチャネル入出力、ホストとの取り決めによる転送制御、初期設定、終了会話がその主な機能である。回線結合の場合には、回線入出力、伝送制御手順（例えばＢＳＣ、ＨＤＬＣ）のサポート、ホストとの間の物理リンク制御、メッセージの転送制御などがこの機能の中に含まれる。

② 論理リンク制御

ホストとの間は複数個の論理リンクが設定されるので、この論理リンクを制御する。データ転送制御では物理リンクのコントロールが行なわれるが、ホストとＮＦＥＰの結合方式によっては、論理リンクと物理リンクの数は一致しない場合があり、一般的には一本の物理リンクによって複数の論理リンクを設定する事が多く、この場合には回線多重処理を行なう必要がある。この論理リンク制御では物理リンクにマッピングする操作が含まれる。また論理リンクは、NFEP内のTSSサービス、リモートバッチサービス、ファイル転送サービス、プロセス間通信サービス機能との間の会話に用いられるパスとなるので、サービス機能へのリンクの割当て及び開放、各リンク毎の再送、異常処理を行なう。

③ サービス機能とのインターフェース制御

ＮＦＥＰ内には、ＴＳＳサービス、リモートバッチサービス、ファイル転送サービスなどの高位プロトコルサービス及び、ホスト内のプロセスが他系のＮＣＰ下のプロセスと会話をする手助けを行なうプロセス間通信サービスの各モジュールがあるか、そのサービス機能は非同期にホストへのデータ転送を要求する。この時のメッセージ・トラフィックの制御、バッファの制御を行なう機能を持つ。また、各サービス機能とホストとのインターフェース機能の間は、ある一定のコマンドによって通信されるので、このコマンドのインタプリートも必要となる。

(2) ＮＣＰ

ＮＦＥＰ内のＮＣＰは、サービス機能と他系ＮＣＰとの間にあってホスト・ホストプロトコルの規定に基づく処理を行なう。サービス機能とＮＣＰとの間はＮＣＰサービス・コマンドによるインターフェースがあり、他系ＮＣＰとの連絡の際にはＮＦＥＰ内のＨＩ及びサブネットとの

インターフェース機能を用いる。ホスト・ホストプロトコルはネットワークにまたがるプロセスとプロセスの間の通信を円滑に行なうための手順である。図４－１１にＮＣＰの機能を示す。

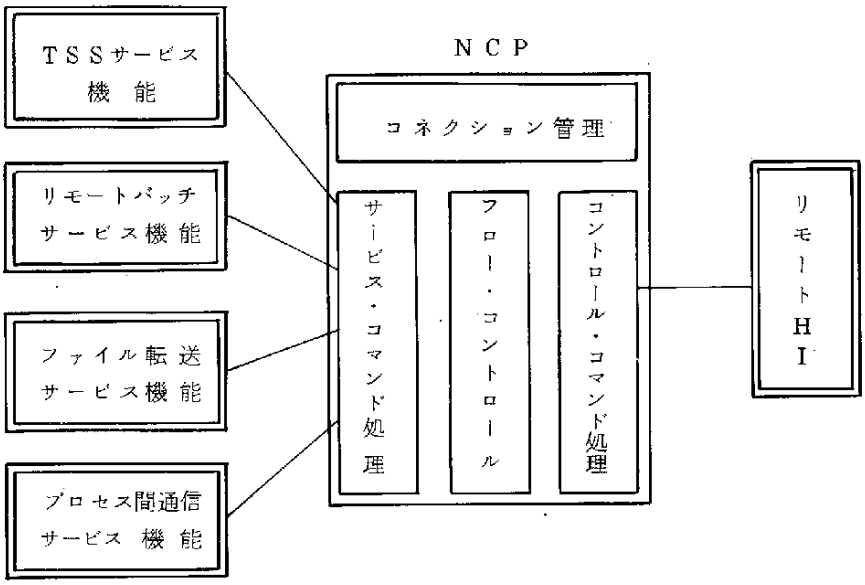


図４－１１　ＮＣＰ機能

①　コネクション管理

ネットワークを利用するプロセス間では、バーチャル・コール方式にもとづいてコネクションを確立する。このコネクションは各ＮＣＰの内部で管理されている。ＮＣＰ内のコネクション管理機能は、プロセス又は他系ＮＣＰの要求により、コネクションの確立、切断、コネクションを通じてのデータの授受、プロセス間の同期信号の授受などの処理を行なう。

②　フロー・コントロール

ＮＦＥＰ内のＮＣＰは、コネクションに関してバッファ確保方式によって他系ＮＣＰとの間でフロー・コントロールを行なう。ＮＣＰ間で１つのコネクションを確立すると、受信用のバッファとしてバッファ

を１ないし数個確保し、その個数を互に連絡する。送信側は受信側で準備されたバッファの個数分のメッセージを送信すると、新たにバッファ確保の通知がくるまで、そのコネクションに対するメッセージの送信を一時中断する。この時受信側はできるだけ速かにバッファを確保し、送信側に通知する必要がある。

③　コントロール・コマンド処理

ＮＣＰ間で授受されるホスト・ホストプロトコルで規定されたコントロール・コマンドの解析及び処理を行なう。このコントロール・コマンドは、リモートＨＩを介してＮＦＥＰ内のＮＣＰに知らされ、他系ＮＣＰへのコントロール・コマンドの送出はリモートＨＩを通じて行なう。

④　サービス・コマンド処理

ＮＦＥＰ内のＴＳＳサービス、リモートバッチサービス、ファイル転送サービス及びプロセス間通信サービスは、ＮＣＰと連絡を行なうために、ＮＣＰサービス・コマンドを用いる。ＮＣＰ内のサービス・コマンド処理では、ユーザからのサービス・コマンドを解析し、コントロール・コマンドの作成並びに、コネクションの管理を行なう。

(3)　リモートＨＩ

リモートＨＩは、ＮＦＥＰとそれに結合されるＩＭＰとの間で、ホスト－ＩＭＰプロトコルの処理を行なう。このホスト－ＩＭＰプロトコルは、ＮＣＰ内蔵型ホストのＮＣＰが持つホスト－ＩＭＰプロトコルとは多少異なり、実際のホストとＩＭＰ間の入出力については規定しないプロトコル（リモートホスト－ＩＭＰプロトコル）に従った処理が行なわれる。リモートＨＩはＮＣＰとサブネットとのインターフェース機能の間にあって両者のデータ・フォーマット変換、ホスト－ＩＭＰ間でのデータ転送制御、転送手順の制御などを行なう。図４－１２にリモートＨＩの機能を示す。

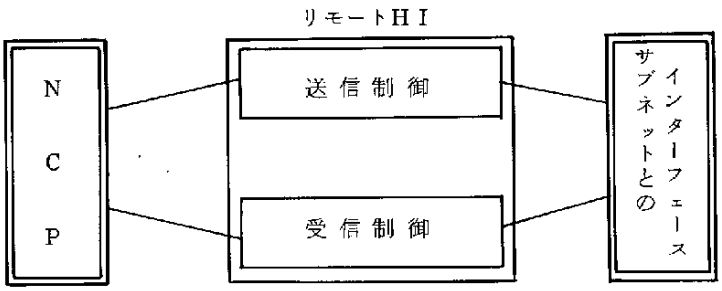


図4－12　リモートHI機能

① 送信制御

ＮＣＰよりホストーホストコマンド、通常メッセージを受け取った時サブネットとのインターフェース機能を介してＩＭＰへメッセージを送信する。通常メッセージは単一セグメント・メッセージと多重セグメント・メッセージに分類され、単一セグメント・メッセージとホストーホストコマンドは、シングルパイプ・メッセージの送信手順で送られ、多重セグメント・メッセージは、ロングパイプ・メッセージの送信手順でＩＭＰに渡される。シングルパイプ・メッセージの送信手順は、ＮＣＰからの連絡により、レシーブを受け取っていないシングル・メッセージの総数が4未満ならば直ちに、ＩＭＰに送出し、4ならば、レシーブを受け取った時に送出される。すなわち、ＩＭＰへ同時に送出可能なシングルパイプ・メッセージを4個でおさえている。ロングパイプ・メッセージの送信手順は、先ず制御情報を送り、それに対するバッファ予約ＯＫがＩＭＰより返された時点でメッセージ本体をＩＭＰへ送出する。ＩＭＰからのレシーブが返却されるまでメッセージを保持しておき、レシーブ受信後にメッセージを構成するバッファをフリーする。

② 受信制御

サブネットとのインターフェース機能を介してＩＭＰよりメッセージを受け取ると、ホストーＩＭＰコマンド、ホストーホストコマンド

通常メッセージのいずれかを解析し、各々の受信制御を行なう。ホストーＩＭＰコマンドはＩＭＰとの間での送信コントロールに関するコマンドでリモートＨＩがその処理を行なう。ホストーホストコマンド及び通常メッセージは、ＮＣＰへ連絡する情報である。

(4) サブネットとのインターフェース

サブネットとＮＦＥＰの間は回線結合され、リモートＨＩの要求によってＩＭＰへデータ転送を行なう事とＩＭＰからのデータをリモートHIへ連絡する機能を持ち、データ伝送を制御する。ＩＭＰとＮＦＥＰの間の回線制御方式は各種のものが考えられるが、現状及び将来の傾向を考えるとＨＤＬＣを用いる事が適当と考えられる。サブネットとのインターフェース機能をＨＤＬＣをもとに図４－１３に示す。

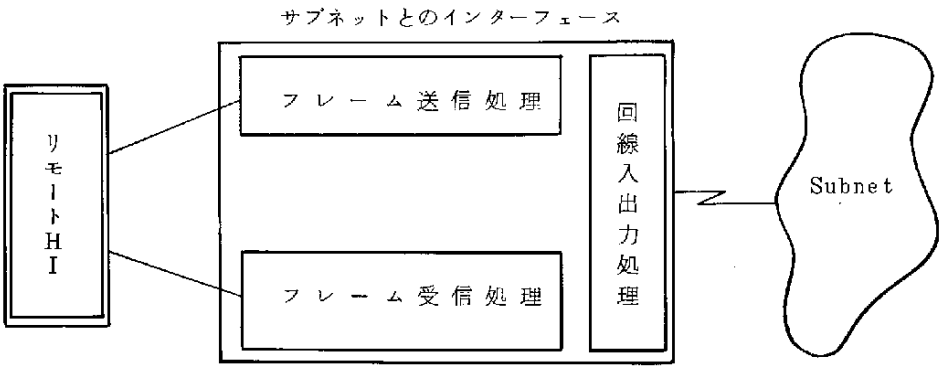


図４－１３　サブネットとのインターフェース機能

① フレーム受信処理

ＩＭＰからのメッセージを受信して、ＨＤＬＣ手順にもとずく処理を行なう。情報フレームの場合にはリモートＨＩへ送出し、監視フレーム、非番号フレームに関しては、受信処理が、フロー制御、バッファ制御などに用いる。フレーム受信処理においては、フレーム受信用のバッファの制御、サブネットから流入してくるフレームのフロー制御を行なう。

② フレーム送信処理

リモートＨＩからの情報フレームあるいはフレーム受信処理からの制御情報に従って、ＩＭＰに送るフレームの送信制御を行なう。

③ 回線入出力処理

回線結合されたＩＭＰへの実際の入出力処理を行なう。すなわちアダプタを通じて知らされる入出力コマンドの終了、割込み等の処理を行なう。又、入出力コマンドの異常処理、コマンドチェイン、バッファチェインなどの処理を含み、ハードウェアとのインターフェースを持つ。

(5) プロセス間通信サービス

ＮＦＥＰを介したプロセス間通信は、ホスト内にプロセス（ユーザ・プログラム）があり、ＮＦＥＰ内のプロセス間通信サービス機能を利用して、他系ＮＣＰ下のプロセスと通信することによって実現する。すなわちＮＦＥＰ内のプロセス間通信サービスは、ホストとのインターフェース機能を通じて論理リンクを確立し、ホスト内のプロセスと連絡をとる事と、ＮＦＥＰ内のＮＣＰを通じて他系ＮＣＰ下のプロセスとのデータ授受を行なう機能を有する。図４－１４にＮＦＥＰを用いてプロセス間通信を行なう場合のプロセス間通信サービスの機能を示す。

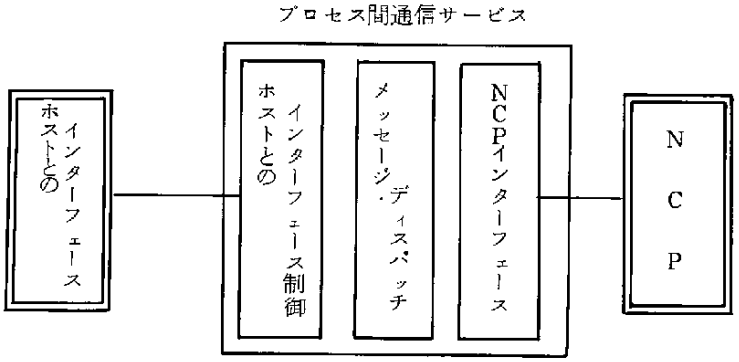


図４－１４　プロセス間通信サービスの機能

① ホストとのインターフェース制御

ホスト内のプロセスに対してデータの授受を行なう為に、論理リンクの確立、開放、論理リンクを通じたデータ授受をホストとのインターフェース機能に依頼する。この時何らかの形でホスト内のプロセス名、論理リンクの識別が可能でなければならない。

② メッセージ・ディスパッチ

ホスト内のプロセスは複数個存在する事があり、また1プロセスが複数の論理リンクを持つ事がある。この時、プロセス名と論理リンクの対によってコネクションへのマッピングが必要とされる。ホストとのインターフェース制御より渡された受信データは、あるコネクションに対応させて、ＮＣＰサービス・コマンドに反映させる処理を行ない、ＮＣＰからのコネクション指定のデータをプロセス名と論理リンクによって指定プロセスへ送信する。

③ ＮＣＰインターフェース

ホスト内のプロセスから送信されてきたデータをＮＣＰサービス・コマンドに変換してＮＣＰに連絡する。又、ＮＣＰから連絡されるサービス・コマンドの終了によってホスト内のプロセスにデータを渡す為の準備を行なう。ＮＣＰサービス・コマンドは、コネクションの確立、切断、コネクションを通じたデータの授受及び状態情報の通知の為のコマンドがある。

(6) ＴＳＳサービス

ＴＳＳサービスは、ＮＦＥＰに接続されている自ホストのＴＳＳを他系ユーザに利用させるサービスと、他系ホストのＴＳＳを自ホスト内のユーザが利用するサービスの2面性を持つ。ＮＦＥＰ内には前者のサービスを提供するサーバと、後者のサービスの手助けをするユーザが必要となる。

サーバは自ホストのＴＳＳが扱える端末をシミュレートする機能及び、

ＴＳＳサービス用の高位プロトコルを処理する機能を持つ。図４－１５にＴＳＳサーバの機能を示す。

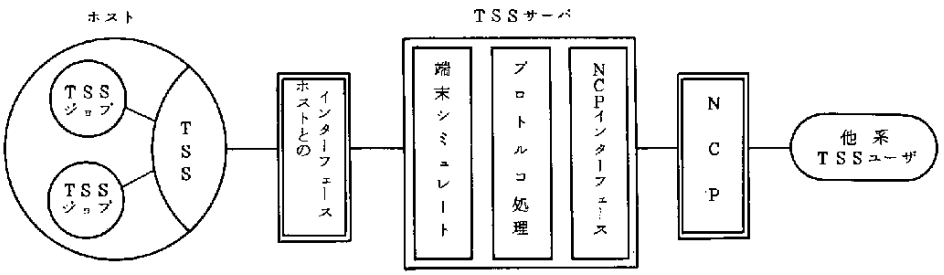


図４－１５　ＴＳＳサーバの機能

①　端末シミュレート

ホストのＴＳＳは一般に特定の端末を想定したインターフェースを公開している。この為ＴＳＳが扱える端末をシミュレートしてホストとのデータ授受を行なわなければならない。また、他系ＴＳＳユーザからの要求によってホストのＴＳＳに対しての接続指令、切断指令を出す為の機能をもつ。この時ＴＳＳサーバは１本の論理リンクを確保、開放するためのリンク確立要求をホストとのインターフェース機能に通知する。ホスト内のＴＳＳジョブはＴＳＳサーバとの間に１本の論理リンクを持つため、ＴＳＳサーバは、複数の論理リンクを管理しなければならない。

②　プロトコル処理

ＮＦＥＰ内のＴＳＳサーバと他系ＮＣＰ下のＴＳＳユーザ又はＴＩＰ内のＴＳＳユーザの間は、ＴＳＳサービスの為の高位プロトコルが規定されるので、ＮＦＥＰ内のＴＳＳサーバはそのプロトコルを処理する機能を持ち、他系のＴＳＳユーザと通信を行なう。

③　ＮＣＰインターフェース

ＮＣＰインターフェースは先に述べたプロセス間通信と同様な働きを持つが、ＴＳＳサーバでは特にＮＣＰに対して他系ＴＳＳユーザが

コネクション確立要求を出した時その事象が確認できるように、コネクション確立要求をイベントとしてとらえられる事ができるＮＣＰサービス・コマンドを出しておく。

他系ホストのＴＳＳを自ホストから利用する場合、ＴＳＳユーザは２つのプログラムからなる。１つはＮＦＥＰ内に存在し、ＴＳＳ用のプロトコルをインタプリートする（ＴＳＳユーザ１）。いま１つは自ホスト内に存在し、ホストのローカルな端末と通信するプログラムである（ＴＳＳユーザ２）。両プログラムは論理リンクを用いて通信し、その通信の方法は、プロセス間通信と同様な考え方で実現する。ＴＳＳユーザ２は、ホストの持つ端末アクセス法に基づいて端末と通信するので、ＴＳＳジョブとなる事が多く、ＴＳＳユーザ１とＴＳＳユーザ２の間は、ローカルな取り決めを行なう。以下に他系ホストのＴＳＳを使う手順の例を示す。

① 他系ＴＳＳを用いる端末ユーザは、自系ホストの端末より、ＴＳＳユーザ２と結合する。

② ＴＳＳユーザ２はプロセス間通信を行なうと同様な手順で、ＴＳＳユーザ１と論理リンクを確立する。

③ ＴＳＳユーザ１は、目的ホストのＴＳＳサーバに対してコネクションの確立を要求する。

④ コネクションが確立すると、端末ユーザは目的とするホストのＴＳＳに従って、そのサービスを受ける。

ＮＦＥＰ内のＴＳＳユーザ１の機能は、ＴＳＳユーザ２との会話ＴＳＳサービス用高位プロトコルのインタープリート、ＮＣＰとのインターフェースからなる。

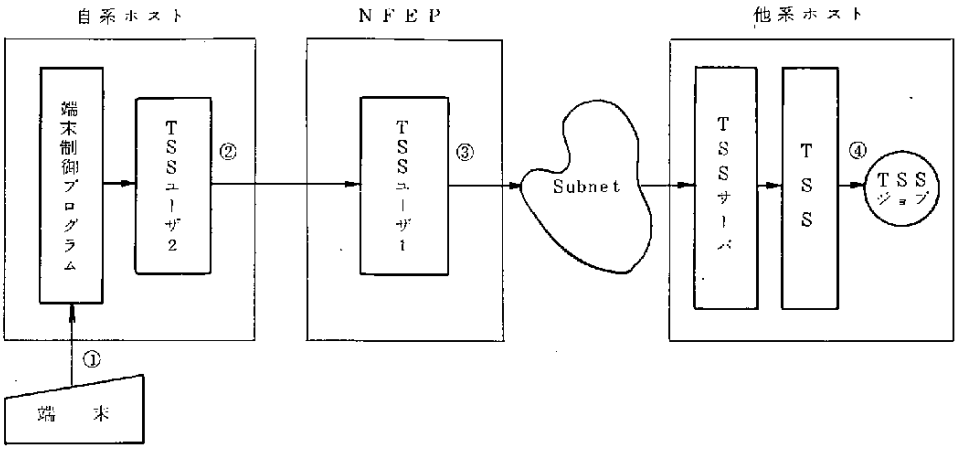


図４－１６　他系ＴＳＳの利用

(7)　リモートバッチサービス

リモートバッチのサービスもＴＳＳサービスと同様に、自ホストのリモートバッチ機能を他系のユーザに利用させるサービス（サーバ）と、他系ホストのリモートバッチ機能を自ホスト内のユーザが利用するサービス（ユーザ）の２面がある。

自ホストのリモートバッチ機能を利用させるためにＮＦＥＰ内にはリモートバッチサーバが存在し、リモートバッチがサポートする端末群又はステーションをシミュレートする。リモートバッチ機能をＴＳＳの一部として持つＯＳの場合には、ＴＳＳサーバと同様に考えればよい。リモートバッチサーバは、ＴＳＳサーバとほぼ同様な機能を持ち、プロトコル処理部が、リモートバッチプロトコルの処理となる。

他系ホストのリモートバッチを使う場合には、自ホストのリモートバッチ端末から使う場合と、バッチジョブとしてプロセス間通信と同様な方式を用いて他系リモートバッチを用いる方式がある。前者の場合には、ＮＦＥＰ内にリモートバッチユーザ１があり、ホスト内にリモートバッチユーザ２があって、ＴＳＳユーザと同様な形態をとる。後者の場合には、バッチジョブ自身がリモートバッチユーザとしてインプリメントさ

れ、ネットワークへのアクセス手段として基本的なプロセス間通信機能を利用する為、ＮＦＥＰ内では、プロセス間通信サービスがあればよい。図４－１７にリモートバッチサーバの実現法を、図４－１８にリモートバッチユーザの実現法を示す。

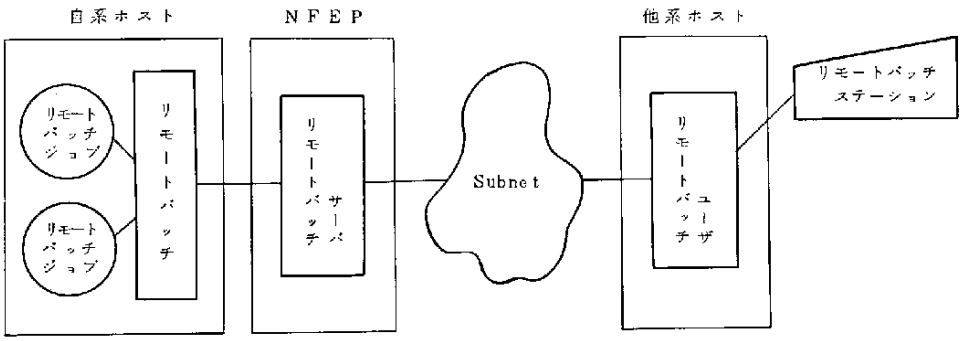


図４－１７　リモートバッチサーバ

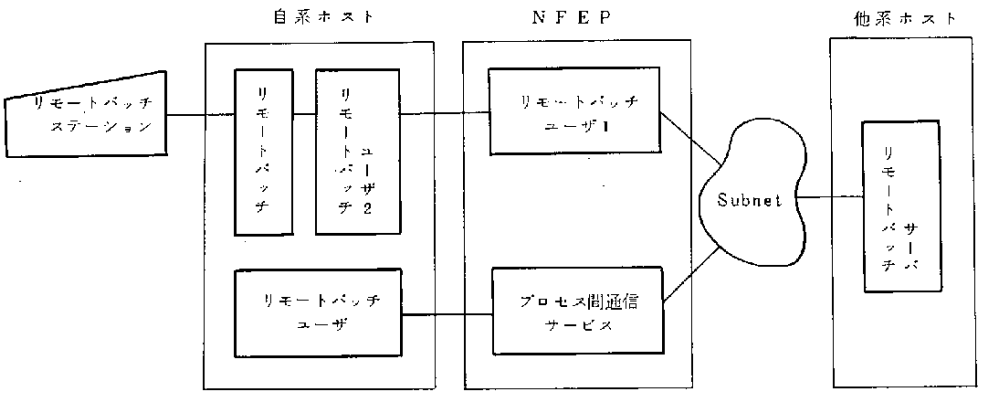


図４－１８　リモートバッチユーザ

(8)　ファイル転送サービス

ファイル転送サービスもＴＳＳサービスと同様２つのサービスを持つ。１つは他系のユーザの要求によって転送されてきたファイルを自ホスト内で蓄積管理を行なうサービス（サーバ）であり、いま１つは、自系ユーザの要求によって他系ホストにファイルを転送し、他系で蓄積管理して

もらい、又は他系に管理してもらっているファイルを送り返してもらう機能をサービス（ユーザ）するものである。ファイル転送サービスの特徴は、ファイルのアクセス部分がホストに存在し、ホスト特有のファイルアクセス方式が必要とされる事である。この事により、ファイル転送サービスの実現には、ホスト内にファイルアクセス部分、ＮＦＥＰ内にプロトコル処理部分と明確に機能分割されている。

ファイル転送サーバは、ＮＦＥＰ内のファイル転送サーバ（サーバ1）と、ホスト内のファイルアクセスプロセス（サーバ２）の２つのプログラムからなる。両プログラムの間は、プロセス間通信と同様な方法で論理リンクを確立することができる。サーバ１はファイル転送プロトコルを処理する機能と、サーバ２との通信機能及びＮＣＰインターフェース機能を持つ。サーバ２は、ネットワークファイルの入出力、ファイルの創成、消却、検索などの機能を持ち、さらにサーバ１との通信機能を持つ。サーバ１とサーバ２の間はローカルな規約を設けておく必要がある。図４－１９にファイル転送サーバを示す。

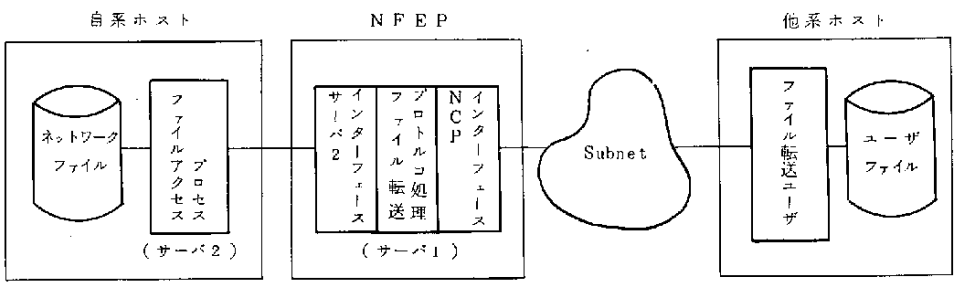


図４－１９　ファイル転送サーバ

ファイル転送ユーザは、ファイル転送サーバと同様に２つのプログラムからなる。１つはＮＦＥＰ内に存在し、ファイル転送プロトコルのプロトコルインタープリート、ＮＣＰとのインターフェース及び、ホスト内のファイルアクセスプロセスとのインターフェースをもつファイル転送ユーザ（ユーザ１）である。他の１つは、ホスト内に存在し、ユーザ

の持つファイルにアクセスする機能を持つユーザファイルアクセス用プロセス（ユーザ２）であり、このユーザ２は、ＮＦＥＰ内のユーザ１とのインターフェース処理機能も持つ。

4.3.2　ＮＦＥＰとホストの結合

ＮＦＥＰとホスト・コンピュータの結合方式には、通信回線を経由するものと、チャネル結合によるものがある。通信回線を用いるものには、さらに三つの代表的な結合方法がある。これらの結合方式のうちいづれを選択するかは、ホスト・コンピュータの機能及びミニコンピュータに対するインターフェースによって異なってくる。以下に代表的な結合方式及び特徴を示す。

(1)　端末装置をシミュレート

ホストの持つ通信制御装置を介してＮＦＥＰを端末と見なして回線結合する。ＴＳＳ、リモートバッチ、オンライン等のホストの持つ機能をそのまゝ利用でき、ホスト・コンピュータに新しいソフトウェアを追加する必要がない。図４－２０にその形態を示す。

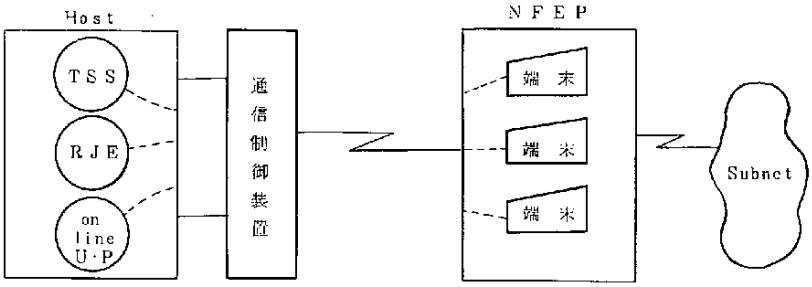


図４－２０　端末装置シミュレートによる結合

この方式によれば、結合は非常に容易であるが、通信制御装置とＮＦＥＰ間の伝送制御手順がＯＳの持つ手順を用いる為、トランスペアレント伝送、会話の開始等で規制を受ける可能性がある。又ホストの持つ通信制御の機能に大きく左右される。

(2) 通信制御装置のソフトウェアを変更

ホストと通信制御装置のインターフェースは(1)と同じ方法で、通信制御装置とＮＦＥＰ間のインターフェースをＮＦＥＰとの結合に便利なように変更する。この場合にも(1)と同様にＴＳＳ、リモートバッチ、オンライン等のホストの機能をそのまゝ利用できるが、通信制御装置の変更又は改造という問題が生じてくる。図４－２１にその形態を示す。

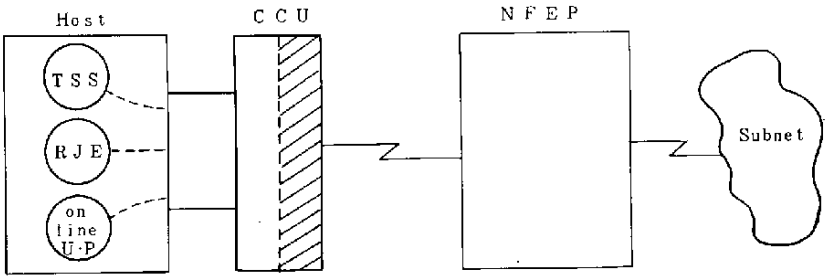


図４－２１　通信制御装置のソフトウェア変更による結合

通信制御装置とＮＦＥＰとの間は、伝送制御手順を適当に定める事によって、いかなる方式のデータ伝送も可能となる。すなわちＮＦＥＰにとって非常に都合のよい結合形態となる事ができるが、通信制御装置がホストに最適なものとならなくなる危険性がある。

(3) ホスト側のインターフェースを変更

ホストに回線経由で直接ＮＦＥＰを結合する。この時、ホストの既存の通信制御装置インターフェースをそのまゝ使用できない事があり、ホスト側の変更という事が生じる。又、ホスト側のＴＳＳ、リモートバッチ、オンライン等の利用の為のソフトウェアが必要となる事がある。図４－２２にその形態を示す。

通信制御装置を用いないので(2)の様な改造は生じないが、ホスト側の変更を必要とする時にはＯＳとのインターフェース部分の調査、インプリメント等で問題があり、又ＯＳのバージョンアップ時の対策も考えておかなければならない。但し、ＮＦＥＰとホスト間の伝送制御手順はか

なり自由に設定することができる。

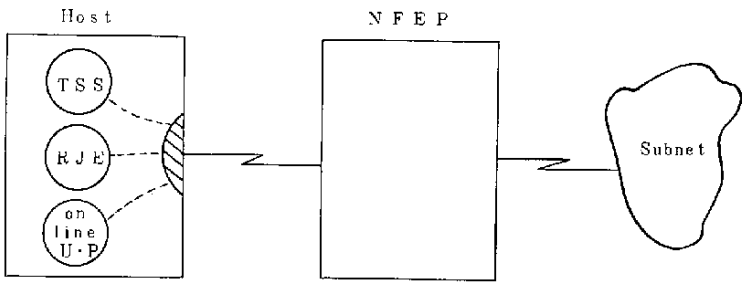


図４－２２　ホスト側のインターフェース変更による結合

(4)　チャネル結合

ホストとＮＦＥＰをチャネルで結合するもので、両者の間には、インターフェース・アダプタが接続されるのが一般的である。この時、ホスト内にユーザ・プログラムとのインターフェースを持つ為のソフトウェアが必要となる。図４－２３にその形態を示す。

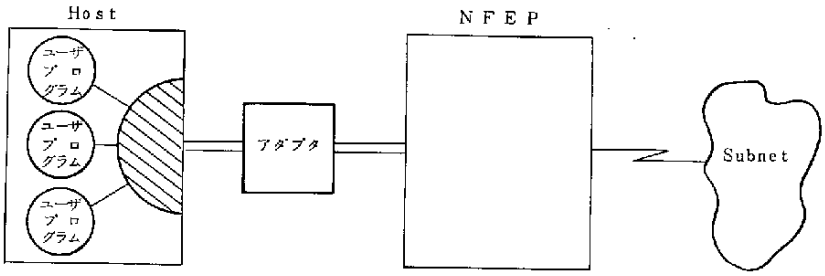


図４－２３　チャネル結合

この方法の利点は、ホストとＮＦＥＰ間で、トランスペアレントなデータの送受を高速に行なうことができることと、ホストとＮＦＥＰ間で任意な規約を設ける事ができる事である。また欠点としては、ホストとＮＦＥＰの間に特殊なインターフェース・アダプタを必要とする事がある事及び、ホスト内にネットワーク処理用ソフトウェアの一部が必要とされる事などである。

以上の結合方式のうちどれを選択するかは、ＮＦＥＰに要求されている

機能及び、その機能の重みづけによって決められるものであり、又、ホスト側の制約条件を吟味したものでなければならない。

次に物理リンクと論理リンクについて述べる。

ＮＦＥＰとホストの間には論理的に複数のリンクで結合されている必要がある。すなわちホスト内の多くのユーザがこのリンクを通じて任意にＮＦＥＰを通じてネットワークに参加する事ができなければならない。この複数のリンクのうちあるものはＴＳＳサービス、リモートバッチ等の高位プロトコル処理の為に用いられ、又他のものはプロセス間通信に用いられる。この複数のリンクをサポートする場合ホストとＮＦＥＰ間には、複数の物理リンクを持つ場合と、１本の物理リンクで回線多重をサービスする場合とがある。

(1) 複数の物理リンク

複数の論理リンクに対応して同数以上の物理リンクが割当てられる。この結合によると、ハードウェア・コストは高くつくが、リンクの制御が簡単である。又ホストによっては、１端末１回線を標準的なリンクとしているものがあり、この様な場合にホスト側のソフトウェアの変更なしでＮＦＥＰと結合しようとする場合に有効である。図４－２４にその概念を示す。

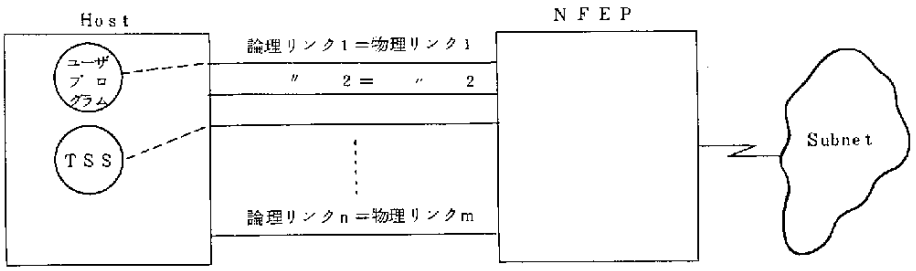


図４－２４　複数の物理リンク

物理リンクの方が論理リンクより多い場合には、その物理リンクはリンクの制御（リンク割当て、確立、開放等）の為に用いる事ができる。

(2)　1本の物理リンク

1本の物理リンクで複数の論理リンクをサービスする為には、ホスト側に回線多重処理用のソフトウェアが必要となる。このソフトウェアがホストの標準的な機能としてサポートされている場合には問題がないが、回線多重化処理ソフトウェアが含まれないホストの場合にはユーザがこのソフトウェアを用意する必要がある。一般にこのソフトウェアは、OSとの関係が非常に密接なものなので、作成時の労力がかかると考えられる。

この結合によると、ハードウェア・コストは(1)に比べて非常に少なくてすむ利点がある。図4－25にその概念を示す。

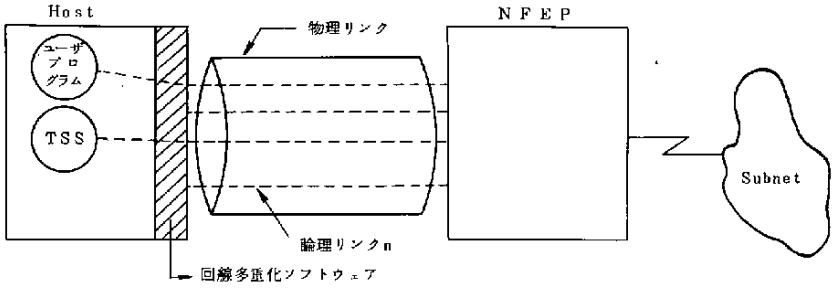


図4－25　1本の物理リンク

4.4　プロトタイプNFEPの論理設計

ホストコンピュータをある特定の機種に仮定した時のNFEPの実現性及び論理設計について、ホストインターフェースを重点にして考察する。

以下に、ホストとNFEPのインターフェース、NFEPの実現方法、NFEPの論理設計、VTAM高位プロトコル及びエンドプロセスの機能について述べる。

4.4.1　ホストとNFEPのインターフェース

ホストコンピュータとNFEPとの結合は、通信制御装置を介した通信

回線結合とし、ホスト側の通信アクセス法はＶＴＡＭ（仮想通信アクセス法）を採用する。このＶＴＡＭはホストの持つ標準的な計算機間リンクであり、１本の物理リンクで複数の論理リンクの取扱いが可能な通信方法を持つ。以下にＶＴＡＭを用いた通信方式の概要を示す。

(1)　ＶＴＡＭ通信ネットワーク

ＶＴＡＭはホスト内に唯一つ存在し、通信制御装置、クラスタ、網制御装置、端末によるＶＴＡＭ通信ネットワークを構成する。

ＶＴＡＭが最終的なユーザすなわち端末との通信を行なう場合の通信方式には、基本モードの通信と、レコードモードの通信がある。基本モードの通信は、基本モード伝送制御手順の回線に接続する端末と通信するもので、レコードモードの通信は、クラスタ制御装置内の論理端末と通信する方式である。ＮＦＥＰとの通信方式は後者のレコードモード通信を採用している。図４－２６にＶＴＡＭ通信ネットワークの構成と、レコードモード通信を示す。

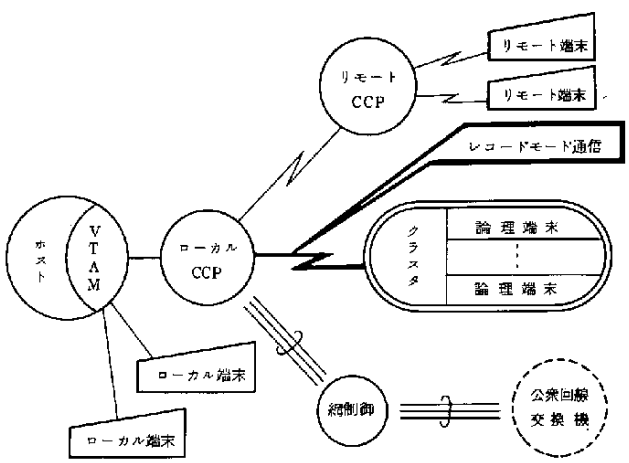


図４－２６　ＶＴＡＭ通信ネットワークとレコードモード通信

レコードモードの通信は、論理端末との通信を行なうので、基本モードのように端末の実体に対応した物理的属性及び、伝送制御手順を意識

することなく、論理的な端末の性格だけを意識して通信することができる。また、論理端末との通信で用いるデータは、基本モードのように端末に依存したものではなく、全ての論理端末が共通に利用できるメッセージと応答によって通信を行なう。このメッセージと応答は、ホスト内のユーザプログラムと、クラスタの論理端末との間で定義されるものである。

論理端末は、インテリジェントを要求され、物理的属性をその内部で吸収できるようにプログラム可能な端末又はこれと同等の機能を有する端末、あるいは、プログラム自体と考えることができる。

ＶＴＡＭ通信ネットワークの中でレコードモードを用いてアクセスするネットワーク体系では、端末の役割は従来の端末と大きく異なり、端末の種別をホストシステムに影響させず統一的、標準的に取り扱うことができる。この事によりホストシステムへの負荷の低減をはかることが可能となる。以下にレコードモードを用いた時のＶＴＡＭ通信ネットワークの特徴を示す。

① データリンク制御、経路制御、転送制御、機能制御及び業務という機能が明確に分割され、ホストシステム、端末システム間で論理的に対象である。

② 各々の対応する機能は、制御上の規約（プロトコル）を設定する。特にエンドユーザ（業務）が使用するプロトコルをＶＴＡＭの高位プロトコルと称し、ＶＴＡＭのユーザ・プログラムと端末システム内の論理端末との間での通信を規定する。

③ レコードモードを用いたネットワーク体系下の端末はある程度のインテリジェンシを持たせられなければならない。

(2) ＶＴＡＭ高位プロトコル

ＶＴＡＭは通信ネットワークを制御するためにいくつかの通信規約(プロトコル）を設定している。ＶＴＡＭのプロトコルには、ＶＴＡＭがシ

ステム内部で持つ通信網制御、転送制御、機能制御などを行なうプロトコルとエンドユーザが使用する高位のプロトコルがある。

ＶＴＡＭ高位のプロトコルは、ホスト側のＶＴＡＭユーザプログラムと、端末システム側の論理端末との間で設定される。このプロトコルはユーザプログラムと論理端末の間の相互の了解により任意に定める事ができ、両者の間でのデータ転送の規約となる。

(3) セッション

ホスト内のＶＴＡＭユーザプログラムと端末システム側の論理端末はそれぞれエンドプロセスと称し、両プロセス間にはセッションが設定される。１本の物理リンク中にはプロセスの対と同数のセッションが存在でき、それぞれのセッションは互に干渉する事なく独立のパスとして平等に制御される。

エンドプロセス間で通信を行なおうとする時、セッションの確立を行ないエンドプロセス間の通信が終了した時セッションの解除を行なうが、セッションの確立と解除はいずれかの側のエンドプロセスの要求によって行なう。

(4) ホストとＮＦＥＰの結合

ホストコンピュータとＮＦＥＰ間は１本の物理回線で結合され、伝送制御手順は、ＨＤＬＣを用いる。ホスト側からみるとＮＦＥＰは複数個の論理端末を持つクラスタ・ノードと考える事ができる。

ホスト側のエンドプロセスとＮＦＥＰ側のエンドプロセスの間はＶＴＡＭ高位プロトコルで通信規約が設定され、それぞれのエンドプロセスは、ＶＴＡＭ高位プロトコルの処理とネットワークを利用する為の各種処理機能を持つ。図４－２７にホストとＮＦＥＰインターフェースの概念を示す。

ホスト側のＴＳＳ、ＶＳＰ（VTAM Service Program）にユーザプログラムがバインドされたもの及び、ＮＦＥＰ側のＴＳＳサービス、

ＰＣＳ（inter Process Communication Service ）はそれぞれエンドプロセスである。

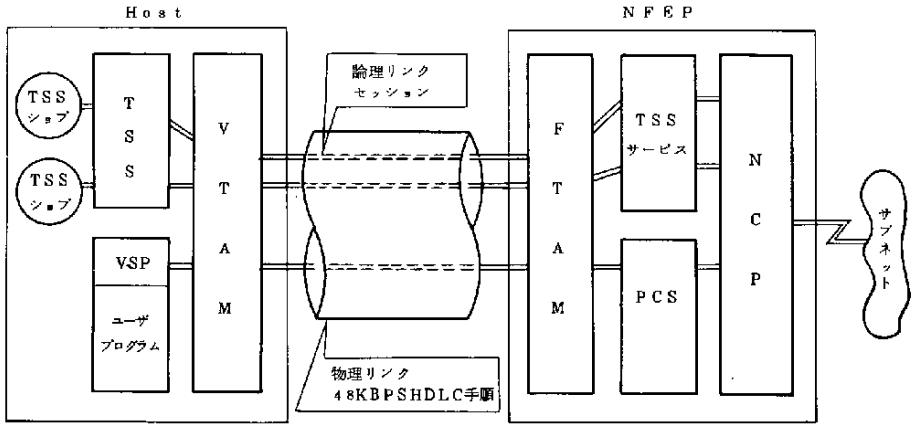


図４－２７　ホストとＮＦＥＰのインターフェース

ホストのＶＴＡＭはＶＴＡＭネットワークのレコードモード通信の機能を持ち、ＮＦＥＰのＦＴＡＭ（Front-end Telecommunication Access Method ）と協力して通信網制御、転送制御、機能制御等の処理を行なう。

ホストのＶＳＰの形態は、ＴＳＳと同様な独立したプロセスとすることが可能であり、又、図４－２７に示すようにユーザプログラムにバインドする形にすることもできる。このＶＳＰはＶＴＡＭが提供するマクロを用いてＮＦＥＰのＰＣＳと通信を行なうことによってプロセス間通信を実現することができる。

ホストのＴＳＳエンドプロセスとＶＴＡＭ高位プロトコルを実現するのはＮＦＥＰのＴＳＳサービスである。ＴＳＳサービスはホストコンピュータのＴＳＳ機能にもとずいたプロトコルを実施するように設定する。

(5)　ＶＴＡＭレコードモードの特徴

ＶＴＡＭレコードモード・インターフェースを用いてＮＦＥＰを実現するときの特徴を以下に示す。

① 1本の物理回線には複数の論理リンク（セッション）が設定でき、互のセッションは独立のパスとして平等に制御される。

② セッションの確立、解放はエンドプロセスの要求によって行なわれるがホスト側、ＮＦＥＰ側いずれのエンドプロセスでもよい。

③ ＶＴＡＭはそれ自体セッション内のメッセージ・トラフィクの制御を行なう。

④ セッション上のデータはビットを基調とするので、エンドプロセス間のデータの透過性は保障される。

⑤ シグナルデータによってエンドプロセスは、割込みデータを知ることができる。

⑥ エンドプロセスがＶＴＡＭのユーザ・プログラムの時、ＶＴＡＭマクロを利用して簡単にＶＴＡＭ高位プロトコルを定義できる。

⑦ セッション上のデータフローは2種類あり、緊急メッセージの優先処理を行なうことができる。

4.4.2 ＮＦＥＰの実現方法

ＶＴＡＭレコードモード・インターフェースを用いたＮＦＥＰは、プロセス間通信及びＴＳＳサービス、リモートバッチサービス、ファイル転送サービス等の高位プロトコルのサービス機能を持つ。以下に各サービス機能の実現方法を示す。

(1) プロセス間通信の実現方法

ＮＦＥＰを介したプロセス間通信は、ホスト側にユーザプログラムがありＮＦＥＰ内のプロセス間通信サービスプロセスを利用して他系NCP下のプロセスと通信を行なうことによって実現する。図４－２８にプロセス間通信の概要を示す。

ホスト内のユーザプログラムは、プロセス間通信を行なおうとするユーザのプログラムでありＶＳＰをバインドした形で1つのプロセスを形成

する。ＶＳＰはネットワーク・サービスを行なうためにホストに付加したモージュールであり、ＮＦＥＰ内のＰＣＳとの間でＶＴＡＭ高位プロトコルを設定し、メッセージのフロー制御等の機能を果す。ユーザプログラムとＶＳＰの間は、ユーザ・サービスコマンド（ユーザ・マクロの形で実現）を用いて会話を行ない、ＶＳＰはＶＴＡＭマクロを用いてＶＴＡＭと会話する。

ＮＦＥＰ内のＰＣＳは、ＮＣＰとの間でＮＣＰサービスコマンドを用いて会話し、ＮＦＥＰ内のＦＴＡＭ、ホストのＶＴＡＭを通してＶＳＰと通信する。

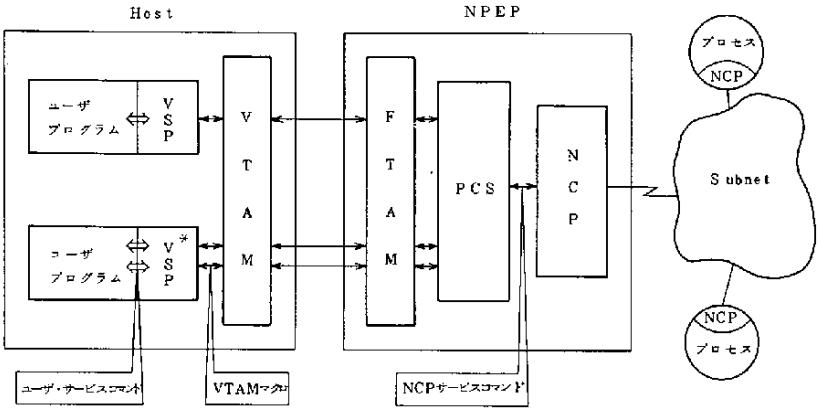


図４－２８　プロセス間通信の実現方法

(2)　ＴＳＳサービスの実現方法

ホストのＴＳＳ機能を他ホストのユーザにサービスするためには、ＮＦＥＰ内にＴＳＳサーバが必要となる。ホストのＴＳＳとＮＦＥＰのＴＳＳサーバ間はＶＴＡＭ高位のプロトコルによってデータ転送が制御される。図４－２９にＴＳＳサービスの実現方法を示す。

他系のＴＳＳ端末（ＴＳＳユーザ）からのlogonによりＮＦＥＰ内のＴＳＳサーバは、ＦＴＡＭに対して論理端末からのセッション確立要求

を出す。

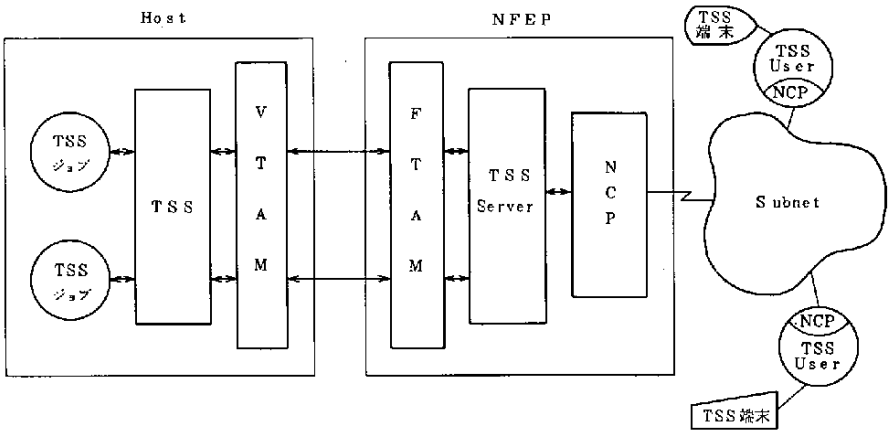


図4－29　ＴＳＳサービスの実現方法

ＦＴＡＭは、空きセッションを選択しVTAM を通してＴＳＳに logon 要求をかけ、ＴＳＳとＴＳＳサーバの間でセッションが確立する。そのセッションを通してホスト側のＴＳＳジョブと他系のＴＳＳ端末の間のデータ転送が可能となり、TSSサービスを行なうことができる。NFEP側のＴＳＳサーバは、他系ＴＳＳユーザとの間でのプロトコルインタプリート、仮想端末の処理及びホストのＴＳＳとの会話を行なう。

(3)　リモートバッチサービスの実現方法

リモートバッチサービスはＴＳＳサービスと同様にホストのリモートバッチ処理機能を利用する。ＮＦＥＰ内にはホストのリモートバッチ処理機能とのインターフェースをとるためのリモートバッチサーバが必要である。ホストのリモートバッチ処理用制御プログラムとＮＦＥＰのリモートバッチサーバの間はＶＴＡＭ高位プロトコルが設定される。図4－30にリモートバッチサービスの概念を示す。

他系ＮＣＰのもとのリモートバッチユーザ（リモートバッチ端末システム又はリモートバッチプロセス）からの logon によりＮＦＥＰ内のリモートバッチサーバはホストのリモートバッチ処理機能に対して logon

要求を行ない、セッションを確立する。ホスト側のリモートバッチファイルは、リモート入力及びリモート出力用のファイルとしてリモートバッチシステムの管理下にある。ＮＦＥＰ内のリモートバッチサーバは前述のＴＳＳサーバと同様な機能をもつ。またリモートバッチを用いないでＴＳＳのジョブ起動の機能を用いてリモートバッチサービスを行なう事も可能である。

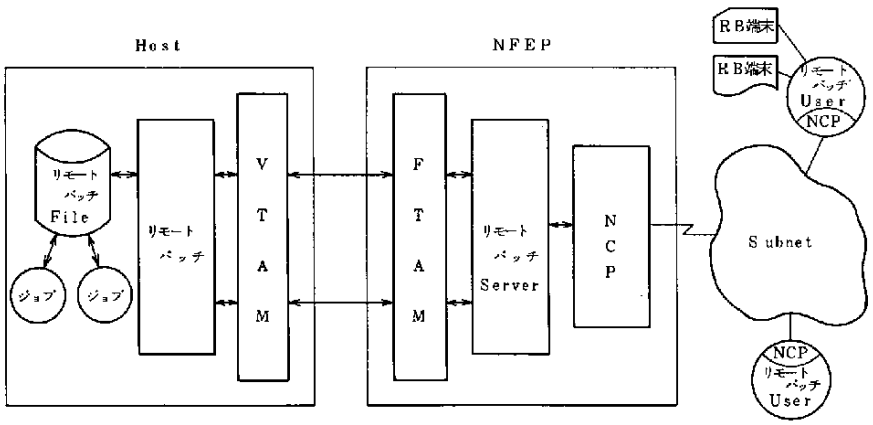


図４－３０　リモートバッチサービス

(4)　ファイル転送サービスの実現方法

ファイル転送サービスはネットワーク内のホスト間でのファイルの転送及びローカルなホストのネットワークファイルの管理を行なうものである。ファイルのアクセス部分はホストに存在し、ファイルの読み書き及び創成、消却、検索などの機能を持つ。ＮＦＥＰ側のファイル転送サーバは、他系のファイル転送ユーザとの間のファイル転送のプロトコルのインタープリート、ホストのファイル処理プロセスとの会話機能を持つ。図４－３１にファイル転送サービスの概念図を示す。

ホスト側のファイルアクセス用のプロセスは、プロセス間通信機能を用いて、ＮＦＥＰ側のファイル転送サーバとの間でセッションを確立しておく。他系のファイル転送ユーザからＮＦＥＰのファイル転送サーバ

に対してファイル転送の要求があると、ホストのファイルアクセス用プロセスへ連絡する。これにより1つのセッションが1人のファイル転送ユーザ専用となるので、ファイルアクセス・プロセスはさらに別のセッションを用意してファイル転送サーバとの間で論理リンクを確立する。これによりユーザ単位に独立したファイル転送をサポートする。

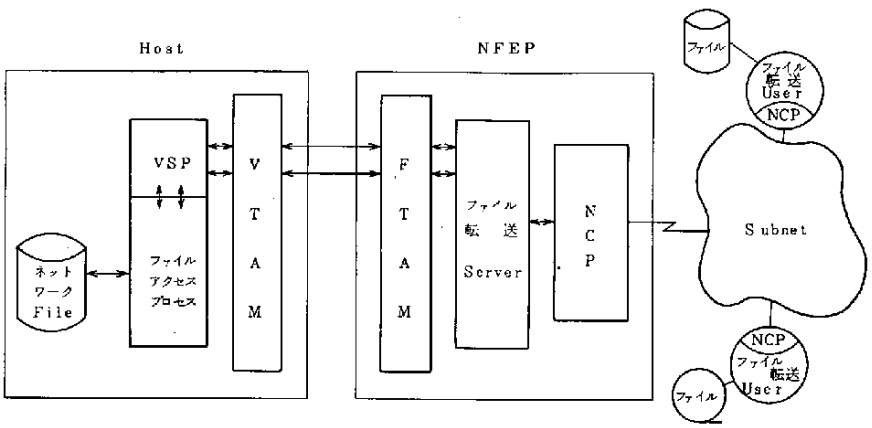


図4－31・ファイル転送サービス

(5) ネットワーク高位プロトコルの利用方法

ネットワークにつながる他系ホストのＴＳＳ、リモートバッチ、ファイル転送等の高位のプロトコルを利用するには、各ユーザ処理部分をホスト内に持ち、ＮＦＥＰ側にプロトコル処理を行なうプロセスを持つ。

ホスト内のユーザ処理プロセスは、プロセス間通信機能を用いて、ＮＦＥＰのＴＳＳユーザ、リモートバッチユーザ、ファイル転送ユーザとの会話を行なう。ユーザ処理プロセスの形態は、他系ＴＳＳを利用したい時には、ホストのＴＳＳを用い、ＴＳＳジョブがユーザ処理プロセスの機能を持つジョブとなり、リモートバッチ、ファイル転送を利用する場合にはバッチジョブが一般的な利用法である。ＮＦＥＰのユーザとホスト側のユーザ処理プロセス間のセッションは、ホスト側のプロセスからの要求によって確立される。図4－32にネットワーク高位のプロト

コルの利用概念を示す。

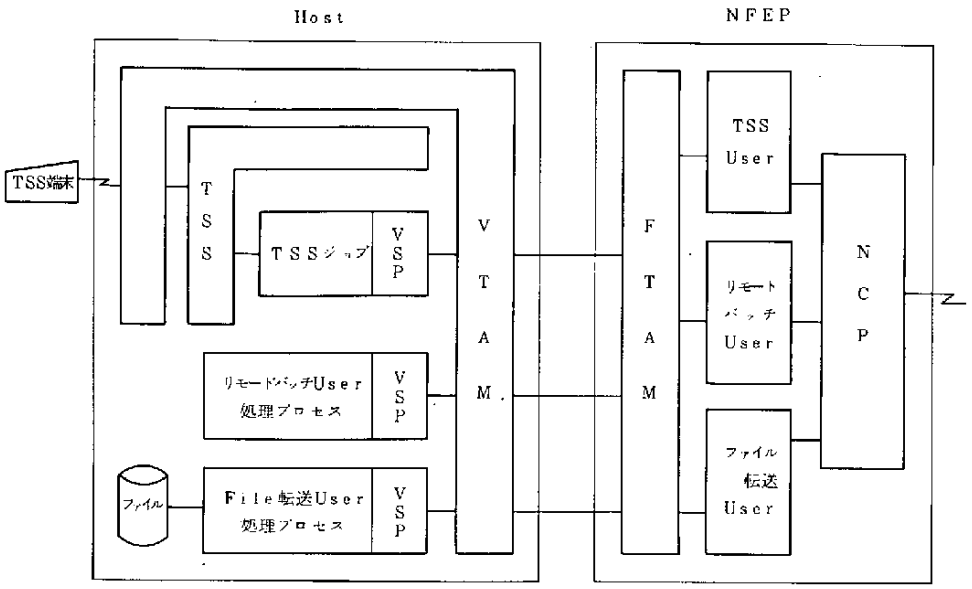


図４－３２　高位プロトコルの利用

4.4.3　ＮＦＥＰの論理設計

ＶＴＡＭレコードモード通信を用いたＮＦＥＰの論理構成を図４－３３に示す。

図でホスト側のＴＳＳ、リモートバッチ及びＶＳＰ＋ユーザプログラムはそれぞれＮＦＥＰ内のＴＳＳ・Ｓ、ＲＢＰ・Ｓ、ＰＣＳとの間でセッションを確立する。これらはＶＴＡＭ通信ネットワークで定めるエンドプロセスであり、それぞれの間でＶＴＡＭ高位のプロトコルが実現される。

ホスト内のＴＳＳ、リモートバッチは、ホストのサポートする制御プログラムであり、ＶＳＰはユーザ・プログラムに付加されるＶＴＡＭマクロサービスプログラムである。

ＮＦＥＰ内のＴＳＳ・Ｓ、ＲＢＰ・Ｓ、ＰＣＳは、ＴＳＳサービス、リモートバッチサービス、プロセス間通信サービスを行なうタスクであり、

ＮＦＥＰ内のＦＴＡＭとＮＣＰとの間で、各サービス機能を実現するものである。FTAMは回線制御プログラム（ line program ）とLCUP(local communication user program ）から構成される。

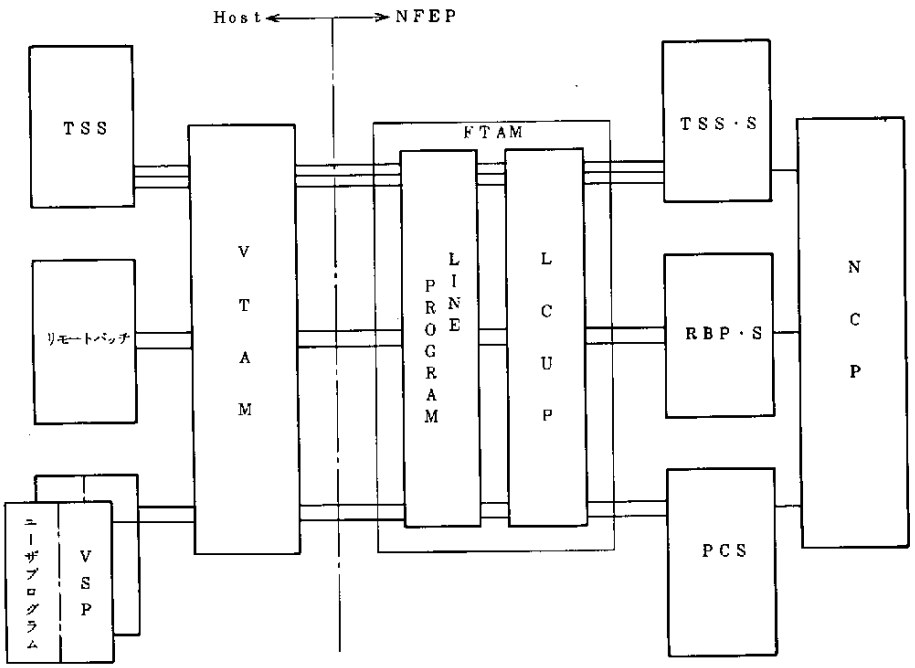


図４－３３　ＶＴＡＭ用ＮＦＥＰの論理構成

ＴＳＳ、リモートバッチ、ＶＳＰとＶＴＡＭとのインターフェースはＶＴＡＭマクロを用いる。ただし、ＴＳＳ及びリモートバッチに関しては各制御プログラムのサポート形態に従い、ＶＳＰとＶＴＡＭとの間は、ユーザがインプリメントする必要がある。

ＶＴＡＭとの間は、ＨＤＬＣ手順で結合され、ＶＴＡＭ側が一次局、回線制御プログラム側が２次局の制御手順を持つ。

回線制御プログラムとＬＣＵＰは共同して作業を行ない、全体としてＦＴＡＭ機能を有する。また回線制御プログラムとＬＣＵＰの間は、ベーシックなコマンドによって通信を行なう。

ＬＣＵＰとＴＳＳ・Ｓ、ＲＢＰ・Ｓ、ＰＣＳの間はＬＣＵＰサービス・コマンドによってアクセスされ、両者の間で、セッション番号、レスポンの可否及び指令、ユーザ・データ等を指定する事ができる。

ＴＳＳ・Ｓ、ＲＢＰ・Ｓ、ＰＣＳとＮＣＰとの間はＮＣＰサービス・コマンドがサポートされる。

(1) ＶＴＡＭとＦＴＡＭのインターフェース

ＶＴＡＭとＦＴＡＭ（実際にはＦＴＡＭ内の回線制御プログラム）は、４８ＫＢＰＳの通信回線１本で結合され、伝送制御手順はＨＤＬＣを用いる。この時ＶＴＡＭ側をＨＤＬＣ一次局、ＦＴＡＭ側を２次局とする。

１本の物理回線は複数のセッションとして用いられ、各セッションは論理的に独立したパスを持つ。

ＶＴＡＭとＦＴＡＭ間のセッション数、セッションを流れるメッセージの最大長及びバッファ（トラフィック制御の為のペーシングで使用する）数はシステム・ジェネレーションの時点で定義される。これらの値はそれぞれ以下の通りとする。

① エンドプロセス間で確立するセッションの最大数は同時には２０セッション迄とする。

② セッションを流れるメッセージの最大長は１１５２＋２バイトとする。２は、ヘッダ及びメッセージ付加情報である。

③ １セッション当りのバッファは２面迄とする。

(2) セッションとコネクション

１つのセッションに対して１プロセスを対応づけるか１コネクションを対応づけるかを考えると、以下の長短がある。

① １セッション１プロセス

＜長所＞・セッションを多くのコネクションで利用できるので有効である。

・ＮＦＥＰ内のＰＣＳの処理が簡単である。

・プロセスの動きを一括管理できる。

＜短所＞・ＮＣＰ機能の一部をホスト側に持つ事になる。すなわちコネクションの管理が必要となり、ＶＳＰの構造が複雑となる。

・セッション中を流れるデータのデータ・フロー、バッファ制御等が複雑となり、データの中にコネクション情報が必要となる。

② 1セッション1コネクション

＜長所＞・セッション中のデータがコネクションを通したデータと一致するので、セッションの制御が簡単である。

・ホスト側のＶＳＰの構造が簡単である。

＜短所＞・ＮＦＥＰ内でのＰＣＳの処理が複雑となる。

・セッションの数が少ない場合には、確立できるコネクション数に制限が生じる。

これらの事がらとＮＦＥＰの目的とを考慮すると、1セッション1コネクションの形態が望ましい事が分る。なお現構成ではホスト－ＮＦＥＰ間のセッション数は最大２０本、ＮＣＰの扱うコネクションは１６本である。

(3) 通信方式

ＶＴＡＭの持つ通信方式にはブラケット通信、方向変換通信、静止通信があるが、ＮＦＥＰを実現する時考えられるのは、方向変換通信、静止通信である。以下に各通信方式へのアプローチを示す。

① ブラケット通信

1セッション1コネクションである為、コネクションが確立されてから切断される迄の間同一セッションを用いる事となり、このセッションは他のユーザが使用する事はできないのでブラケット通信は取扱わない。

② 方向変換通信

トラフィックが片方向にかたよる場合に使用する。また会話形処理の場合にも利用できる。いずれの場合にもユーザ・プログラムの指定により行なう事とする。

③ 静止通信

大量データの一括転送の場合に使用する。メッセージ長が１１５２バイトにすれば、静止通信の必要性はあまりない。内部処理としては、ユーザが指定した場合に行なうか、もしくはメッセージを出力する時同一セッションに出力するメッセージキューがある場合にチェインメッセージとする事が考えられる。

(4) メッセージ転送とレスポンス

メッセージ転送時にレスポンスをかえす事を指定するか否かはＶＴＡＭ高位プロトコルによってエンドプロセス間で任意に規定する事ができるがＮＦＥＰ実現の為には以下の項目を原則とする。

① ユーザデータ及び標識を含むユーザデータに関して

- メッセージ処理後の応答は扱わない。すなわちFME（Function Management End ）の取扱いは、正常、例外ともに扱わない。
- メッセージ転送の正常ＲＲＮ応答（Reached Recovery Node ）は、ユーザが指定する事によって可能とする。
- メッセージ転送の例外ＲＲＮ応答は、システムでサポートする。この例外ＲＲＮ応答は、受信バッファの無い時、シーケンスエラーの時等に応答する。

② 指令に関して

- 同期データフローの場合には応答をかえす事を前提とする。
- サポートされない指令、サポートされる指令等により応答の種類を定める。（例えばＦＭＥ、ＲＲＮ等で分類する）
- 非同期データ・フローについては特殊な処理を考える。

上記の原則は、メッセージに対するレスポンスがユーザデータに関してはあまり必要でない事を示している。これは、実際のホスト－NFEP間の伝送誤りがほとんどなく、しかもＨＤＬＣで吸収されている事から考えて当然と言える。指令に関しては、その確認が必要な場合が多いのでレスポンスを原則として返す事が要求される。

(5)　ホストとＮＦＥＰ間のデータトラフィック制御

ホストとＮＦＥＰ間のデータ転送はセッション毎に行なわれる。各セッションを流れるデータ・トラフィックはＶＴＡＭのペーシング機能によって制御する事ができる。これは１番ベーシックなレベルでトラフィック制御となる。

次にホストとＮＦＥＰの各ソフトウェアの間でデータ・トラフィックの制御を考える。一般にホストコンピュータは記憶領域が大きく、バッファ量の心配をする事はないが、ＮＦＥＰはミニコンピュータであり、内部に持つバッファにも制限があるので、バッファ利用効率を考慮に入れたデータ・トラフィックの制御をする必要がある。

ホストからＮＦＥＰにデータ転送する場合には、ＶＴＡＭ／回線制御プログラム、ＬＣＵＰとプロセスとの関係を重視する必要があり、ＮＦＥＰからホストにデータ転送をする場合には、プロセスとＮＣＰとの関係を考慮しなければならない。

①　ホストからＮＦＥＰへのデータ転送

ＮＦＥＰ内のＬＣＵＰがプロセスへメッセージ到着を連絡し、メッセージバッファを得る方式を用いる。

ホストからメッセージを受け取った時、ＬＣＵＰに１バッファ以上あるかどうかを調べ無い時にはエラー・応答をかえす。１バッファ以上ある場合には、メッセージを受け取った事をプロセスに連絡する。プロセスでは交換バッファを確保してＬＣＵＰに連絡する。

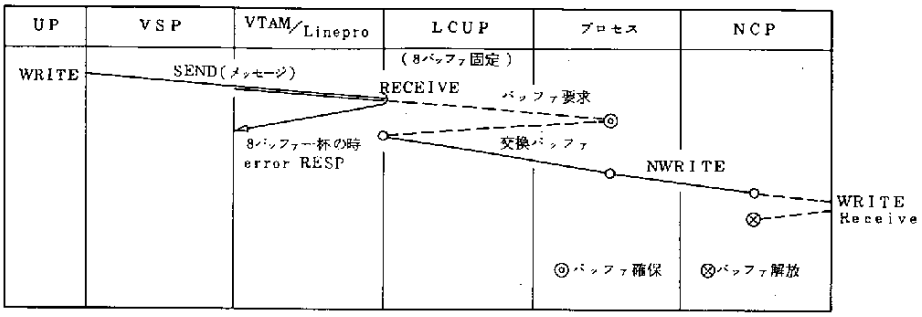


図４－３４　ホストからＮＦＥＰへのデータ転送

② ＮＦＥＰからホストへのデータ転送

ＮＣＰよりメッセージ到着が知らされた後ＮＣＰに対してＮＲＥＡＤサービス・コマンドを出す。

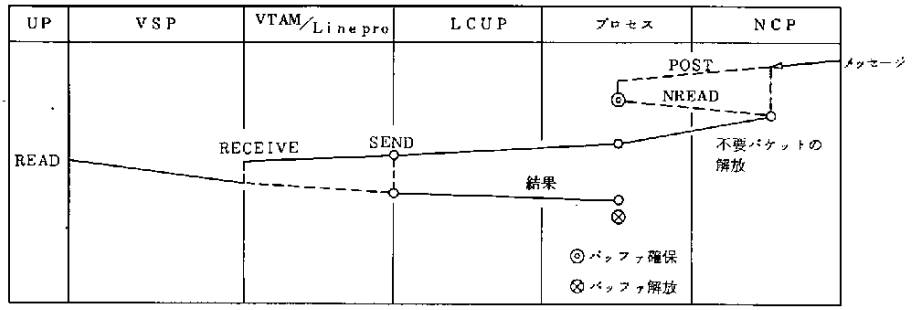


図４－３５　ＮＦＥＰからホストへのデータ転送

メッセージが他系より到着すると、ＮＣＰはプロセスに対してメッセージ受信をポストする。プロセスはバッファを確保しＮＣＰに対してＮＲＥＡＤを出す。ＮＲＥＡＤの終了がポストされるとプロセスは、ＬＣＵＰを通してホストにデータ転送を行なう。データ転送の終了がポストされると、プロセスはバッファを解放する。（ＬＣＵＰでバッファ解放を行なう場合にはプロセスでバッファ解放を行なう必要はない）

4.4.4　ＶＴＡＭ高位プロトコル

ＶＴＡＭ高位プロトコルはエンドプロセス間で定められるものであるが、ＮＦＥＰの実現にあたっては、ホスト側とＮＦＥＰ側で多少事情が異なる。ホスト側のエンドプロセスはホストのＯＳによって既にサポートされているＴＳＳとリモートバッチがあり、それ以外にネットワーク特有なものとしてＶＳＰが考えられる。ＮＦＥＰ側のエンドプロセスは、各種サーバタスクが考えられるが、それらが一般的にサポートするＶＴＡＭ高位プロトコルについては、その機能を標準的に１つのタスクが取り扱い、各業務個有な規則のみを各サーバにサポートさせる事が望ましい。

以下にプロトコルの階層及び高位プロトコルの設定について述べる。

(1)　プロトコルの階層

プロトコルは図４－３６の様な階層を持つ。

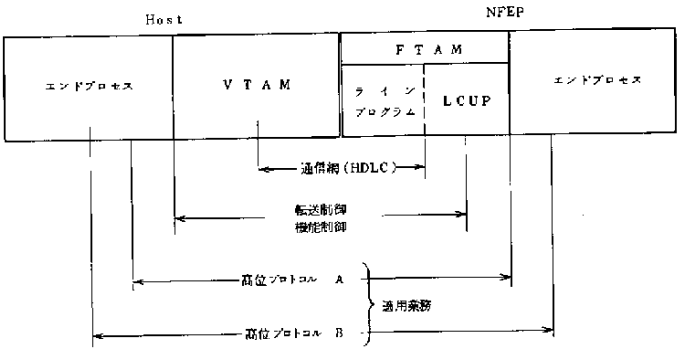


図４－３６　プロトコルの階層

ＶＴＡＭとラインプログラムの間はＨＤＬＣに関するサービスを行なう。このサービスでは、ＨＤＬＣ手順の全ての機能をサポートする。

ＶＴＡＭとＬＣＵＰの一部の間では、転送の宛先を選択するネットワーク制御及び宛先とのセッションを確立するセッション制御機能と（転送制御）データ転送の優先順位、通信網の資源の管理、端末情報の吸収(機能制御）が行なわれる。

エンドプロセス間ではＶＴＡＭ高位プロトコルが設定され適用業務処理が行なわれる。ＶＴＡＭ高位プロトコルはＮＦＥＰ側のインプリメントを考慮に入れて高位プロトコルＡとＢを設ける。

(2) 高位プロトコルＡ

高位プロトコルＡはＮＦＥＰ側のＬＣＵＰとホスト側のエンドプロセスとの間で定義される規約であり、エンドプロセス間の標準的なプロトコルを規定する。ＮＦＥＰ内においては、ホストのＴＳＳ、リモートバッチの持つプロトコルの標準部分と、プロセス間通信における標準的なプロトコルを処理する為の機能をＬＣＵＰに持つ。

高位プロトコルＡで定める規約は以下のものが考えられる。

① シーケンス・チェック：シーケンシングされたユーザデータのシーケンスチェックを行なう。

② バッファ管理：全セッションに割当てられたバッファの制御を行なう。

③ 応答制御：指令に対する応答の制御を行なう。

④ 非同期メッセージの制御：非同期に発生するメッセージの制御を行なう。

⑤ セッションの確立、開放の管理：セッションの確立時、終了会話、開放の制御及びセッション割当てのチェックを行なう。

⑥ チェーンメッセージ処理：チェーンメッセージの作成、制御処理を行なう。

⑦ 異常状態の検出と連絡：各種メッセージ転送中の異常、バッファ管理からの異常通知に対する処理を行なう。

(3) 高位プロトコルＢ

高位プロトコルＢはすでに述べた高位プロトコルＡを基礎として、ＮＦＥＰ側のエンドプロセスとホスト側のエンドプロセスとの間で定義される規約であり、各エンドプロセス間でユニークなプロトコルである。

一般に高位プロトコルＢでは、ユーザの要求によって通信方式を指定したり、応答が必要な時の指示を行なったりする為に使用される。

ほとんどのＶＴＡＭ高位プロトコルは先に述べたプロトコルＡでサポートされるが、特殊な規約を設ける時及び、既存のＴＳＳ、リモートバッチとのインターフェースを取る必要がある時等に用いる。

ＮＦＥＰ側のエンドプロセスは、ホスト側のＶＳＰエンドプロセスに対してはＰＣＳタスク、ＴＳＳに対してはＴＳＳ・Ｓタスク、リモートバッチに対してはＲＢＰ・Ｓタスクが対応し、それぞれの個有な処理に対する取り決めを行なう。またＶＳＰ－ＰＣＳの特殊な例として、ホスト側のファイルアクセス処理＋ＶＳＰに対してＮＦＥＰ側にはＦＴＰ・Ｓがある。

高位プロトコルＢで定める規約としては以下のものが考えられる。

① 応答指定：ユーザの要求により応答の種別、可否が記述された時の処理

② 通信方式指定：ブラケット通信、方向変換通信等をユーザが指定した時

③ 割込み信号の取扱い：非同期な割込み信号をユーザが指定した場合の処理

(4) ＴＳＳ、リモートバッチとエンドプロセス

ＴＳＳとＶＴＡＭ、リモートバッチとＶＴＡＭ間のインターフェースは、ＯＳによって定められていて、ＮＦＥＰからこれらを利用する場合には、一般のＴＳＳユーザ、リモートバッチユーザを使用すると同様なコマンドを用いてアクセスすることができる。すなわち、ＮＦＥＰ側からのＴＳＳ、リモートバッチに対するlogonコマンドにより１つのセッションを開設し、そのセッションを通じてデータの授受を行なう。またセッション中のデータトラフィック、メッセージフォーマット等に関してはＯＳがサポートするＴＳＳ、リモートバッチ機能にディペンドする。

ＴＳＳ、リモートバッチとの間のセッションは、ＮＦＥＰ内のＴＳＳ、ＲＢＰサーバが片側を受け持ち、ＴＳＳ、ＲＢＰはセッションに対して１つのコネクションを割り当てる。

これ等の事から、ＮＦＥＰ内のＴＳＳ、ＲＢＰサーバは、ホストのＴＳＳ、リモートバッチ機能を持つプロセスとの間でＶＴＡＭ高位のプロトコルを実現するが、そのプロトコルは既に決ったものであると言える。したがって、ＴＳＳ、ＲＢＰの各サーバはそのプロトコルに従ってインプリメントされなければならない。

(5) プロセス間通信とエンドプロセス

プロセス間通信は、ホスト側のユーザプログラムからの要求がＶＳＰを仲介プロセスとしてＮＦＥＰのＰＣＳを利用して他系プロセスへの連絡という手順で実現される。この時ホスト側のＶＳＰとＮＦＥＰ側のＬＣＵＰ及びＰＣＳとの間でＶＴＡＭ高位プロトコルが規定される。

4.4.5 エンドプロセスの機能

ホスト側のエンドプロセスとしては、ＴＳＳ、リモートバッチの各制御プログラムと、ユーザプログラムにバインドするモジュールＶＳＰがある。ＴＳＳ、リモートバッチの各制御プログラムはホストのＯＳによってサポートされ、ＮＦＥＰ内のＴＳＳ・Ｓ、ＲＢＰ・Ｓと協同してＴＳＳ、リモートバッチサービスを行なう。ＶＳＰはユーザが作成するプロセス間通信用プログラムまたは、ファイルアクセスプログラムに付加するモジュールで全体として１つのプロセスを形成し、ＮＦＥＰ内のＰＣＳまたはＦＴＰ・Ｓと協同して、プロセス間通信サービスおよびファイル転送サービスを行なう。また、ホスト側から他系ＴＳＳ、リモートバッチ、ファイル転送等の機能を利用しょうとする時には、プロセス間通信機能を用いたユーザ・プロセスをホスト内に用意する事により実現することができる。

ＮＦＥＰ側のエンドプロセスは、ＴＳＳサービスを行なうためのTSS･S

リモートバッチサービスを行なう為のＲＢＰ・Ｓ及びファイル転送サービスを行なう為のＦＴＰ・Ｓの各サーバがあり、プロセス間通信のサービスを提供するＰＣＳがある。図４－３７にその形態を示す。

サービス機能は、ネットワーク上の他系ホスト内のプロセスからの要求により、自系ホストの持つＴＳＳ、リモートバッチ及びファイル転送等を提供するものであり、利用機能は、自系ホストのユーザが、ネットワークのリソースを利用するためのものである。

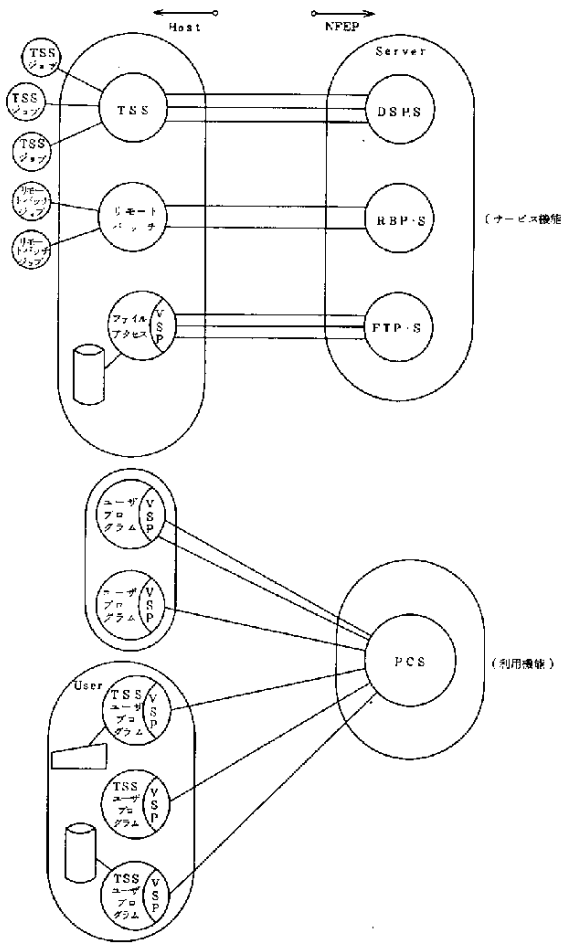


図４－３７　サービス機能と利用機能

ホスト側のエンドプロセスの機能を設定する場合、既存のＴＳＳ、リモートバッチ制御プログラムに関しては、そのものを利用するので、各マニアルを参照する必要がある。以下では、ユーザプログラムにバインドするＶＳＰの機能についてのみ考える。

ＮＦＥＰ側のエンドプロセスは、大きくサーバと、プロセス間通信とに分けられる。サーバの持つ機能のうちＶＴＡＭ高位プロトコルに関係するものは、相対するホストのＴＳＳ、リモートバッチに依存するので、各制御プログラムのプロトコル定義に従う必要がある。

以下にホスト側のＶＳＰの機能、ＮＦＥＰ側のサーバ機能及びプロセス間通信用プロセス（ＰＣＳ）の機能について述べる。

(1) ＶＳＰの機能

ホスト側のエンドプロセスは、次の機能を持つ。

① ユーザ・プログラムとのインターフェース処理

② ＶＴＡＭ高位プロトコル処理

③ ＶＴＡＭとのインターフェース処理

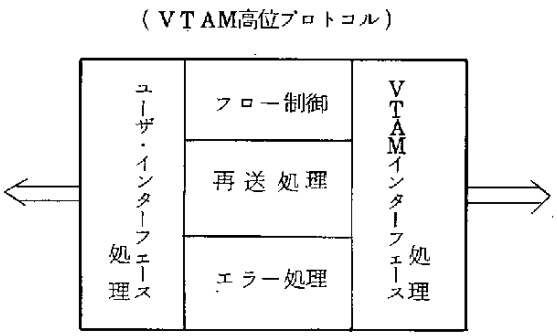


図４－３８　ＶＳＰの機能

ＴＳＳ、リモートバッチの各制御プログラムにおいても上記の機能は必ず持つが、ここではＶＳＰを主体にしてその機能を述べる。

① ユーザ・プログラムとのインターフェース機能

ユーザ・プログラムはバインドしたＶＳＰに対しマクロ指令の形で各種の情報の授受を行なう。このマクロ指令がユーザ・サービスコマンドである。ユーザインターフェース処理部分では、ユーザ・サービスコマンドの解析及び、ＮＦＥＰに送出するメッセージの組み立てを行ない、ＶＴＡＭインターフェース処理に制御を移す。また、ＮＦＥＰから送られてきたメッセージをユーザ・プログラムに渡すための処理機能を持つ。

② ＶＴＡＭ高位プロトコル処理

ＶＴＡＭ高位プロトコルの項で述べたプロトコル処理を行なう。その主なものは、フロー制御、再送処理、エラー処理等である。この処理内容は、ＮＦＥＰ内のＰＣＳ及びＦＴＰ・Ｓとの間で定められた規約にもとずく。

③ ＶＴＡＭとのインターフェース処理

ユーザからの要求がユーザインターフェースで解析され、ＮＦＥＰへメッセージ転送が必要となった時、ＶＴＡＭマクロを作成し、ＶＴＡＭに対し連絡をとる。また、ＮＦＥＰより送られてきたメッセージの読み込みのためにもＶＴＡＭマクロを発行する。これ以外に、ＶＴＡＭ高位プロトコルの依頼に従って、セッション制御指令を出したり、セッションの確立、開放等の処理をＶＴＡＭに要求する。

ＶＴＡＭインターフェース処理内では、非同期に生じるメッセージを受け付ける為にＶＳＰが使用するｅｘｉｔルーチン等の制御も行なう。

(2) ＮＦＥＰ内サーバ機能

ＮＦＥＰのＴＳＳ・Ｓ、ＲＢＰ・Ｓ、ＦＴＰ・Ｓの各サーバは、次の機能を持つ。

① ＮＣＰとのインターフェース処理

② ネットワーク高位のプロトコル処理

③　ＶＴＡＭ高位プロトコル処理

④　ＬＣＵＰとのインターフェース処理

各サーバは独立したタスクとしてＬＣＵＰとＮＣＰとの間に存在する。図にサーバ機能の概念を示す。

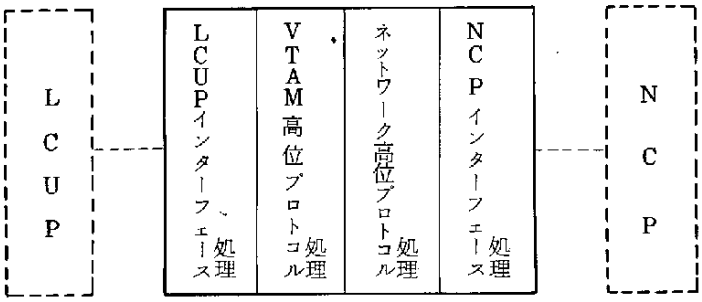


図４－３９　サーバ機能

①　ＮＣＰとのインターフェース処理

ＮＣＰサービスコマンドを用いてＮＣＰに対して連絡を行なう。ＮＣＰと連絡を行なう時にはコネクションを論理パスとして用いる。すなわち指定したコネクションに対して読み書きのコマンドを出す事によりデータの授受を行なう。

②　ネットワーク高位プロトコル処理

ネットワークの高位のプロトコルをサポートする。サーバ機能としてのＴＳＳ、ＲＢＰ、ＦＴＰのそれぞれについてプロトコルが定められており、それぞれのプロトコル処理、及びサーバが用いるコネクションの制御を行なう。

③　ＶＴＡＭ高位プロトコル処理

ホスト側のＴＳＳ、リモートバッチ及びファイルアクセスプログラムに付加されたＶＳＰとの間で、ＶＴＡＭ高位プロトコルが定められるのでそれに従った処理を行なう。ＴＳＳ・Ｓ、ＲＢＰ・ＳのＶＴＡＭ高位プロトコル処理はホストのＴＳＳ、リモートバッチ機能と関連

して決定される。FTP・Sの場合には、ホストに用意した標準的なネットワークサービス用のモジュール（ＶＳＰ）との間でプロトコル設定がなされる。

④　ＬＣＵＰとのインターフェース処理

ＬＣＵＰからのコマンドを解析し、ＶＴＡＭ高位プロトコル処理部へのデータの授受、ＮＣＰインターフェース処理への連絡などを行なう。また、タスクの要求によってＬＣＵＰを通じてホストにデータを送信する為に用いられる。ＬＣＵＰとのインターフェースではセッション番号を明記し、１セッションと１コネクションとの対応付け等も処理される。

(3)　ＮＦＥＰ内ＰＣＳの機能

ＮＦＥＰ内のＰＣＳは、プロセス間通信を行なう為のサービスを提供する。

ホストのユーザプログラムがプロセス間通信を行ないたい時、ホストのＴＳＳユーザがネットワークに接続されている他系のＴＳＳを使いたい時、ファイル転送を行ないたい時等に、ユーザプログラム＋ＶＳＰという形態をとってＰＣＳを通して他系プロセスとの間でデータの授受を行なう。ＰＣＳは、各種のユーザから同時に利用されるために、複数のコネクションとセッションの管理を行なう。ＰＣＳの機能は以下の通りである。

①　ＮＣＰとのインターフェース処理

②　コネクション管理

③　ＶＴＡＭ高位プロトコル処理

④　ＬＣＵＰとのインターフェース処理

このうち①と④は、サーバ機能で述べたとほぼ同様である。またコネクション管理は、ホストから要求のあった複数のセッションとコネクションの対応づけ及びその制御を行なう。ＶＴＡＭ高位プロトコル処理は、

ホストのＶＳＰとの間で定義されたＶＴＡＭ高位プロトコルの処理を行なう。このＶＳＰ－ＰＣＳ間のプロトコルは標準的な形としてサポートされる事が望ましく、ホストのユーザがいつでも簡単にネットワークへ参加する事が可能な様な形で設定されるべきである。

第４章の参考文献

1. David L Retz

   Speech Communications Research Laboratory , Inc , Santa Barbara , California

   " Operating system design considerations for the packet-switching environment " , NCC 75 , pp155 ～ 160.

2. Adrian V. Stokes , Peter L. Higginson

   Department of Statistics and Computer Science University College London , U. K.

   " The problem of connecting host into ARPANET " EUROCOMP 75 , pp. 25 ～ 33.

3. D. C. Wood , R. K. Trehan

   The MITRE Corporation USA

   " An assessment of the Performance of Packet Switching Networks " EUROCOMP 75 , pp. 1 ～ 11.

4. J. du Masle

   Ecole Nationale Superieure d Ingenieurs Mathematicien-de Grenoble

   " One year of experience with CYCLADES "

   Proceedings SHARE European Association 75 , Doublin

5. W. R. Crowther , F. E. Heart , A. A. Mckenzie , J. M. Mcquillan , D. C. Walden

Bolt Beranek and Newman Inc , Cambridge , Massachusetts
" Issues in packet switching network design "
NCC 75 , pp. 161 ～ 175

6. 電子計算機利用に関する技術研究会
" リソースシェアリングシステム（ネットワーク定量分析）"
昭和５１年３月、 p. 197

5．協同ジョブ処理(CJP)

5．協同ジョブ処理(CJP)

5.1　CJP

5.1.1　CJPの概要

A．CJPの目的

ネットワーク内のホスト・コンピュータ群を論理的な1つのコンピュータとして考え、複数ホストにいくつかの処理を分担し、1つの作業を複数ホストの協業並行処理により行おうというCJP（Cooperated Job Processing）をネットワークの利用という観点から考えてみる。

今までのネットワークの利用法は主に次の3つが上げられる。

① タイム・シェア・システムに対するリモート・アクセス

② 特色のあるリソースをもったホストのサブルーチン的利用

③ ファイル転送

リモート・アクセス利用法は商用タイム・シェアリング・ネットワークの利用法と似ている。例えば、ユーザはTIP（Terminal Inter Message Processor）端末を通してネットワークへアクセスし、ネットワーク上の多数のタイム・シェア・ホスト・コンピュータの1つを利用する。信頼性のために、ユーザは通常ファイルをいくつかの互換性のあるホストに保っておき、利用可能ないずれかのホストを使用する。2番目のネットワークの利用法は、例えば科学技術計算のアプリケーション・システムで仕事のインタラクティブな部分（プログラムの編集、データ入出力処理、表示）は、TSS専用コンピュータを使用して、大量の数値処理は大規模なリモート・バッチ・ホストにまかせるのである。アプリケーション・プログラムにとって、データの出し入れの操作は必須のものであるが数値処理の中心的仕事は、数値計算に大きな処理能力をもつホストにまかせると、はるかに安く、速く処理することができる。3

番目のネット・ワークの主な利用法はファイル転送である。前述の2つの利用法のためにファイルはホスト・コンピュータ間で非常に頻繁に転送し合わなければならない。また大きなデータやプログラムの全体をそのまま移動させるためにネットワークを使用することもある。

これらのネットワークの主な利用法のうち、2番目の利用法すなわち、〃アプリケーション・プログラムのうちのある部分を、その仕事に適したホストに分担する〃のがCJPの出発点である。CJPはこの利用法に、〃いくつかのホストに協力して並行作業させる〃という利用法を付加したものである。つまりコンピュータ・ネットワークの究極の目的である〃ネットワーク内の全リソースを統合された1つの仮想マシンと見なし、入力されたジョブが最適のホストまたは複数ホストで処理され、その結果が出力されるという高度なユーティリティであるべきだ〃という仮想マシンとしてのネットワークを実現するための1つのステップとしてCJPを位置づけることができる。このCJPを利用している例としてRANDのVIEWシステムを紹介しておく。

(1) VIEWシステム(注)

現在、RANDでは自然現象の調査、シミュレーション及び分析を行っている。このRANDの気象プロジェクトの中心課題は、グローバル気象を偶然かあるいは故意に変更させている人間の能力を定量的に分析することである。この課題を研究するために、気象プロジェクトではいくつかのシミュレーション・モデルを使っているが、現在もっとも重要なモデルはM/Aモデル(Mintz-Arakawa Two-Level Atomospheric Circulation Model)である。このM/Aモデルは地球全体の大気を表わすため2レベルの46×72の格子を使って気

(注) Eric F. Harslem and Suganne D. Landa, 〃VIEW: A distributed system for graphical analysis of large data bases,〃 Rand PP. 29, P-4972, March '73

象現象をシミュレートする。このモデルを使用するには、大量のＣＰＵ時間及びデータ記憶装置が必要であるが、RAND のメイン・コンピュータはIBM360/65である。360/65 でM/Aモデルを使って実際の1日をシミュレートするためには、４００Ｋバイトの高速記憶装置と5時間のCPU時間を必要とする。このため明らかに、年間規模のシミュレーションは実行不可能である。この問題を解決したのがARPAネットワークである。RANDはARPAネットワークの1つのサイトなので、ARPAネットを介してUCLAの360/91 のリモートジョブ・サービスを使用して大規模な計算のすべてを行った。360/91では実際の1日をシミュレートするのに30分のCPU時間しかかからず3ケ月間のシミュレーションも可能となった。しかし解析すべきデータが大量であるので実行が妨げられている。このデータ解析問題がＶＩＥＷ システムの開発をうながす動機となった。

① ＶＩＥＷシステムの目標

簡単に使用でき、オンラインで融通性のある表示フォーマットでグラフィック端末に動的にデータを絵にして表示できる能力をもつことである。

② ＶＩＥＷシステム設計の制約事項

- ネットワーク中の別の場所にいるユーザからアクセスできなければならない。
- ネットワーク中の別の型の端末からアクセスできなければならない。
- ネットワーク中の別のサイトにあるデータをアクセスできなければならない。
- 長いセッション時間とsporadic（時々起こる）なＣＰＵ使用が可能な環境（すなわち、バッチ環境ではない）が常時用意されていなければならない。

③　ＶＩＥＷシステムの構造

これらの制約事項を満足させるＶＩＥＷの構造を組み立てるために、それを３つの主要なセグメントに分けた。

- グラフィック端末Ｉ／Ｏセグメント
- データ検索セグメント
- ユーザ要求を受け付け、分析し、データ検索セグメントを呼んだり、データを表示するためにグラフィック端末Ｉ／Ｏセグメントを呼んだりするメイン・アナリシス・プログラム

ＶＩＥＷシステムをこれらのセグメントに分割すると、これらのセグメントをＡＲＰＡネットワーク中のＨｏｓｔに分散することによって制約事項が満足される。

図５－１にセグメントの可能な分散の例を示す。

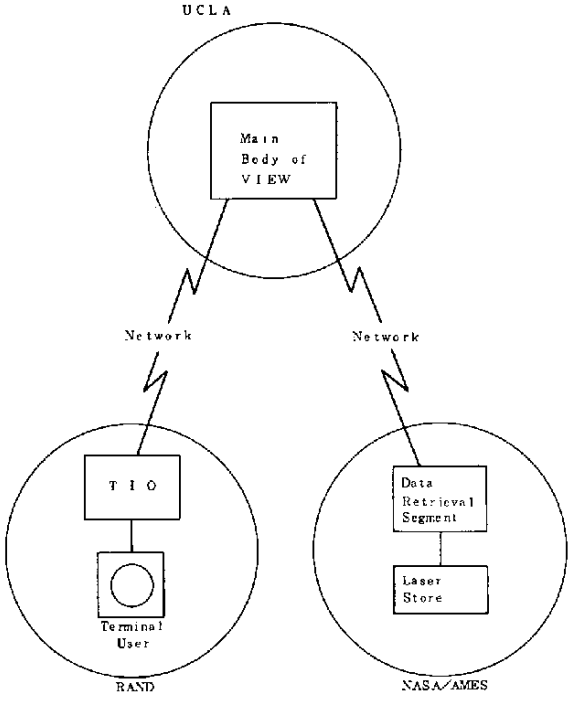
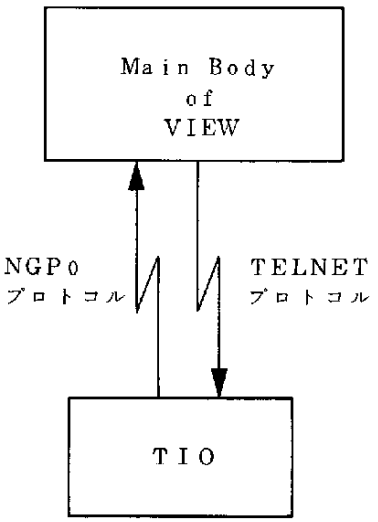


図５－１　ＶＩＥＷシステムの分散配置例

④ 端末アクセス

ネットワーク・プログラム・シェアリング思想の下では、プログラムは標準インタフェースを使って設計され、通常のコンピュータ環境にある間はリモートからアクセスされる。この目的で、端末からＶＩＥＷへの入力はネットワーク標準端末入力プロトコル（TELNET）に従い、端末への表示出力はネットワーク・グラフィック・プロトコル・レベル０（ＮＧＰ０）で規定されたオーダ・コードから成る。



NGP0：Network Graphic Protocol Level 0

ＴＩＯ：Terminal I/O

図５－２　Main Body／TIO プロトコル

B. ＣＪＰの定義

ここでＣＪＰとはどういうものか定義すると、『コンピュータ・ネットワーク内のホスト・コンピュータ群を、論理的に１つのコンピュータとして考え、複数のホストにユーザの指定するジョブをそれぞれ発生させ、それらのジョブ間の適当な同期をとり、コントロール及びデータの

交信をおこないながら、複数ジョブの協同作業によりあるまとまった処理をするもの』と言える。すなわちこれは、ネットワーク内の複数ホストにまたがるマルチ・タスクの実現にほかならない。

5.1.2 実験計画

A. 目　的

当協会にはＪＩＰＮＥＴ（ＪＩＰＤＥＣ Integrated Project Network）というリソース・シェアリング・コンピュータ・ネットワークがあり、プロセス間の同期をとるコマンドがユーザ・プロセスから利用可能なので、ＣＪＰの利用が容易である。また当協会のグラフィック・システムはＨ－８４５０をホスト、Ｈ－８８１１をサテライトとする２プロセッサ方式で稼動している。しかしアプリケーション・プログラムが巨大になる（例えば、隠れ線消去、ＰＥＲＴなど）につれ、ホストにかかる負荷が大きくなり、グラフィック・ジョブが動いている間は他のバッチ・ジョブが動かない場合さえあった。そこでＣＪＰを利用したグラフィック・システムを実現して、ＣＪＰを利用してこの問題を解決できるか、即ち〝ＣＪＰは有効か〟をさぐるのがこの実験の目的である。

B. 方　法

(1) 測定項目

① ＣＰＵ時間

ＣＪＰを利用してＣＰＵ時間、コストに関して効率をみるために、同一のアプリケーション・プログラムを、単独ホストで実行した場合のＣＰＵ時間と、ＣＪＰを利用したときの２つのホスト（FACOMとＨＩＴＡＣ）のＣＰＵ時間をメッセージ転送回数、ホップ数（通過する回線数）を変えて測定した。

② コア・タイム

シングル・ジョブで計算機を使用した場合のコア・タイムは、レ

スポンス・タイムの総和と考えることができるので、ＣＪＰを利用してレスポンス・タイムが速くなったかどうかを見るために、単独ホストの場合とＣＪＰを利用した場合のそれぞれのコア・タイムをメッセージ転送回数、ホップ数を変えて測定した。

(2) 実験手順

① ＨＩＴＡＣのグラフィック・システム単独での測定。

② ＣＪＰ利用時の測定。

ⓐ 実験は、図５－３に示す手順により行う。

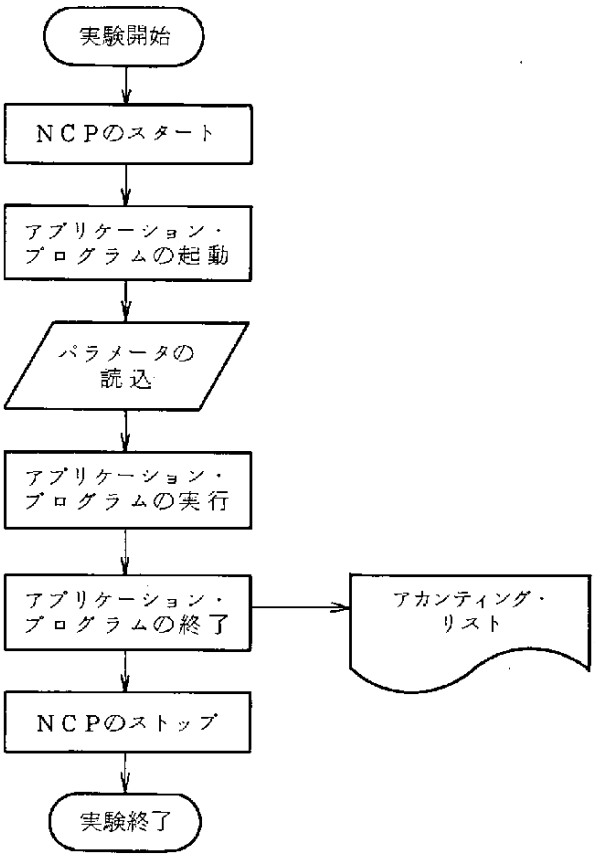
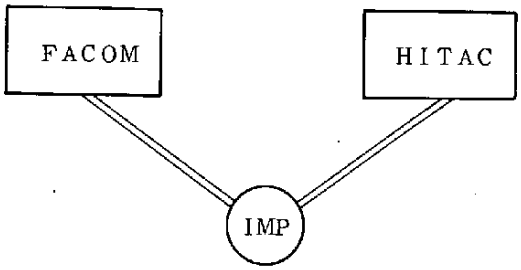


図５－３ 実 験 手 順

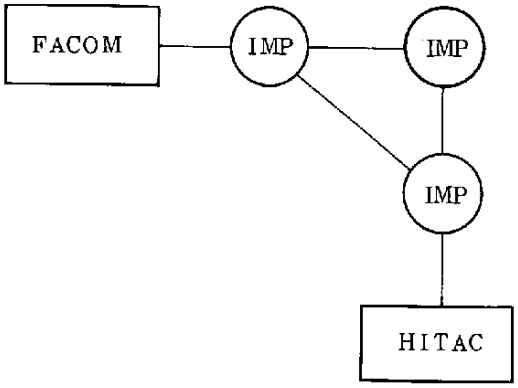
ⓑ　パラメータはメッセージ転送回数である。

ⓒ　アプリケーション・プログラムは、受信側のＨＩＴＡＣを先に、送信側のＦＡＣＯＭを後に起動する。

ⓓ　全ての転送回数に対する測定が終了したら、ホップ数を変えて（０－ｈｏｐ→ｆｕｌｌｃｏｎｎｅｃｔ）同じ手順、同じパラメータで測定する。



0－hop



fullconnect

図５－４　サブネット構成図

(3) 実験結果の集計

実験の結果は各ホストのモニターの集計するジョブ・アカント・リストから得る。

5.1.3 実験環境

A. グラフィック・システム

CJPを利用したグラフィック・システムのソフトウェア構成は、ネットワークに関するものとしてはFACOM、HITACそれぞれにNCP（Network Control Program）と、NCPインタフェース・プログラムが存在し、IMP（Interface Message Processor）にはIMPモニタが存在する。

またグラフィックに関するものとして、H−8450内にDXC I／Oインタフェース・ルーチン（DXC：データ交換装置）、グラフィック・システムを利用するユーザに対し、図形データを記述する上での便宜をはかるために用意されたサブルーチン・パッケージである、GSP（Graphic Subroutine Package）がある。グラフィック・サテライトコンピュータH−8811内には、ディスプレイ・コマンド・コンパイリング、ホストとの相互のデータ転送、そしてCRT制御などをつかさどるGCP（Graphic Communication Program）が存在する。

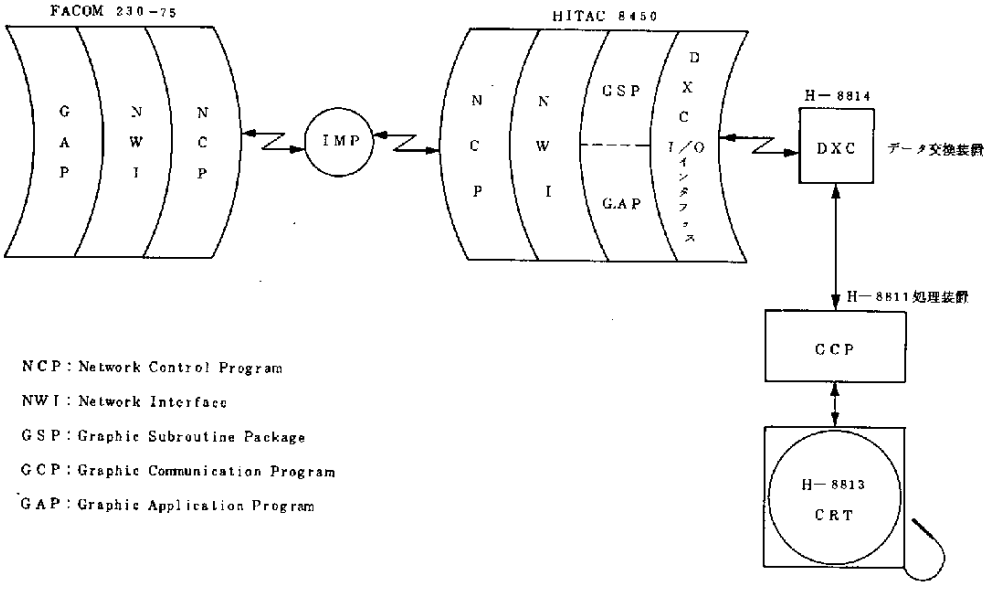


図５－５　CJPを利用したグラフィック・システムのソフトウェア構成

B. アプリケーション・プログラム

実験に使用したグラフィック・アプリケーション・プログラムの概略を紹介する。

このアプリケーション・プログラムは２変数関数の立体表示プログラムで、３次元である２変数関数を、２次元に投象して表わす。その際、立体的に見えるように、関数の曲面自身によって視点から見えない線は消去（Hidden Line Elimination）してある。曲面はメッシュによって表わされメッシュ間は直線で結ばれる。

ここで表わされる関数は、次の条件を満さなければならない。

(1) 関数Z（x，y）は一価関数であること、即ち、（x，y）に対してただ１つの値をもつ。

(2) 関数は、FUNCTION SUBROUTINE としておく。

(3) 関数の値の絶対値が無限大になるときには極限値を与えるようにFUNCTION SUBROUTINE を作成する。

このアプリケーション・プログラムの特徴は、

1) 隠れ線消去をほどこしてあるので、1つのインタラクションに対して、CPUを非常に多く消費し、レスポンス・タイムが長い。

2) プログラム全体を2つ（図形処理と演算処理）のモジュールに明確に分割できる。

3) 分割された2つのモジュール間で、データのやりとりがある。

4) 主に演算処理をするメイン・プログラムはFORTRANで、図形データを取り扱ったサブルーチンは特定のアセンブラで組まれている。

つぎに、実験にあたっては、このアプリケーション・プログラムを図形処理と演算処理に分割した。すなわち、演算処理をFACOMに、図形処理をHITACに分割して2つのホストの協業並行処理によってこのアプリケーション・プログラムを実行した。

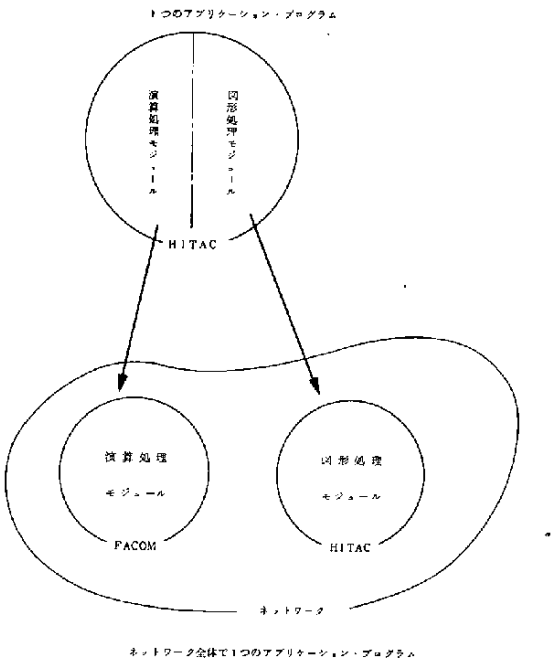


図5－6　アプリケーション・プログラムの分割

このアプリケーション・プログラムの実行結果の出力例が図５－７である。

$$Y = \frac{SIN\ X}{X}$$ の回転体

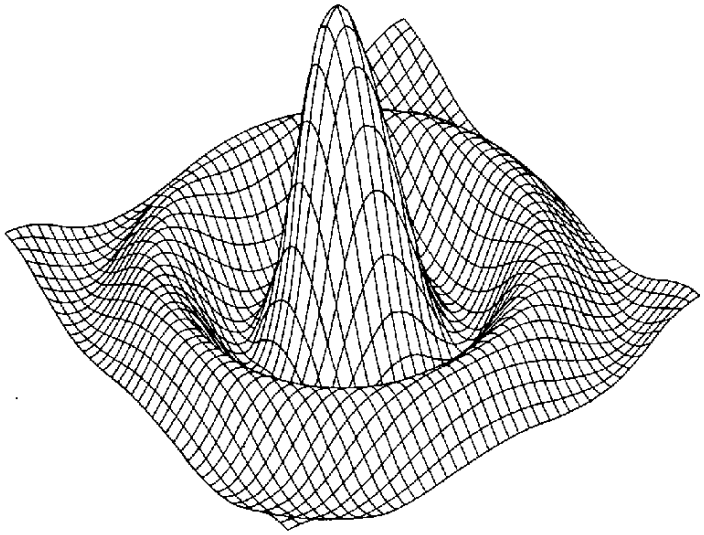


図５－７　出　力　例

5.2　結果の解析と評価

5.2.1　結果の推定

今回行ったＣＪＰの実験は、グラフィック・システムを用いたが、実験の前にＣＰＵ時間、コスト、レスポンス・タイムの効率についての一般式をたて、これに今回の実験の条件をあてはめて結果の推定をした。

実験の結果が推定通りならば、この一般式は、実験がグラフィック・アプリケーション・システムによるものという特殊な条件であることを差しひいてもある程度一般的な評価式になると言える。

ここで、アプリケーション・プログラムを単独で実行していたホストを『メイン・ホスト』、モジュールを分割されるホストを『サブ・ホスト』

と定義する。

A. 一般式

(1) CPU時間

CPU時間に関して、CJPを利用した方が有効な場合とは各ホストは互いに同期をとるとして一般に次の式が満足される場合である。

$$\sum_{m}\left(T'm+\sum_{j}Tmj\right)<\sum_{i}Ti \qquad (1)$$

T ： アプリケーション・プログラムの各モジュールの処理に要するCPU時間。

T′ ： メッセージ転送処理に要するCPU時間。

m ： ホストID。

j ： モジュールID。

i ： メイン・ホストにおけるモジュールID。

mj ： ホストmに分割されたモジュールjを表す。

CJPのCPU効率に影響を与えると思われる要素を整理すると次の4つが考えられる。

① メイン・ホストとサブ・ホストの処理速度の違い。

② アプリケーション・プログラム全体のCPU時間。

③ 各ホスト間でのメッセージ転送回数。

④ 分割するモジュールの、アプリケーション・プログラム全体に対するCPUを消費する割合。

— 注1 — ホスト－ホスト間のメッセージ転送について

メッセージ転送回数を変えないで（1000回に固定）、転送メッセージ長を変えてFACOMからHITACの転送を行った結果が表5－1である。

メッセージ長を固定し、メッセージ転送回数を変えてFACOMからHITACのメッセージ転送を行った場合のHITACの結果が表5－2である。

表５－１　F→Hのメッセージ転送に要するＣＰＵ時間(1)

1000メッセージ Long Pipe, fullconnect

メッセージ(バイト)長	送信側（FACOM）			受信側（HITAC）		
	NCPのCPU (sec)	User Processのcpu (sec)	NCP+U.PのCPU (sec)	NCPのCPU (sec)	User Processのcpu (sec)	NCP+U.PのCPU (sec)
144	22.601	0.1	22.701	14	9	23
288	22.476	0.1	22.576	14	9	23
432	23.082	0.1	23.182	14	9	23
576	22.588	0.1	22.688	15	8	23
720	22.984	0.1	23.084	14	10	24
864	23.117	0.1	23.217	14	10	24
1008	22.620	0.1	22.720	14	9	23
1152	22.908	0.1	23.008	14	10	24
平均	22.797	0.1	22.897	14.125	9.25	23.375

USER PROCESS：メッセージ転送処理を行うプロセス

表５－２　F→Hのメッセージ転送に要するCPU時間(2)

0-hop

メッセージ転送回数	1000（回）		2000（回）	
CPU時間(sec) / メッセージ長(byte)	NCP	USER PROCESS	NCP	USER PROCESS
144	14	9	29	17
576	15	8	28	18
1152	14	10	28	20

表５－１のユーザ・プロセスのＣＰＵ時間をみると、メッセージ転送にかかるＣＰＵ時間は、転送メッセージ長を変えても一定であることが分かる。

また表５－２からメッセージ転送回数が増えるとＣＰＵ時間も増えることがわかる。

これらの要素を考慮して、(1)式を変換する。T′（メッセージ転送に要するCPU時間）は、メッセージ送信と受信の回数に正比例するので送信回数をTMm、受信回数をRMmとし、ホストmの送信、受信オーバヘッド定数をそれぞれδm、σmとすると、

$$\sum_m T'm = \sum_m (\delta m\ TM m + \sigma m\ RM m)$$

メイン・ホストでのモジュール数とその処理が、CJP全体でのモジュール数、処理と同一であるとすると、

$$\Sigma mj = i$$

アプリケーション・プログラムの各モジュールのCPU消費の割合をα、アプリケーション・プログラム全体のCPU時間をToとすると、

$$\alpha i = Ti / (To - Ti)$$

メイン・ホストの処理速度（Po）と、サブ・ホストの処理速度（Ps）の比をr（$r = Ps / Po$）とすると、各ホストに分割されたモジュールのCPU時間TmiとTiの関係は、

$$Tmi = rm\ Ti$$

また、Toと各モジュールのCPU時間Tiの関係は、

$$Ti = \frac{\alpha i}{1+\alpha i} To$$

よって、

$$Tmi = rm \frac{\alpha i}{1+\alpha i} To$$

以上より(1)式は、

$$\sum_m \left\{ \delta m\, TM m + \sigma m RM m + \sum_i \left(rm \frac{\alpha i}{1+\alpha i} To \right) \right\} < \sum_i \frac{\alpha i}{1+\alpha i} To$$

よってＣＰＵ時間に関してＣＪＰを利用した方が有効な場合とは一般に次の式を満足する場合である。

ＣＰＵ時間評価式

$$\sum_m (\delta_m \mathrm{TM}_m + \sigma_m \mathrm{RM}_m) < \mathrm{T}_o \sum_m (1-\gamma_m) \sum_i \frac{\alpha_i}{1+\alpha_i} \qquad (2)$$

TM：メッセージ送信回数

RM：メッセージ受信回数

δ　：メッセージ送信オーバヘッド定数

σ　：メッセージ受信オーバヘッド定数

T_o：メイン・ホスト単独でアプリケーション・プログラムを実行したときのＣＰＵ時間

m　：ホストＩＤ

i　：モジュールＩＤ

γ　：処理速度比（サブ・ホスト処理速度／メイン・ホスト処理速度）

α　：分割モジュールのアプリケーション・プログラム全体に対するＣＰＵ消費割合

ただし(2)式は次の条件を満足しなければならない。

$$0 < \sum_m (\delta_m \mathrm{TM}_m + \sigma_m \mathrm{RM}_m) < \frac{\mathrm{T}_o \alpha}{\alpha+1} \quad (\text{このとき } \gamma > 0)$$

(2)　コスト（コンピュータ使用料金）

コンピュータ使用料金は、通常モニタが集計しているジョブ・アカンティングによって計算される。アカンティングの対象となるものとしては、ＣＰＵ使用時間、主記憶装置占有時間、同占有量、ファイル使用量、チャネル使用回数、入力カード枚数、出力行数、ジョブの優先権などがある。これらの項目に関して、すべての計量しているものもあるし、一部しか計量しないものもある。また、ＣＰＵ使用時間に関

しては、ある程度の誤差があるばかりでなく、ユーザがモニタのサービスを受けている時間（ファイルへのアクセスなど）をユーザの使用時間に加算する場合としない場合がある。この意味において、アカンティング情報は、各ホストにおいて、同じ名前でも内容が違っていることがある。

このため各ホストにおいては、これらの情報をもとにして、コンピュータ使用料金を計算する場合に、どの項目にどれだけ重みをかけるかは、独自に決めている。したがってコンピュータの使用料金は、きわめて相対的なものである。

これらのことを前提として、コストに関してＣＪＰを利用した方が有効な場合とは、次の式を満足するときである。ただし、ネットワークの使用料金は各ホストごとに分散しているものとする。

コストの評価式

$$\sum_{m}\left(\sum_{mj} W_{mj}\, R_{mj}\, S_{mj} + U_m\right) < \sum_{i} W_i\, R_i\, S_i \qquad (3)$$

m ： 使用ホストＩＤ

i ： メイン・ホスト使用リソースＩＤ

j ： 使用リソースＩＤ

W ： リソース使用料金定数

R ： 各種リソース

S ： リソース使用時間

U ： ネットワーク使用料金

(3)式において各種リソースとは、中央処理装置、主記憶装置、補助記憶装置、Ｉ／Ｏチャネル、特殊入出力機器（ＸＹプロッター、ＯＭＲなど）、回線など使用料金の対象となるすべてのリソースをいう。

(3) レスポンス・タイム

現在の計算機はマルチ・プログラミング・マシンが主流を占めてい

る。これはある一定時間に複数のジョブを並行に処理している。図５－８は、このマルチ・プログラミング・マシンでの計算機リソースについてのタイム・チャートの概略である。

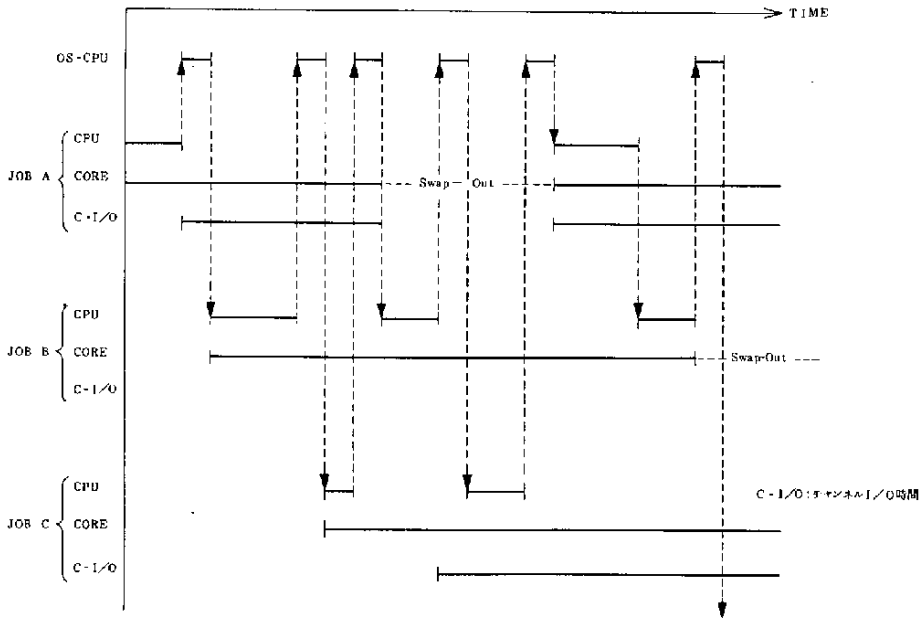


図５－８　マルチ・プログラム・マシンにおける
リソース利用のタイム・チャート

図５－８からわかるように、マルチ・プログラミング・マシンではＯＳはジョブを処理する場合、複数ジョブを効率よく処理するためにコンピュータのリソースを少しずつ順番に割り当てることを考える。このためコア・タイムにはそのジョブ（ＪＯＢ Ａ）が主記憶装置の中に入っていれば、他のジョブ（ＪＯＢ Ｂ）がリソースを占有している時間（すなわちＪＯＢ ＡにとってのＣＰＵ待ち時間）も含まれる。ただしスワップ・アウト時間は含まれない。

よって、そのジョブがコンピュータ・リソースを使用した時間を知るためには、シングル・ジョブでコンピュータを使用したときのコア

・タイムを見ればよい。レスポンス・タイムを〝ジョブが主記憶に入ってから出るまでの時間〟と考えると、シングル・ジョブでのコア・タイムはレスポンス・タイムに等しいと考えることができる。

このレポートでは

レスポンス・タイム＝コア・タイム

として話を進める。

モニターの集計するジョブ・アカンティング・リストのコアタイムは一般に次のように集計されている。

コア・タイム＝CPU＋OS－CPU＋CHANNEL－I／O
（レスポンス・タイム）

CPU　：純粋にそのジョブが演算と制御のために中央演算処理装置を占有した時間でＯＳが働いている時間は含まれない。

OS－CPU　：そのジョブを処理するために必要とするＯＳの機能を使用した時間。

CHANNEL I／O　：そのジョブを処理するために必要な、チャネルI／O時間。

次にネットワークを利用したときコア・タイムに影響を与える要素を列挙すると、

① サブネット内のメッセージ制御。

② ホスト間のメッセージ制御。

これらのことを考慮してレスポンス・タイムに関してＣＪＰを利用した方が有効な場合とは、一般に次の式が満足される場合である。

レスポンス・タイムの評価式

$$\sum_{m} (T_m + T'_m + I_m) + I' < T_o + T'_o + I_o$$

T ： ユーザ・プログラムのＣＰＵ時間

T′： ＯＳ－ＣＰＵ時間（ＮＣＰのＣＰＵ時間も含む）

I ： チャネルＩ／Ｏ時間

I′： ホストが動いていないときのサブネット使用時間

m ： 使用ホストＩＤ

o ： メイン・ホストＩＤ

B．実験結果の推測

ＣＰＵ時間、コスト、レスポンス・タイムの各効率評価式に、今回の実験の条件をあてはめ結果の推測をしてみる。

この実験の場合、メイント・ホストが図形処理をするＨＩＴＡＣ、サブ・ホストが演算処理をするＦＡＣＯＭである。つまり使用ホスト数は２（m ＝２）、モジュール数は図形処理、演算処理モジュールの２（j ＝２）、ＨＩＴＡＣのグラフィック・システム単独でこのアプリケーション・プログラムを実行した結果が表５－３である。

表５－３　ＨＩＴＡＣのグラフィック・システムで実行した結果

ＣＰＵ時間 (sec)	ユーザ時間 (sec)	プログラム・サイズ (KW)	コスト (円)
２１	３９	１５.８	１,２５０

ＣＰＵ時間２１秒の内容

図形処理　　１秒

演算処理　　２０秒

次にＣＪＰを用いた実験では、このアプリケーション・プログラムの場合、ＦＡＣＯＭ側がメッセージ送信側、ＨＩＴＡＣ側がメッセージ受信側で、m ＝１をＦＡＣＯＭ、m ＝２をＨＩＴＡＣとすると、表５－１のユーザ・プロセスのＣＰＵ時間の平均よりＦＡＣＯＭのメッセージ送信オーバヘッド定数δ_1とＨＩＴＡＣのメッセージ受信オーバヘッド定数σ_2は、

$$\delta_1 = 0.1 \times 10^{-3}$$

$$\sigma_2 = 9.25 \times 10^{-3}$$

となる。またFACOMのメッセージ送信回数TM_1と、HITAC のメッセージ受信回数RM_2は等しい（$TM_1 = RM_2$）ので、これをメッセージ転送回数Nとすると、

$$\sum_{m}^{2} (\delta_m TM_m + \sigma_m RM_m) = \delta_1 N + \sigma_2 N = (\delta_1 + \sigma_2) N$$

$$= 9.35 \times 10^{-3} N \qquad (4)$$

つまり(4)式は、FACOMからHITACにN回メッセージ転送したときの、CPU時間とメッセージ転送回数の関係式である。

(1) CPU時間に関する推測

CPU時間の評価式(2)に、今回の実験の条件をあてはめて結果の推測をしてみる。

評価式(2)は、条件をあてはめると次のようになる。

$$\beta \cdot N < T_o (1-\gamma) \frac{\alpha}{1+\alpha} \qquad (5)$$

このとき

T_o ：HITACのグラフィック・システム単独でアプリケーション・プログラムを実行したときのCPU時間

γ ：FACOMとHITACの処理速度の比

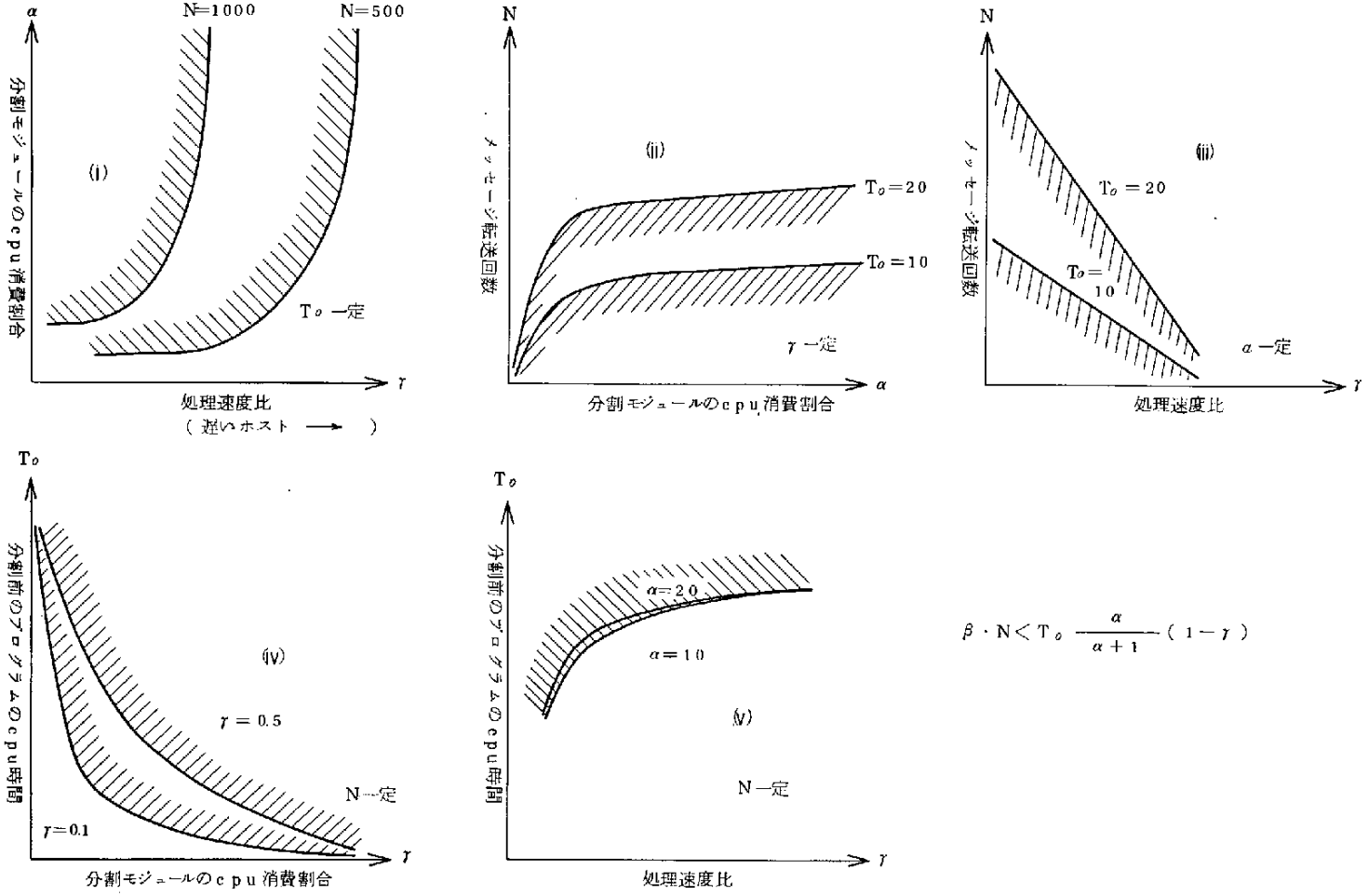
（γ＝FACOM処理速度／HITAC処理速度）

α ：分割モジュール（演算処理モジュール）のT_oに対するCPU消費の割合

N ：メッセージ転送回数

β ：メッセージ転送オーバヘッド定数（$\beta = 9.35 \times 10^{-3}$）

(5)式でCJPを利用して効果のあがる範囲を斜線で表したのが図5－9である。



$$\beta \cdot N < T_0 \frac{\alpha}{\alpha+1}(1-\gamma)$$

図５－９　ＣＪＰを利用してＣＰＵ時間に関して効果のあがる場合

図５－９のグラフ(i)～(v)から次のことがわかる。(i)、(ii)、(iii) のグラフから分割したモジュール間のデータのやりとり（すなわち、ホストーホスト間のメッセージ転送）の回数は少ない方が、処理速度比、分割モジュールのＣＰＵ消費の割合の制限がゆるくなる。また転送回数によって分割モジュールのＣＰＵ消費の割合の効果のあがらない下限が定まる。つまり、多量のデータを扱う場合は、一回の転送でなるべく多くのデータを転送できるように、データのコンパクトが必要となる。

(ii)、(iii)、(iv)、(v) のグラフから、分割前のアプリケーション・プログラムのＣＰＵ時間は長い方が分割したとき有利である。

(i)、(ii)、(iv) のグラフから、分割するモジュールのアプリケーション・プログラム全体に対するＣＰＵ消費の割合は大きい方が有利である。

(iii)、(iv) からサブホストの処理速度が速い方が有利である。

(v) から転送回数が一定ならば、効率は分割モジュールの、アプリケーション・プログラム全体に対するＣＰＵ消費の割合にはほとんど依存しない。

(2) コストに関する推測

このレポートでのコストの算出は、当協会における料金体系を使用した。

当協会におけるコンピュータ使用料金は、

ＣＰＵ時間 ： n秒

主記憶装置占有量 ： ℓ KW

のとき

コスト＝n（CPU 料金＋主記憶定数×主記憶料金）＋（I／O 料金）

によって算出されるが、今回の実験においては、Ｉ／Ｏ 料金は、ほぼ０とみなされるので無視する。

表５－４　ＪＩＰＤＥＣのコンピュータ使用料金

機　種	ＣＰＵ料金〔円〕	主記憶料金〔円〕
ＦＡＣＯＭ２３０／７５	１６０	４０
ＨＩＴＡＣ８４５０	５６	１４

主記憶装置占有量	主記憶定数
１６ＫＷ（　６４ＫＢ）未満	0.25
１６ＫＷ以上　３２ＫＷ（１２８ＫＢ）未満	0.50
３２ＫＷ以上　６４ＫＷ（２５６ＫＢ）未満	1.00
６４ＫＷ以上　９６ＫＷ（３８４ＫＢ）未満	1.50
９６ＫＷ以上１２８ＫＷ（５１２ＫＢ）未満	2.00
１２８ＫＷ以上１６０ＫＷ（６４０ＫＢ）未満	2.50
１６０ＫＷ以上	3.00

(4)式よりＮ回メッセージ転送したときのコストを求めると、ＦＡＣＯＭ　ＨＩＴＡＣとも主記憶定数を0.25として、

$$\text{メッセージ転送コスト} = 0.57 \times N \qquad (6)$$

以上のことを前提として、今回の実験用にコストの評価式(3)を変換すると次のようになる。

$$\sum_{m} (C_m + \sigma_m \cdot E_m) T_m < (C_o + \delta_o \cdot E_o) T_o \qquad (7)$$

m ：使用ホストＩＤ

o ：メイン・ホストＩＤ

Ｃ：ＣＰＵ使用料金

δ ：主記憶定数

Ｅ：主記憶使用料金

Ｔ：ＣＰＵ時間

ホストmのコスト関数をA_mとすると、

$$A_m = C_m + \delta_m \cdot E_m$$

(7)式をＣＰＵ時間の評価式(2)を求めたと同様に展開すると、

$$\sum_m A_m \; T_m < A_o \; T_o$$

よって、

$$\sum_m \left\{ \delta_m TM_m + \sigma_m RM_m + \sum_i \left(\gamma_m \frac{\alpha_i}{1+\alpha_i} \; T_o \right) \right\} A_m$$

$$< \sum_i \frac{\alpha_i}{1+\alpha_i} \; T_o A_o \qquad (8)$$

(6)式より

$$\sum_m \left(\delta_m TM_m + \sigma_m RM_m \right) A_m = 0.57 N$$

Ｎ：メッセージ転送回数

$i = 1$ を演算処理モジュール、$i = 2$ を図形処理モジュールとして(8)式を展開すると、

$$\underbrace{0.57 N}_{\text{F→Hのメッセージ転送にかかるコスト}} + \underbrace{\gamma \frac{\alpha_1}{1+\alpha_1} \; T_o A_f}_{\text{Fの演算処理にかかるコスト}} + \underbrace{\frac{\alpha_2}{1+\alpha_2} \; T_o A_h}_{\text{Hの図形処理にかかるホスト}}$$

$$< \underbrace{\frac{\alpha_1}{1+\alpha_1} \; T_o A_o}_{\text{Hの演算処理にかかるコスト}} + \underbrace{\frac{\alpha_2}{1+\alpha_2} \; T_o A_o}_{\text{Hの図形処理にかかるコスト}} \qquad (9)$$

$\alpha_2 = 1/\alpha_1$ を代入して(9)式を整理すると、

$$0.57 N < T_o \; \frac{\alpha (A_o - \gamma A_f) + (A_o - A_h)}{\alpha + 1} \qquad (10)$$

$A_o < A_h < A_f$ としてコスト関数を定数とすると、(10)式は次のようになる。

$$N < \frac{\alpha}{\alpha+1} T_o (1 - K_1 \gamma) K_2 - \frac{T_o K_3}{\alpha+1} \quad (11)$$

T_o ： メイン・ホスト単独でアプリケーション・プログラムを実行したときのＣＰＵ時間

γ ： ＦＡＣＯＭとＨＩＴＡＣの処理速度比
（γ＝ＦＡＣＯＭ処理速度／ＨＩＴＡＣ処理速度）

α ： 分割モジュール（演算処理モジュール）のT_oに対するCPU消費割合

N ： メッセージ転送回数

K_i ： 正定数
$(i=1,2,3)$

(11)式でＣＪＰを利用してコストに関して効果のあがる範囲を斜線で表したのが図５－１０である。

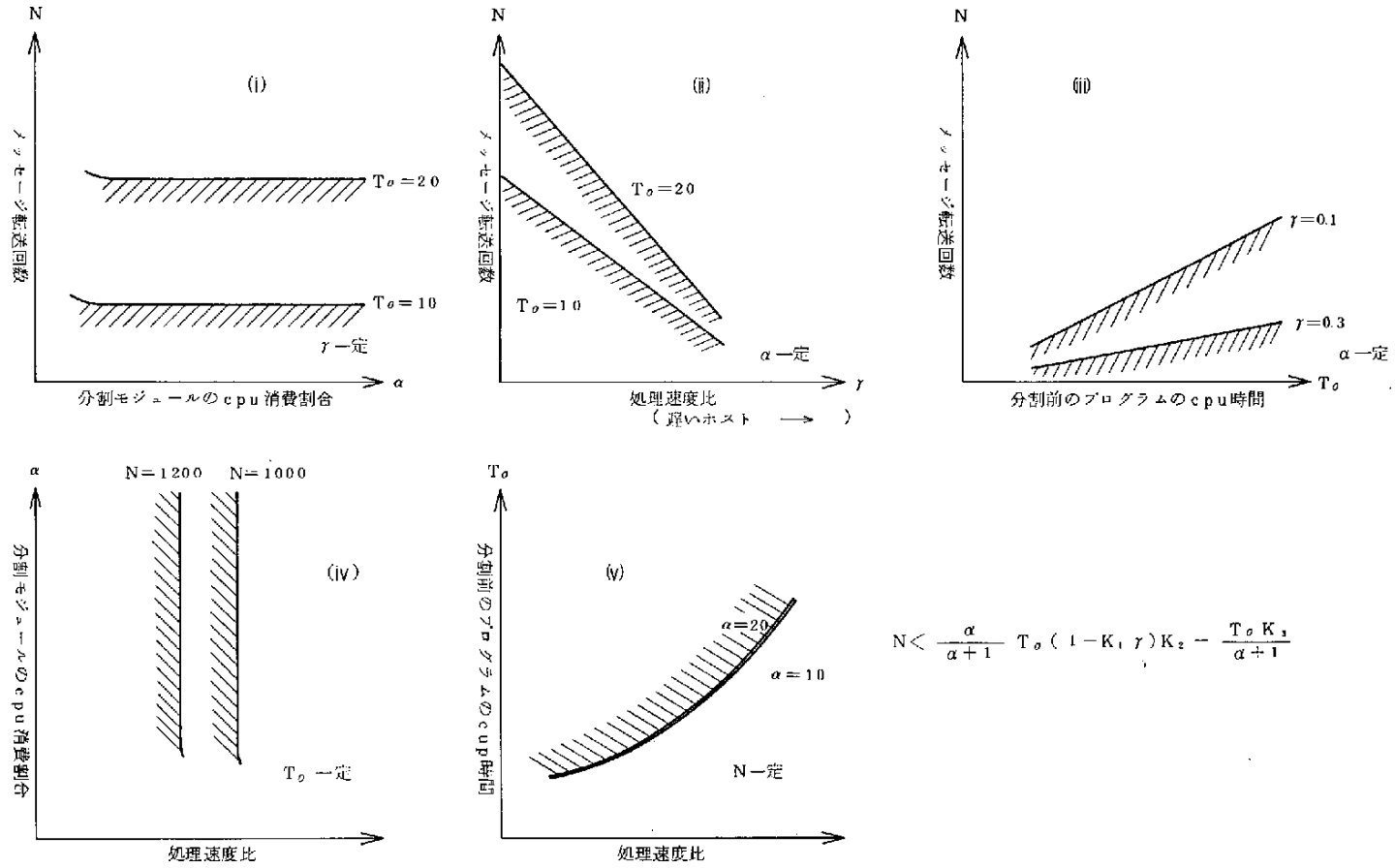
図５－１０のグラフ(i)～(v)から次のことがわかる。

(i)のグラフから、処理速度比γが一定ならば、コスト効率が上がるメッセージ転送回数Ｎの範囲は、分割前のアプリケーション・プログラム全体のＣＰＵ時間T_oに依存し、分割モジュールのＣＰＵ消費割合αには依存しない。またαに下限がある。

(ii)、(iii)のグラフからサブホストはより処理速度の速いホストの方が許されるメッセージ転送回数Ｎの条件は緩くなる。処理速度比γに上限、分割前のアプリケーション・プログラムのＣＰＵ時間T_oに下限がある。

(iv)のグラフから、分割前のアプリケーション・プログラムのCPU時間T_oが一定ならば、コスト効率はメッセージ転送回数Ｎに依存し、処理速度比γには依存しない。また分割モジュールのＣＰＵ消費割合αにはＮによって下限がある。

(v)のグラフから、メッセージ転送回数Ｎが一定ならば、コストの効



$$N < \frac{\alpha}{\alpha+1} T_o (1-K_1\gamma)K_2 - \frac{T_o K_3}{\alpha+1}$$

図５－１０　ＣＪＰを利用してコストに関して効果のあがる場合

率は分割モジュールのＣＰＵ消費割合αにはほとんど依存しない。また分割前のアプリケーション・プログラムのＣＰＵ時間 T_0 により処理速度比γに下限がある。

(3) レスポンス・タイムに関する推測

ＪＩＰＮＥＴを使用して、ＣＪＰを行うときにレスポンス・タイム（コア・タイム）に影響を与えると思われるのは次の６つの要素である。

① ＯＳのＣＰＵ時間

② ＮＣＰのＣＰＵ時間

③ ユーザ・プログラムのＣＰＵ時間

④ サブネット内のメッセージ転送時間

⑤ メッセージ転送回数

⑥ アプリケーション・プログラムが使用するチャネルＩ／Ｏ時間

このうちチャネルＩ／Ｏ時間は今回の実験では無視できるので他の５つの要素の検討をする。この５つの要素は互いに並行して処理されるときがあるので各要素がどれだけレスポンス・タイムに影響を与えるかという解析は、モニターの集計するジョブ・アカンティング・リストからは不可能であるので、このレポートではどの要素がレスポンスタイムに影響を与えるかを推測する。

表５－５はＪＩＰＮＥＴでＦＡＣＯＭからＨＩＴＡＣへメッセージ転送をしたときの結果である。これからわかるように、ＮＣＰのＣＰＵ時間、ユーザ・プロセス（メッセージ転送を行うプロセス）のＣＰＵ時間は転送回数に比例して増加するが、ホップ数が増加しても一定である。ラウンド・トリップ時間（サブネット内のメッセージ転送時間と考えられる）は図５－１１から分かる様にホップ数が増加すると増加し、また１回のメッセージ転送バイト数に比例して増加する。

表５－５　F→Hのメッセージ転送に要するラウンド・トリップ時間

転送バイト数（byte）	ホップ数	NCPのCPU時間（sec）	User ProcessのCPU時間（sec）	ラウンド・トリップ時間（sec）	Between Message時間（sec）
140×1000回	0 － hop	22.443	0.1	16.379	11.358
	fullconnect	22.601	0.1	67.341	11.504
576×1000回	0 － hop	22.625	0.1	24.149	11.489
	fullconnect	22.588	0.1	132.671	11.503
1152×1000回	0 － hop	22.908	0.1	34.669	11.689
	fullconnect	22.908	0.1	201.671	11.842

Between Message時間：NCPがメッセージ転送完了情報を受け取ってから次のメッセージを転送するまでの時間

ラウンド・トリップ時間　：サブネットがメッセージを受け取って、それを目的地に転送しメッセージ転送の完了を知らされるまでの時間

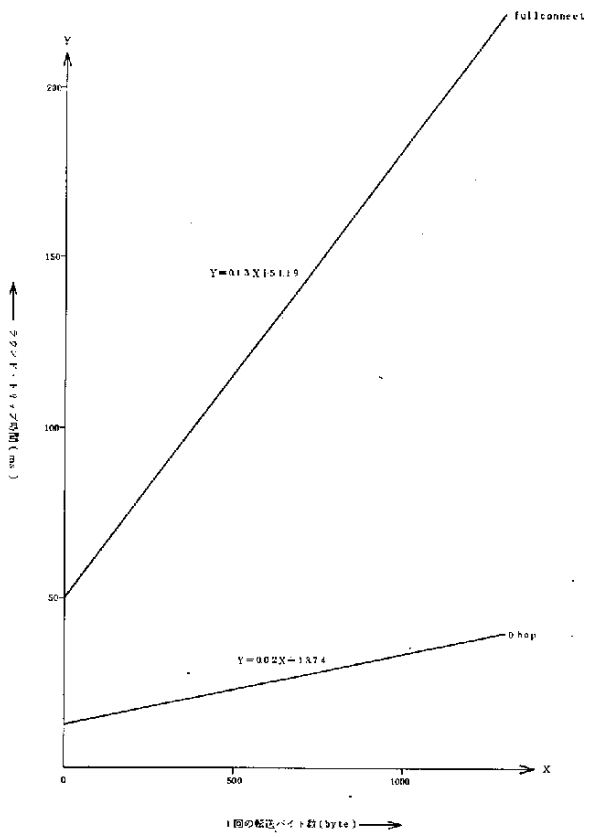


図５－11　１回の転送バイト数 vs. ラウンド・トリップ時間

以上のことより、レスポンス・タイムが増加する場合とは総転送バイト数が一定ならば、

① メッセージ転送回数が増加

② ホップ数が増加する

ときである。

5.2.2 CPU時間の解析

表5－6はFACOM、HITACでCJPを実験したCPU時間の結果である。

実験の結果からメッセージ転送回数NとCPU時間Tの近似式を求めると次のようになる。

0－hopのとき	$T = 8.83 \times 10^{-3} \times N + 3.02$	(12)
fullconnectのとき	$T = 9.01 \times 10^{-3} \times N + 3.09$	(13)
理論式	$T = 9.35 \times 10^{-3} \times N$	(4)′

(4)′式の理論式とはFACOMからHITACにN回メッセージ転送するときに要するCPU時間を表わしている。(12)、(13)、(4)′式そしてHITACのグラフィック・システムでアプリケーション・プログラムを実行したときのCPU時間、

$$T = 21 \qquad (14)$$

をグラフで表わしたのが図5－12.である。

図5－12の(12)、(13)式がほぼ一致することから、ホップ数の変化はCPU時間に影響を与えないことがわかる。また(12)、(13)式と(4)′式がほぼ平行なことからCPU時間の増減はメッセージの転送回数に依存することが分かる。次に最少の転送回数（24回）のCPU時間（3.1）秒とHITACのグラフィック・システム単独でこのアプリケーション・プログラムを実行し

表５－６　ＣＰＵ時間の実験結果

メッセージ転送回数(回)	ホップ数	HITAC CPU時間 (sec)	FACOM CPU時間 (sec)	H+FのCPU時間 (sec)
1361	0 hop	12	2.3	14.3
	fullconnect	13	2.3	15.3
681	0 hop	6	2.2	8.2
	fullconnect	7	2.2	9.2
454	0 hop	5	2.2	7.2
	fullconnect	5	2.2	7.2
341	0 hop	4	2.2	6.2
	fullconnect	4	2.2	6.2
195	0 hop	3	2.1	5.1
	fullconnect	3	2.1	5.1
124	0 hop	2	2.1	4.1
	fullconnect	2	2.1	4.1
24	0 hop	1	2.1	3.1
	fullconnect	1	2.1	3.1

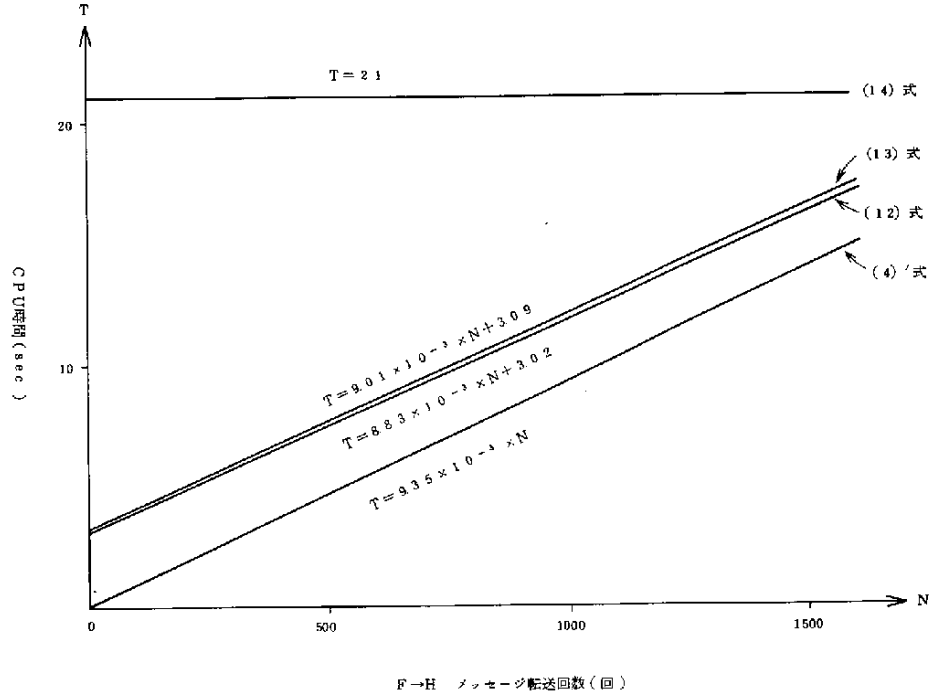


図５－１２　実験結果　メッセージ転送回数vs．ＣＰＵ時間

たときのＣＰＵ時間（２１秒）を比べてみると、ＣＪＰを利用してＣＰＵ時間は約８５％減少した。

ここでＣＰＵ時間の評価式(2)にこの実験の条件をあてはめると、（表５－３参照）

$\beta = 9.35 \times 10^{-3}$ 、$T_0 = 21$、$\gamma = 0.11$、$\alpha = 20$ だから

$$N < 1904$$

実験の結果＜１２＞式と＜１４＞式の交点を求めると、

$$N = 2037$$

となり、これは誤差を考えるとほぼ実験の結果が評価式と一致する。よってＣＰＵ時間の評価式(2)はある程度一般的な評価式といえる。

以上のことより、ＣＪＰの実験をしてＣＰＵ時間に関して分かったことを整理する。

① メッセージ転送回数は少ない方が効率が良い。

② 分割するアプリケーション・プログラムのＣＰＵ時間は長い方が分割したとき有利である。

③ 他のホストへ分割するモジュールのアプリケーション・プログラム全体に対するＣＰＵ消費の割合は大きい方が分割したとき有利である。

④ モジュールを分割するホストの処理速度は速い方が有利である。

⑤ ホップ数はＣＰＵ時間に影響を与えない。

5.2.3 コストの解析

表５－７は、コストに関する実験結果である。これからメッセージ転送回数(N)とコスト(C)の近似式を求めると次のようになる。

$$C = 0.53N + 447.76 \qquad (6)$$

またＦＡＣＯＭ→ＨＩＴＡＣの、メッセージ転送回数に対するコストは次の式によって表わされる。

$$C = 0.57N \qquad (6)'$$

表５－７　コストの実験結果

転送回数 (回)	F－CPU 時間 (sec)	F－コスト (円)	H－CPU 時間 (sec)	H－コスト (円)	CJP－コスト (円)
1361	2.3	414	12	756	1,170
681	2.2	396	6	378	774
454	2.2	396	5	315	711
341	2.2	396	4	252	648
195	2.1	378	3	189	567
124	2.1	378	2	126	504
24	2.1	378	1	63	441

コストは次のように算出した。

コスト＝（CPU料金＋主記憶定数×主記憶料金）×CPU時間

Hのプログラム・サイズ＝16.16KW、Fのプログラム・サイズ＝29KW

表５－４より、

F－コスト＝（160＋0.5×40）×　（F－CPU時間）

H－コスト＝（56＋0.5×14）　×　（H－CPU時間）

HITACのグラフィック・システムでアプリケーション・プログラムを実行したときのコストは1,250円なので、

$$C = 1250 \qquad (16)$$

この３つの式をグラフで表わしたのが図５－１３である。

図５－１３で(15)式と(6)'式はほぼ平行である。このことからCJPを利用したときのコストはメッセージ転送回数に依存することがわかる。最少のメッセージ転送回数（２４回）のときのコスト（４４１円）とHITACのグラフィック・システム単独でこのアプリケーション・プログラムを実行したときのコスト（1,250円）を比べると、CJPを利用してコストが約６５％減少した。

ここでコストの評価式の展開形(10)式にこの実験の条件をあてはめてみる。(10)式に次の条件、

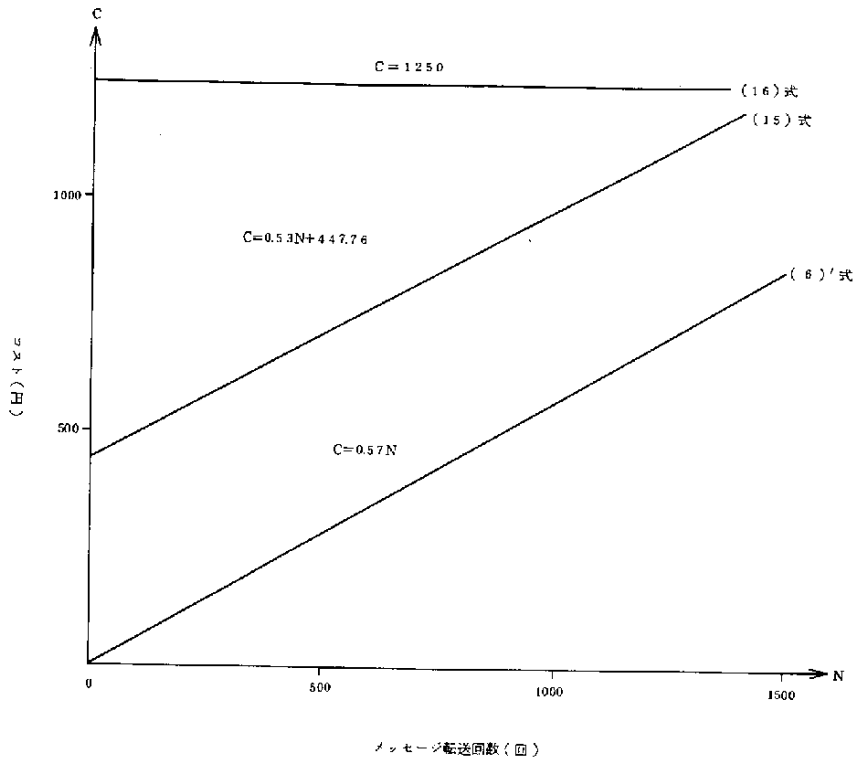


図５－１３　実験結果　メッセージ転送回数 vs. コスト

$A_f = 180$、$A_h = 63$、$A_o = 59.5$、$T_o = 21$、$\gamma = 0.11$、$\alpha = 20$

を代入すると、

$$N < 1387$$

実験の結果(15)式と(16)式の交点を求めると、

$$N = 1514$$

となり、これは計算誤差、測定誤差を考えると実験の結果が評価式と一致していると考えられる。

以上のことにより、ＣＪＰの実験をして分かったコストにとってプラスになる要素を整理すると、

① メイン・ホスト単独でのアプリケーション・プログラム全体のＣＰＵ時間が長いこと。

② メイン・ホストとサブ・ホストの処理速度の差が大きく（サブ・ホストの処理速度＞メイン・ホストの処理速度）かつコストの差が小さいこと。

③ サブ・ホストへ分割されるモジュールのアプリケーション・プログラム全体に対するＣＰＵ消費の割合が大きいこと。

④ ホスト－ホスト間のメッセージ転送回数が少ないこと。

などが上げられる。

5.2.4 レスポンス・タイムの解析

表５－８はFACOM、ＨＩＴＡＣでＣＪＰを実験したときのコア・タイム（レスポンス・タイム）の結果である。表５－８の説明をすると、FACOM、ＨＩＴＡＣは同期をとって開始、終了するのでＣＪＰ全体のコア・タイムと

表５－８　コアタイムの実験結果

転送回数(回)	１回の平均転送バイト数(byte)	ホップ数	H−CPU時間 (sec)	F−CPU時間 (sec)	F−コアタイム (sec)	ラウンド・トリップ時間 (sec)
1361	9	0	12	2.3	91.756	18.95
		full	13	2.3	163.816	71.26
681	17.987	0	6	2.2	53.162	9.6
		full	7	2.2	90.742	36.45
454	26.98	0	5	2.2	39.8	6.48
		full	5	2.2	66.442	24.83
341	35.92	0	4	2.2	35.212	4.93
		full	4	2.2	54.274	19.01
195	62.82	0	3	2.1	25.582	2.92
		full	3	2.1	38.948	11.58
124	98.78	0	2	2.1	21.2	1.95
		full	2	2.1	31.242	7.94
24	510.375	0	1	2.1	15.578	0.57
		full	1	2.1	20.132	2.82

してFACOMのコア・タイムを採用した。ラウンド・トリップ・タイムは、表５－５をもとにした。

ＣＪＰ利用時のコア・タイムは、メッセージ転送回数をＮ、コア・タイムをＥとすると、

0－hop　　$E = 0.057N + 14.5$　　(17)

fullconnect　　$E = 0.11N + 17.8$　　(18)

となり、ＨＩＴＡＣのグラフィック・システムでこのアプリケーション・プログラムを実験したときの結果は、

$E = 39$　　(19)

である。これをグラフで表わしたのが図５－１４である。この実験のコア・タイムにとっての最良の条件（メッセージ転送回数＝２４回、0－hop）

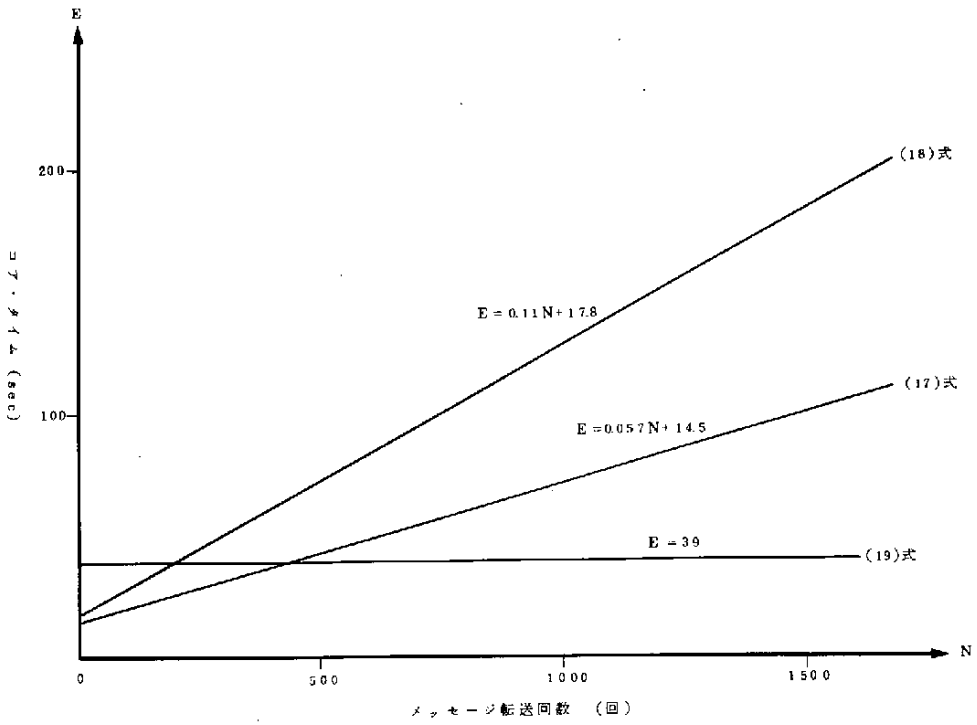


図５－１４　実験結果 コアタイム vs. メッセージ転送回数

のときのコア・タイム（15.578秒）とHITACのグラフィック・システム単独でこのアプリケーション・プログラムを実行したときのコア・タイム（39秒）を比べてみると、CJPを利用してレスポンス・タイムは約60%減少した。

次に(19)式と(17)、(18)式の交点を求めると、

0－hop　　N＝430

fullconnect　　N＝193

となり、このアプリケーション・プログラムの場合、メッセージ転送回数を0－ホップのとき430回未満、フルコネクトのとき193回未満に押さえれば、レスポンス・タイムの効率が良くなることがわかる。

以上のことより、CJPの実験をしてレスポンス・タイムに関して分かったことを整理する。

レスポンス・タイム（コア・タイム）を構成する時間、すなわちOS－CPU時間、NCP—CPU時間、ユーザ・プログラムのCPU時間、サブネット内メッセージ転送時間はどんな要素で増加するか、あるいは一定か、をまとめたのが表5－9である。

表5－9　レスポンス・タイム構成時間と各要素の関係

レスポンス・タイム構成時間	メッセージ転送回数の増加	1回のメッセージ転送量の増加	ホップ数の増加
OS－CPU時間	増加	一定	一定
NCP—CPU時間	増加	一定	一定
サブネット内メッセージ転送時間	増加	増加	増加
ユーザ・プログラムCPU時間	増加	一定	一定

表５－９から分かるようにレスポンス・タイムの効率を良くするには、

① ホップ数を少なく

② メッセージ転送回数を少なく

③ １回のメッセージ転送量を少なく

しなくてはならない。

5.2.5 結　論

5.2.2、5.2.3、5.2.4でCPU時間、コスト、レスポンス・タイムの実験結果の解析をしてきたがこの実験結果からCJP向きのアプリケーション・プログラムとはどういうものか、またCJP向きの環境とはどういうものか整理してみる。

① CJPを利用するときの大前提として、サブ・ホストはメイン・ホストより、処理速度が早いこと。

② CPU時間、コスト、レスポンス・タイムすべてがメッセージ転送回数に比例して増加するので、分割モジュール間でのデータのやりとり（つまり、メッセージ転送回数）が少ないアプリケーション・プログラム。

③ アプリケーション・プログラムのCPU時間が長いこと。

④ あるモジュールをメイン・ホストからサブ・ホストに移す場合、その分割モジュールのCPU消費の割合が、アプリケーション・プログラム全体のCPU消費に対して大きいこと。

⑤ マシンに依存する言語（アセンブラ）で組まれているプログラムはホストが限定されてしまうので、最適分担が可能なように、プログラムはすべて高級言語で組まれていること。

⑥ ホストの選択を容易にするように、使用リソース別のモジュール構造にすること。

⑦ レスポンス・タイムの効率を良くすることを求めるならば、ホップ数、

1回のメッセージ転送量を少なくしなくてはならない。

⑧ メッセージ転送量、転送回数の増加は、ＣＪＰ全体の効率を悪くするので転送データはコンパクト化されていること。

以上のような条件を持つアプリケーション・プログラム、環境がＣＪＰに向いているがこれはかなり困難な条件と言える。なぜならば、データの転送回数を減らすためのデータ・コンパクトにはかなり複雑なデータ操作が必要である。また処理の性質により，すべてのモジュールを高級言語で組むことができない場合が多いし、レスポンス・タイムの効率を良くするためにＣＪＰを利用する場合、ホップ数が増えると効率は悪くなるので、ネットワーク内のメイン・ホストの位置によって条件が異ってくる。

このようにどんなジョブでもＣＪＰを利用すれば効率が良くなるという訳ではないが、ＣＪＰ向きのアプリケーション・プログラム、環境で利用するならば、確かに単独ホストで実行するよりも、ＣＪＰを利用した方が効率は良くなる。

5.3 ＣＪＰ評価式

前章の実験の解析の結果、アプリケーション・プログラムによってはCJPの利用は確かに有効であることが分かった。そこでこの章ではモジュールをどのホストに分割するかの評価式（最適配置式）を考える。ＣＪＰを利用するユーザは『安く、早い処理』を目的としてＣＪＰを利用するので、評価式を立てる準備としてコストと、ＴＡＴ（Turn around time）を定義する。

次の言葉を定義する。

Pu（Processing unit）：処理単位

Tu（Time　　unit）：時間単位

Cu（Cost　　unit）：コスト単位

5.3.1　ターン・アラウンド・タイム

ＴＡＴはユーザがネットワークにジョブを投入してから出力されるまでの時間とする。またサブネットの処理時間はホストの処理時間に含まれるとする。

A．ホストでのモジュール処理にかかわる時間単位

ℓ：ホストＩＤ。

i：モジュールＩＤ。

とし、モジュールiの処理の大きさをＰi（pu）とし、ホストℓの処理能力を$r\ell$（pu／tu）とするとホストℓでモジュールiを処理するにはＰi／$r\ell$（tu）かかる。複数のモジュールをホストℓで処理すると考えるとℓで消費される時間単位は

$$\frac{1}{r\ell}\sum_{\ell i} P\ell i \ (\mathrm{tu}) \qquad 1 \leq \ell i < i$$

と表わせる。

B．ホストでメッセージ転送処理にかかる時間単位

Ｒℓ　：ホストℓのメッセージ受信回数

Ｔℓ　：ホストℓのメッセージ送信回数

α_{ℓ} (tu)　：ホストℓのメッセージ受信処理オーバヘッド定数

β_{ℓ} (tu)　：ホストℓのメッセージ送信処理オーバヘッド定数

とすると、ホストℓでメッセージ転送にかかる時間単位は、

$$\alpha\ell \, R\ell + \beta\ell \, T\ell \quad (\mathrm{tu})$$

ホストℓでの処理に消費される時間単位は、

$$\frac{1}{r\ell}\sum_{\ell i} P\ell i + \alpha\ell R\ell + \beta\ell T\ell$$

また他のホストとの同期をとるための待ち時間をＷℓ（tu）とするとホストℓで消費される時間単位は、

$$\frac{1}{r_\ell}\sum_{\ell i} P_{\ell i} + \alpha_\ell R_\ell + \beta_\ell T_\ell + W_\ell \qquad (20)$$

各ホストに分割されたモジュールは互いに同期をとって開始、終了するとするとＴＡＴは(20)式となる。

ターン・アラウンド・タイム

$$Tcjp = \frac{1}{r_\ell}\sum_{\ell i} P_{\ell i} + \alpha_\ell R_\ell + \beta_\ell T_\ell + W_\ell \quad (tu)$$

Tcjp ： ＣＪＰを利用したときのターン・アラウンド・タイム（tu）

ℓ ： ホストＩＤ

i ： モジュールＩＤ

r_ℓ ： ホストℓの処理能力（pu／tu）

$P_{\ell i}$ ： ホストℓで処理されるモジュールの処理の大きさ（pu）

R_ℓ ： ホストℓのメッセージ受信回数

T_ℓ ： ホストℓのメッセージ送信回数

α_ℓ ： ホストℓのメッセージ受信処理オーバヘッド定数（tu）

β_ℓ ： ホストℓのメッセージ送信処理オーバヘッド定数（tu）

W_ℓ ： モジュール間の同期をとるための待ち時間（tu）

5.3.2 コスト

A．サブネット内処理時間単位

ＣＪＰを利用したときのネットワーク全体でのコストを考えるためにサブネット内での処理時間単位を考える。

メッセージ転送が行われるときホストＡと、ホストＢの間の転送継路は常に同じルートを通るとする。またネットワークは平衡状態（equilibrium）で、回線によるパケット伝送遅延時間、チャネル伝播遅れ時間はないとし、ＩＭＰは互いに独立（ＩＭＰ相互の依存関係なし）で各ＩＭＰの処理能力は同一とする。

N ： 分割モジュール間の総転送パケット数

M ： サブネット内のIMP数

σ ： IMPのメッセージ処理能力（packet／tu）

nk ： ルート k を通るパケット数　$1 \leq nk < N$

mk ： ルート k のIMP数　　$1 \leq mk < M$

以上のことを前提とすると、ルート k における処理時間単位は、

$$\sum_{mk} \frac{nk}{\sigma} \quad (\text{tu})$$

アプリケーション・プログラムによって使用されるルート数を K とすると、サブネット全体での処理時間単位は、

$$\sum_{k=1}^{K} \left(\sum_{mk} \frac{nk}{\sigma} \right) \quad (\text{tu})$$

と表わされる。

B．コスト

ホストのリソースを使用したときのコスト係数を、

CHℓ：ホスト ℓ のリソース使用コスト係数　（cu／tu）

とし、サブネットを使用したときのコスト係数を

CS ：サブネット使用コスト係数（cu／tu）

とすると、ホスト ℓ にかかるコスト単位は、

$$\text{CH}\ell \left(\frac{1}{r\ell} \sum_{\ell i} P\ell i + \alpha\ell R\ell + \beta\ell T\ell \right) \quad (\text{cu})$$

アプリケーション・プログラム全体で使用するホスト数を L とすると、使用ホスト全体でのコスト単位は

$$\sum_{\ell=1}^{L} \text{CH}\ell \left(\frac{1}{r\ell} \sum_{\ell i} P\ell i + \alpha\ell R\ell + \beta\ell T\ell \right) \quad (\text{cu})$$

サブネットの使用コスト単位は、

$$CS \sum_{k=1}^{K} \left(\sum_{mk} \frac{nk}{\delta} \right) \qquad (cu)$$

よってアプリケーション・プログラム全体にかかるコスト単位をCcjpとして、

CJP利用コスト関数

$$Ccjp = \sum_{\ell=1}^{L} CH\ell \left(\frac{1}{r\ell} \sum_{\ell i} P\ell i + \alpha\ell R\ell + \beta\ell T\ell \right) + CS \sum_{k=1}^{K} \left(\sum_{mk} \frac{nk}{\sigma} \right) \qquad (cu)$$

Ccjp ： CJPを利用したときのコスト （cu）

CHℓ ： ホストℓのリソース使用コント係数 （cu／tu）

CS ： サブネット使用コスト係数 （cu／tu）

L ： 使用ホスト数

K ： サブネット内で使用するルート数

N ： 分割モジュール間の総転送パケット数

M ： サブネット内のIMP数

σ ： IMPのメッセージ処理能力（packet／tu）

mk ： ルートkのIMP数 （$1 \leq mk < M$）

nk ： ルートkを通るパケット数（$1 \leq nk < N$）

ℓ ： ホストID

i ： モジュールID

$r\ell$ ： ホストℓの処理能力（pu／tu）

Pℓi ： ホストℓで処理されるモジュールの処理の大きさ（pu）

Pℓ ： ホストℓのメッセージ受信回数

Tℓ ： ホストℓのメッセージ送信回数

$\alpha\ell$ ： ホストℓのメッセージ受信処理オーバヘッド定数（tu）

$\beta\ell$ ： ホストℓのメッセージ送信処理オーバヘッド定数（tu）

5.3.3　アプリケーション・プログラムのリソース要求

アプリケーション・プログラムは、各プログラムによってホストに対するリソース要求が異なるし、ホストのサービス・リソースもそれぞれのホストによって種類、そのサービスの度合も異なるので、全てのアプリケーション・プログラムに適した最適配置式を論じることには無理がある。

そこでまず第1段階としてモデル・ホストを１０機種、モデル・アプリケーション・システムを１０種選び、ホストについてはどんなサービスをし、そのサービスの度合はどのくらいか、アプリケーション・システムについては、ホストにどんなリソースを要求し、その要求はどのくらいの度合かを調査した。表５－１０が各ホストのサービス・リソースの一覧表である。また表５－１１はアプリケーション・システムのホストに対するリソース要求一覧表である。各表の見方を説明する。

表５－１０ ホストのサービス・リソース一覧表の見方

性能：各数字は処理速度、容量の大きさの順を表わしている。

OS機能、サービス・ソフトウェア：各マークは次の事を表わす

◎……強力にサポート

○……十分にサポート

△……サポートしている

言語：各文字は次の事を表わす

A……ASSEMBLA

C……COBOL

F……FORTRAN

P……PL／1

R……RPG

AL ……ALGOL

表５－１１ 各アプリケーション・システムのリソース要求一覧表の見方

印は次のことを表わす

◎……大きな比重を占めるリソース要求

○……必要なリソース要求

表5－10　ホストのサービス・リソース

項目	性能					OS機能		サービス・ソフトウェア							備考	
サービス機能 / ホスト名	CPUの能力	コア容量	二次記憶容量	アクセス・タイム	メッセージ転送速度	TSS	マン・マシン	DBM	IR	OR	数値解析	予測	統計	言語	特殊デバイス	特徴
A	1	1	1	1	1	○	○	◎	◎	◎	◎	◎	◎	A, C, F, P, R, AL	汎用、オンライン、TSS端末	超大型システム
B	7	2	6	1	2	○	○	◎	◎	△	△	○	○	A, C, F, P, AL	OCR, OMR, POSターミナル 汎用, CD, GD, テレプリンタ	大型システム
C	2	3	2	1	8	◎		◎	◎	◎	◎			A, C, F, P, R	汎用、オンライン、TSS端末	高度なオンライン・DB大型システム
D	8	4	4	2	7	◎	○	△	○			△	△	A, C, F, P, R, AL	OCR, OMR, GD, CD, PLOTER, TTY	オンライン・リアルタイムに適した大型システム
E	3	4	4	3	4		◎	◎	○		○			A, C, F, R	CP, MCR, OCR, GD, XY-PLOTER	オンライン・リアルタイム・リモートバッチ可能な中型システム
F	4	5	7	5	3	◎	◎			○	○	◎	○	F, A	XY-PLOTER, PD, CD, GD, R-GD, OCR R-TTY, A-D I/O unit, PROCESS I/O	TSS、リアルタイム向ミニコン
G	6	6	5	7	6		○	△				△	△	A, C, F, P, AL	MSR, OCR, XY-PLOTER, POSターミナル インクワイアリTTY	オンライン・リアルタイム・リモートバッチ可能な小型システム
H	5	7	3	4	5	○	◎	◎	○	△		△		A, C, F, AL	OCR, OMR, リアルタイム端末　漢字端末	オンライン・リアルタイム、TSS、マンマシン指向小型システム
I	10	7	8	6	8	△	○	△		△		△	△	A, C	漢字端末、XY-PLOTER, GD, CD, OCR, ビリング・ターミナル	事務処理向き、オンライン、マシン向小型システム
J	9	8	9	8	2	◎					△	△	△	A, C, F	手書きOCR, オンライン端末	オンライン・オフラインのサテライト向き

表5－11　各アプリケーション・システムのリソース要求

項目	性能			OS機能				サービス・ソフトウェア								備考	
サービス機能要求 / システム名	高速CPU	大量コア	大量二次記憶	TSS	リアル・タイム	オン・ライン	マン・マシン	DBM	IR	OR	数値解析	予測	統計	言語	特殊デバイス	特徴
経営管理	◎	◎	◎	◎			○	◎	◎	○		◎	◎	ユーザ・オリエンティッドな非手続向、POL	GD	TSS可能な大型コンピュータ必要
物流管理	◎		○	◎				◎		◎		◎	○	〃	GD	会話型TSS
研究開発管理	◎		○	◎				◎			◎	○		PL／1		大規模DB
設計管理	○						◎	○			○		○	FORTRAN	GD，プロッタ	図面の記録、保存、表示
環境情報	◎	○	◎					◎	◎		◎	○		予測シミュレーション用 POL	GD，データセンサ XYプロッタ	
教育情報				◎	◎		◎	○	○					学習用POL，コマンド言語	GD，学習用端末	
防災・防犯情報	○		◎		○	○		◎	○		○	○		IR用POL	CRT，センサ	A／D変換
医療情報	◎	○	◎	◎	○		◎	◎	○		◎			A，F	GD，CD	PROCESS CONTROL
地価適正評価	◎	○	○	◎	○		○	○	○	○	○	○		専用会話型	GD	
流通情報	◎	◎	◎	○	○	○	○	◎	◎	◎				A，各種POL	各種	世界的なオンライン・ネットワーク

ＣＪＰを実行するにあたっては、アプリケーション・プログラムが要求するリソースをどのホストがもっているか調べ、次にその要求するサービスの度合を満たすホストを調べて、これらの条件を満たすホストの中でモジュールの最適配置を考えなければならない。図５－１５はホストの提供リソースとアプリケーション・システムのリソース要求の関連を表わしたもの（Associative Structure）である。

経営管理システムを例にして図５－１５を説明する。

これはシミュレーション・モデルなどによって、環境条件の予測、各種施策の効果を測定し、長期的、戦略的な経営計画をサポートするアプリケーション・システムである。データとしてマクロ・内外経済情報、業界・商品市況情報、社内各種経営統計、財務データ等を蓄積して汎用Ｄ･Ｂ･Ｍ（Data Base Management）システムにより機密保護、条件検索、メニュー検索、ファイル・メインテナンス等を行う。ＯＳのサービス機能としては会話型ＴＳＳが必要で、サービス・ソフトウェアとしては各種ＯＲ、予測、統計手法を用いる。また時系列データ・アウトプットなどの為にマン・マシンのサービスも必要である。以上が経営管理システムの概要である。このモデルを使ってＣＪＰの実行を考えてみる。

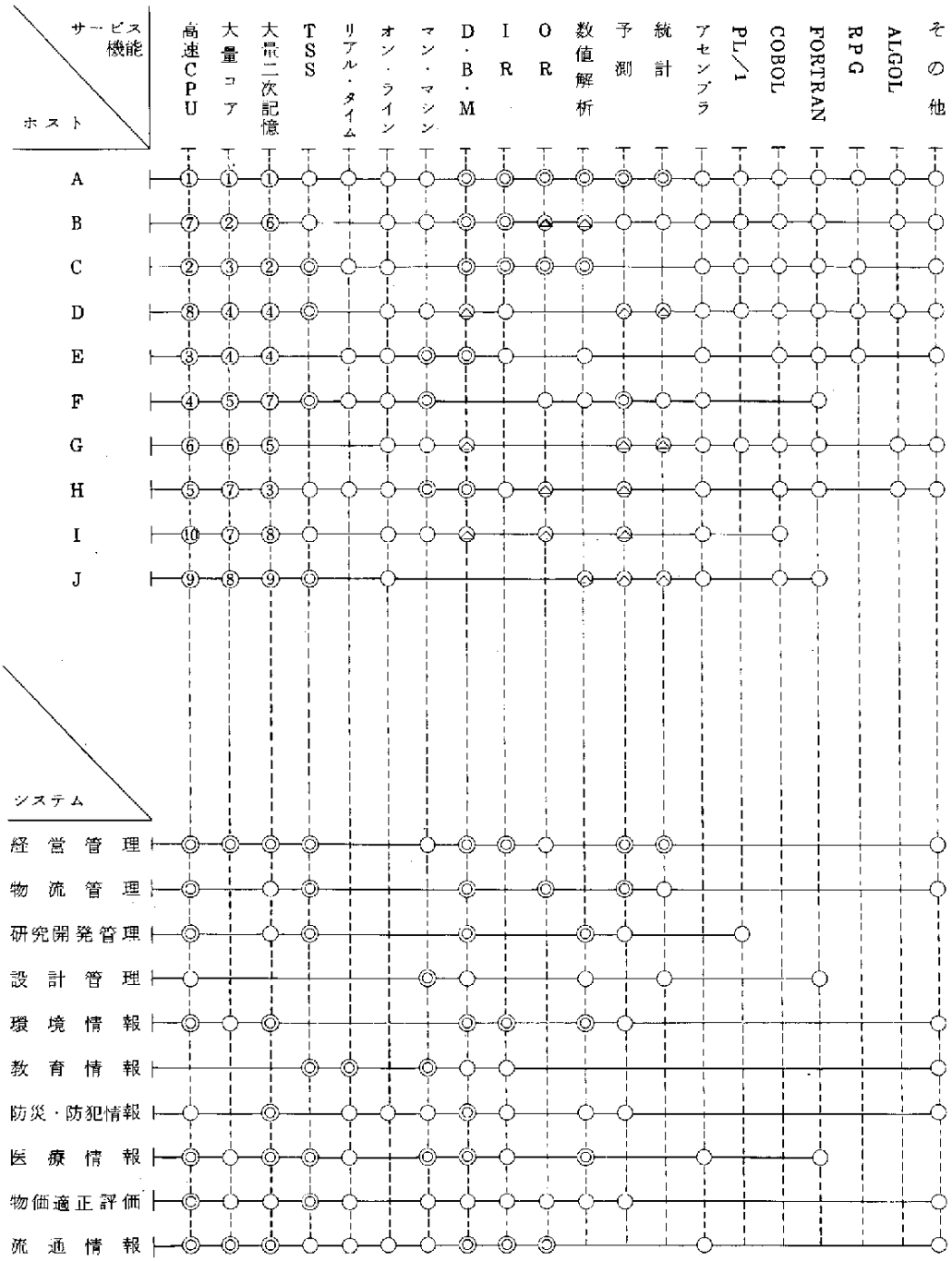


図５－１５　ホストの提供リソースとアプリケーション・システムの
リソース要求の Associative Structure

アプリケーション・プログラムをモジュール分割する場合、その分割の仕方により選べるホストが異ってくるが、ここでは次の様に分割してみた。

モジュール１：データのＩ／Ｏなど会話型ＴＳＳを用いてネットワークにアクセスするモジュール。

モジュール２：ＯＲ、予測、統計処理をするモジュール。

モジュール３：ＤＢＭ、ＩＲを処理するモジュール。

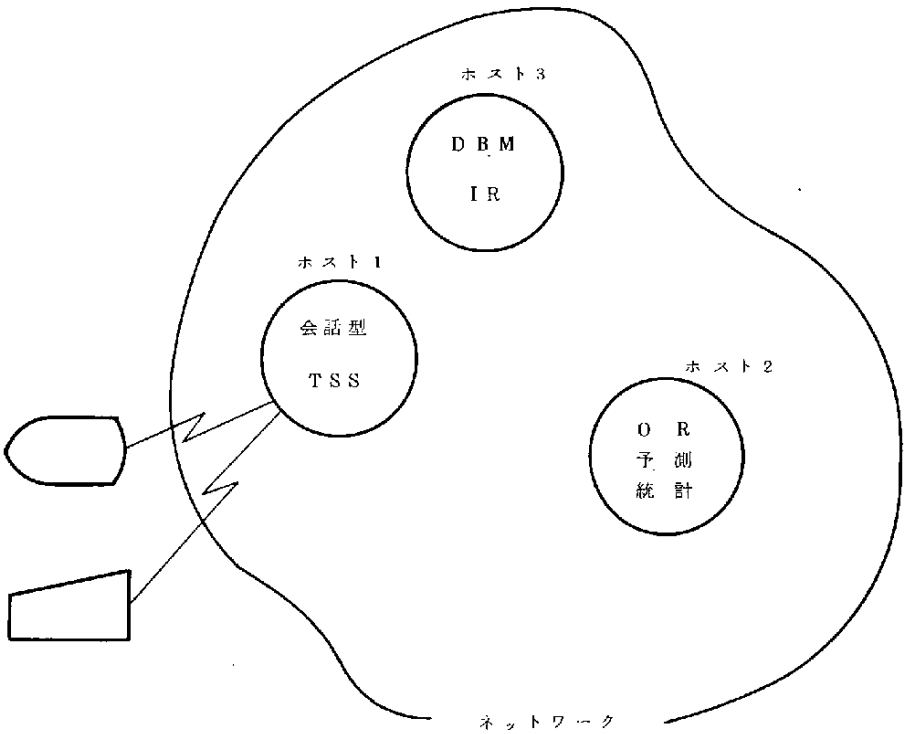


図５－１６　モジュールの分割

図５－１５からモジュール１、２、３がホストに要求するリソース・サービスとその度合からそれを満足するホストを選んでみる。モジュール１がホストに要求するリソース・サービスとしては、会話型ＴＳＳが可能なこと、またサービス言語としてユーザ・オリエンティッドな非手続向き言語、あるいはＰＯＬをサービスしていることがあげられる。この要求を満

たすホストはA、B、C、D、Hである。モジュール2がホストに要求するリソース・サービスは、OR、予測、統計のサービス・ソフトウェア、大量コア（4（524KB）以上）、高速CPU（4（平均演算時間が1.3 μs）以上）である。この要求を満たすホストはAだけである。モジュール3を分割するホストは、汎用DBMシステム、IRのサービス・ソフトウェアをもち、大量コア（4以上）、大量の2次記憶（4（232MB）以上）をもつホストでなければならない。この条件を満たすホストはA、Cである。

モジュール	要求を満たすホスト
1	A、B、C、D、H
2	A
3	A、C

分散処理ということを考えれば選べるホストは、モジュール1はB、D、H、モジュール2はA、モジュール3はCとなるから、最適配置を考える場合、実際にはモジュール1をホストB、D、Hのどこに分担させるかを考えれば良い。コスト、TATの両面からこの最適配置を考えてみる。

各モジュールは同期をとって開始・終了するとアプリケーション・システム全体としてのTATは各モジュールのステータスに依存する。シリアルなステータスの場合には、各モジュールの処理時間を最少にするホストを選べば良い。

しかし、パラレルな処理形態の場合には、あるホストの処理時間が他のホストの処理時間に含まれてしまう場合がある。

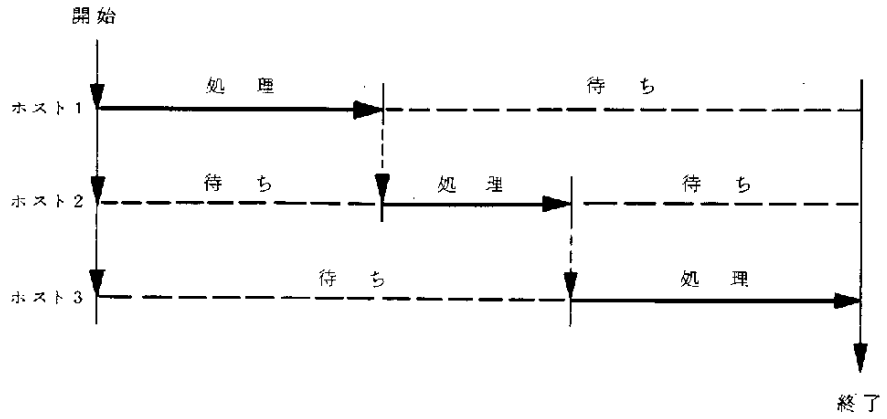


図5－17　シリアル処理

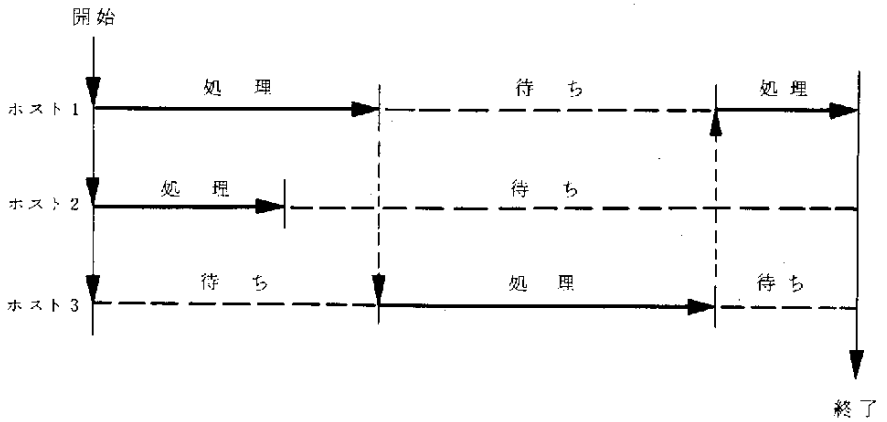


図5－18　パラレル処理

図5－18のホスト1とホスト2はパラレル処理をしていてホスト2の処理時間はホスト1の処理時間に含まれている。このホスト2を選ぶ場合、ホスト1の処理時間内でその分割モジュールを処理できるホストを選べば良いことになるので選べるホストの条件が緩くなる。例にした経営管理システムのモジュール1の場合、I／O処理が主なので他のモジュールとパラレルに処理されていると考えられる。そこで次の2つの場合が考えられ

る。

① モジュール１の処理時間がモジュール２、３の処理時間に含まれる場合。

② モジュール１の処理時間がモジュール２、３の処理時間を越える場合。

ＣＪＰのユーザはアプリケーション・プログラムを〝早く、安く〟処理するために理用するので①のときには使用コストの安いホストを、②のときには最も処理時間の短いホストを選ぶことになる。5.3.1、5.3.2より、モジュール１、２、３をホストℓ_1、ℓ_2、ℓ_3に分割するとしてこの２つの場合の評価式は次の様になる。

①の場合の評価式

$$E_1 = CH\ell_1 \left(\frac{Pm_1}{r\ell_1} + \alpha\ell_1 R\ell_1 + \beta\ell_1 T\ell_1 \right)$$

ただし $$\frac{Pm_1}{r\ell_1} + \alpha\ell_1 R\ell_1 + \beta\ell_1 T\ell_1 \leq \frac{Pm_2}{r\ell_2} + \alpha\ell_2 R\ell_2 + \beta\ell_2 T\ell_2$$

または $$\frac{Pm_1}{r\ell_1} + \alpha\ell_1 R\ell_1 + \beta\ell_1 T\ell_1 \leq \frac{Pm_3}{r\ell_3} + \alpha\ell_3 R\ell_3 + \beta\ell_3 T\ell_3$$

②の場合の評価式

$$E_2 = \frac{Pm_1}{r\ell_1} + \alpha\ell_1 R\ell_1 + \beta\ell_1 T\ell_1$$

E_1、E_2を最少にするホストを選ぶことが最適配置となる。

この様にただ一つのアプリケーション・プログラムを例にしたときでも、処理のステータスによって評価の対象が異ってくるので、すべてのアプリケーション・プログラムにあてはまる評価式を立てることは困難である。

しかし、E_1、E_2を比べるとE_1はE_2にコスト係数ＣＨが付加されただけなので、処理のステータスにかかわらず、モジュールの最適配置を与える評価式Ecjpは次の様に表わせる。

CJP評価式

Ecjp = F (ϕ TAT × ω COST)

Ecjp : CJP評価式

TAT : ターン・アラウンド・タイム関数

COST : コスト関数

ϕ : TATの重み係数

ω : COSTの重み係数

CJP評価式Ecjp はコストとターン・アラウンド・タイムの関数である。しかしターン・アラウンド・タイムの構成要素であるホストの処理能力γとコスト系数CHは正比例($\gamma\ell > \gamma\ell+1$ のとき $CH\ell > CH\ell+1$)するので、コスト、ターン・アラウンド・タイムのどちらを重視するかによって同じアプリケーション・プログラムでもモジュールを分割する最適ホストが異ってくる。コスト、ターン・アラウンド・タイムのどちらを重視するかはϕ、ωの2つの重み係数によって与えられる。

5.4 今後の課題

現在のネットワークは、ユーザにとってネットワークが複数のコンピューティング・システムの集まりとして見える〝コンピュータ・コミュニケーション・ネットワーク〟が大勢を占めているが、ネットワークの究極の目標は、ネットワーク全体を論理的な一つの大きなコンピューティング・システムとみなす事のできる〝コンピュータ・ネットワーク〟である。

なぜならば、コンピュータ・コミュニケーション・ネットワークでジョブ単位の分散処理を実行するときでも、使用可能なリソースが複数ある場合、ユーザがジョブの最適配置をするには、まず各ホストのリソース状況をすべて把握していなくてはならないし、特定システムのリソースを呼び出すのに必要なホスト特有のコマンドに習熟していなくてはならない。しかしコンピュータ・ネットワークならば一種類のネットワーク・コマンドでリソースに

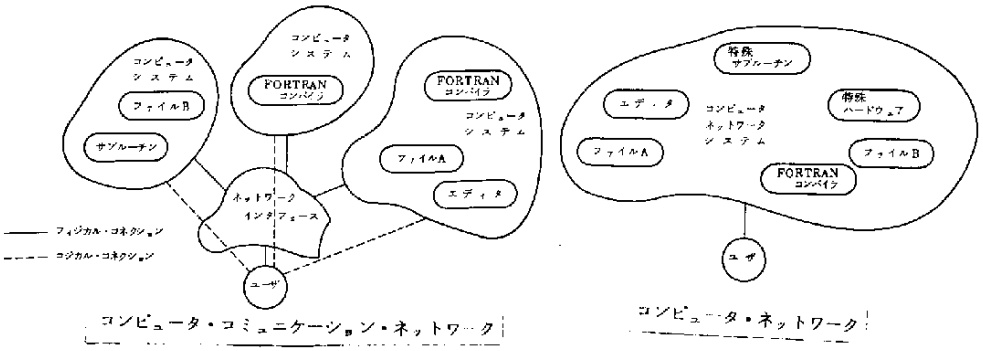


図５－１９　ユーザからみた「コンピュータ・コミュニケーション・ネットワーク」と「コンピュータ・ネットワーク」

アクセスできる。またジョブの実行にどのリソースを使用すれば最適であるか、システムが決定してくれる。つまりシステムがデシジョン・メーキング機能をもっているということである。この機能の最低条件は、

① 指定されたプロセスに対してどのリソースを割り当てるか。

② ファイルはどこに存在するか、あるいは存在するべきか。

③ そのプロセスをどのプロセッサで実行させるか。

の決定を自動的に行うことである。

この〝コンピュータ・ネットワーク〟を実現する為に、ネットワーク内で共通のフォーマットでデータのやりとりが可能な様にネットワーク・スタンダード・データ・プロトコルの早急な確立が望まれる。

将来のＣＪＰの利用形態を予想したのが図５－２０である。

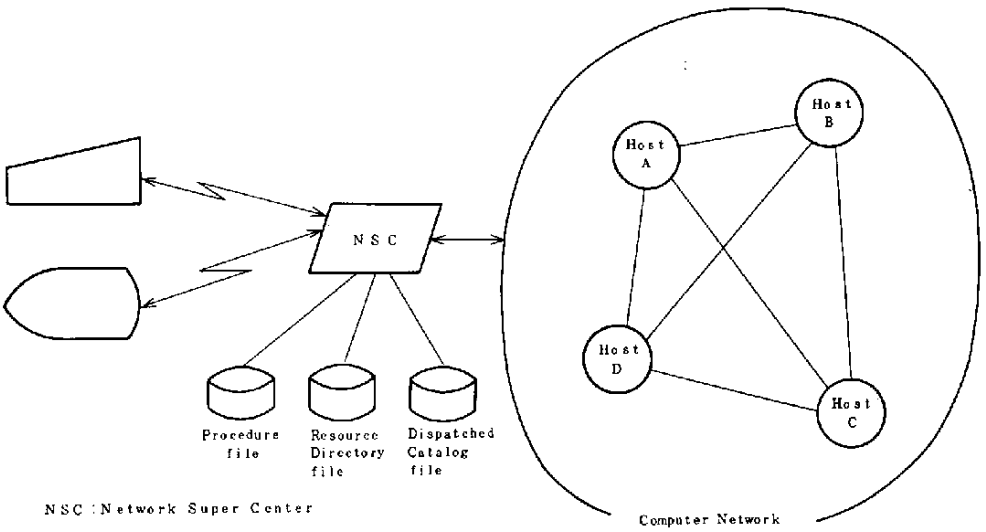


図5－20　ＣＪＰの利用形態

この利用形態の概略を述べると次の通りである。

① まずコンピュータ・ネットワークの窓口であるＮＳＣ（Network Super Center）へのアクセスはＮＳＯＳ（Network Standard Operating System）による。

② ＮＳＣは受け取ったジョブの各モジュール毎にもった実行属性（Execute Attribute）からそのモジュールがどのようなリソースを必要としているか知り、各ホストのリソース状況の入ったファイル（resource directory file）によって最適なホストを決定する。

③ またその際どのモジュールがどのホストへ分担されたかファイル（Dispatched Catalog file）に登録しておき、障害に備える。

④ プログラムは標準インタフェースを使って設計される。

○入出力はネットワーク・標準入出力プロトルコによる。

○プログラムはFORTRANの様な、マシンに依存しない高級言語で書かれていること。

以上の様に、ネットワークの利用形態として、ＣＪＰは一つの究極の理想像である。しかし、実際にＣＪＰを利用するには、まだまだむずかしい問題が残っている。それらの問題の解決にいくら労力を費しても、巨大なシステムを効率よく稼動するために、ぜひ実現したい処理形態である。

6．ネットワーク・ジョブ制御言語：NJCL

6．ネットワーク・ジョブ制御言語：NJCL

6.1 はじめに

最近、コンピュータ・ネットワークが実用段階に達してきている。コンピュータ・ネットワークの研究段階では、コンピュータに精通したエキスパートしかネットワークを使用しなかったが、実用・普及段階に入ると一般のエンド・ユーザも使用するようになってくる。

数多くの異機種コンピュータから構成されたリソース・シェアリング・コンピュータ・ネットワークの利用においてエンド・ユーザがまず直面する問題は各機種用の制御言語の多様性である。

コンピュータ・ネットワークが存在する以前は、自分が利用している計算センターのコンピュータの制御言語のみを知っていれば充分であった。しかし、コンピュータ・ネットワークが実現され、ネットワーク内の複数のコンピュータを使用するためにはそれらのすべての制御言語に精通しなければならない。さらに、いくつかの異なった制御言語を同時に使用することは非常に繁雑である。こうしたことが、初めてネットワークを使用しようとするエンド・ユーザを遠ざけてしまう結果となり、少なからずネットワーク普及の障害となっている。

それ故、ユーザがネットワーク内のすべてのリソースに対して容易にアクセスできるように、すべてのコンピュータに共通の制御言語を必要とする。即ち、ユーザは一つの共通の制御言語のみを学ぶ事によってネットワーク・システム全体がアクセス可能となる。

この制御言語としては異機種間のリソース・シェアリング・コンピュータ・ネットワーク環境下に適した機種に依存せぬハイ・レベルのネットワーク・ジョブ制御言語（ＮＪＣＬ＝Network Job Control Language)である事が必要となり、その一構想を提案する。

ネットワークに加入しているすべてのホスト・コンピュータが共通のＮＪＣＬを採用すると、特定のマシンに依存せぬ（machine－independent）ジョブの実行に対しては、各ホストのワークロード（workload）を調べ、最適なホストを選択し、そのホストにジョブおよびデータ・ファイルを転送（migration）して実行することも可能となる。さらに、ホストに障害が起った場合には、そのホスト向へのジョブを別のホストへ回すことも可能となる。ユーザ側から見ると、投入したジョブに対してターン・アラウンド時間が短縮され、一方、ネットワーク側から見ると、ジョブ実行のロード（load）が広域に渡って分散され、各ホストのロードが処理能力に応じて均等化（leveling）され、ネットワーク全体のスループットを増大せしめる事が可能となる。この機能はロード・レベリング（load－leveling）と呼ばれる。

コンピュータ・ネットワークの究極の目標は、ユーザがネットワーク中の全リソースを一つのコンピューティング・システムとしてみなすことができるようにすることである。これは仮想マシンまたは仮想ネットワークと呼ばれる。ＮＪＣＬはコンピュータ・ネットワークを仮想ネットワークへと発展させる第一ステップである。

6.2 ＮＪＣＬの概要

6.2.1 目　　的

ＮＪＣＬは、ユーザがネットワークの環境下でプログラムの実行やリソースの割付けの指令などを記述するためのハイ・レベルの共通の制御言語である。

ＮＪＣＬと既存のＪＣＬとの大きな違いは、プログラムを実行するサイト（ホスト）と必要なデータやファイルが存在するサイトが異なってもよい点である。

ＮＪＣＬは、以下に示す目的で開発された。

① ＮＪＣＬは、ネットワーク中の共通の標準の制御言語とする。

② 従来のアセンブラ言語風のＪＣＬではなく、高級言語としてのＮＪＣＬを設定する。

③ ＮＪＣＬは、バッチ・モードと同様に会話モードでも使用可能とする。

④ 複数サイトに渡るプログラムの実行やファイルを一元的に管理・制御する。

⑤ サイトのワークロードやファイルの存在場所を考慮して、ジョブの実行に最適なサイトを自動的に選択する。

⑥ 複数のサイトでそれぞれ実行されているプログラムの監視を容易に実現できるようにする。

⑦ 従来のファイル転送プロトコル機能を包含する。

⑧ 従来のリモート・バッチ・プロトコル機能を包含する。

⑨ そ の 他

6.2.2 言語の概要

セッションとは、ネットワーク内での一番大きな仕事の単位である。即ち、ユーザがネットワークの使用を開始してから一連のすべての仕事を終了するまでである。セッション・プログラムは、プログラムの実行順序とそのプログラムの実行に必要なデータ・カードやファイルの宣言の記述からなる。

セッション・プログラムは、図６－１に示すように構成される。

プログラムは、カードの第１～７２桁の範囲ならば自由に書くことができるが、空白も意味をもつので、語の切れ目などで区切り記号のないところには１字以上の空白を入れておかなければならない。

行がえは語の切れ目などでなら自由に行うことができる。したがって、１つの文を数行に分けて書いてもよく、そして、１行に２つ以上の文を書いてもよい。

（＊と＊）とで囲まれた文字列は注釈として扱われる。注釈は、空白を入れられるところならどこに書いてもよい。

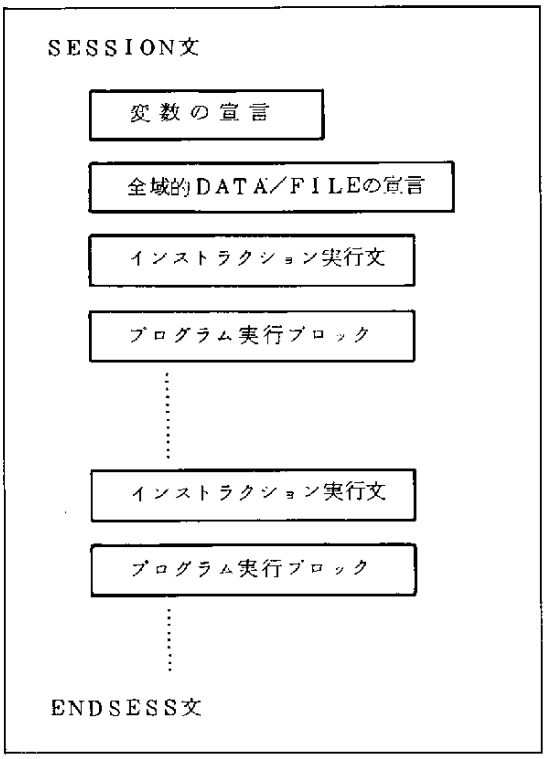


図６－１　セッション・プログラムの構造

使用する変数は、型宣言文で指定し、変数名と型を対応づける。

使用するデータ・カードとファイルはそれぞれＤＡＴＡ文とＦＩＬＥ文で宣言する。

実行文は、通常は記述されている順序で実行されるが、無条件分岐のＧＯ文や条件分岐のＩＦ文を用いることによって実行の順序の変更を指定することができる。

文の前に名札を付けることができ、ＧＯ文とＩＦ文でその名札を指定することによって、その文へ実行の順序を移すことができる。

プログラムの実行時に使用するデータ・カードやファイルはＤＡＴＡ文とＦＩＬＥ文で指定するが、会話モードでの使用の簡便さを考慮し、ファイル指定が省略されるとプライマリ・ファイル（Primary File）が指定されたと見なす。

文としては、次に示すものがある。

プログラム構造文

- ＳＥＳＳＩＯＮ文
- ＥＮＤＳＥＳＳ文
- ＢＥＧＩＮ文
- ＥＮＤ文

宣言文

- 型宣言文
  - ＩＮＴ文
  - ＣＨＡＲ文
  - ＳＴＡＴＵＳ文
- ＤＡＴＡ文
- ＦＩＬＥ文

プログラム実行文

- ＣＯＭＰＩＬＥ文
- ＬＩＮＫ文
- ＥＸＥＣＵＴＥ文
- ＲＵＮ文
- ＣＯＰＹ文
- ＲＥＮＡＭＥ文
- ＤＥＬＥＴＥ文
- ＤＥＳＣＲＩＢＥ文
- ＥＤＩＴ文

・ＰＲＩＮＴ文

・ＰＵＮＣＨ文

・ＸＢＡＴＣＨ文

・ＡＳＫ文

・ＣＡＮＣＥＬ文

インストラクション実行文

・代入文

・ＧＯ文

・ＩＦ文

・ＤＩＳＰＬＡＹ文

・ＤＥＬＡＹ文

・ＷＡＩＴ文

次に、ブロックと有効範囲の概念について述べる。

(1) ブロック

ブロックとは、セッション・プログラムの一部を構成しながら、他とは区別されている一連の文である。

ブロックには、

(1) プログラム実行ブロック

(2) サブセッション・ブロック

の２種類がある。

(2) プログラム実行ブロック

プログラム実行ブロックは、次に示す順序で構成される。

プログラム実行文

〔ＤＡＴＡ／ＦＩＬＥ文〕

⋮

〔ＤＡＴＡ／ＦＩＬＥ文〕

即ち、プログラム実行ブロックは、プログラム実行文で始まり、必要

ならばこのブロックのみで使用するデータ・カードをＤＡＴＡ文で、ファイルをＦＩＬＥ文で宣言して、終了する。

(3) サブセッション・ブロック

サブセッション・ブロックは、次に示す順序で構成される。

〔＜名札＞：〕ＢＥＧＩＮ　〔ＡＴ＜サイト指定子＞〕……；

⋮

ＥＮＤ〔＜名札＞〕；

即ち、サブセッション・ブロックは、ＢＥＧＩＮ文で始まり、ＥＮＤ文で終る一連の文であり、指定したサイトで一連の文が実行される。

ＢＥＧＩＮ文には、名札を付けても付けなくてもよく、ＥＮＤの後の名札はなくてもよいが、もし名札を付ける場合にはＢＥＧＩＮ文の名札と同じものを書く。

サブセッション・ブロックは必ず頭から実行され、ＧＯ文またはＩＦ文によってブロックの途中から実行してはならない。

サブセッション・ブロック内には、プログラム実行ブロックとインストラクション実行文のみしか含むことができない。

(4) ＤＡＴＡ文とＦＩＬＥ文の有効範囲

ＤＡＴＡ文とＦＩＬＥ文がセッション・プログラムのどの部分にあるかによって、その文の有効範囲が決められる。これらの文が最初のプログラム実行文よりも前にある場合には、すべてのプログラム実行文より参照することが可能であるが、この他の場所にある場合には、直前のプログラム実行文のみよりの参照に限定される。

6.2.3 簡単なセッション・プログラムの例

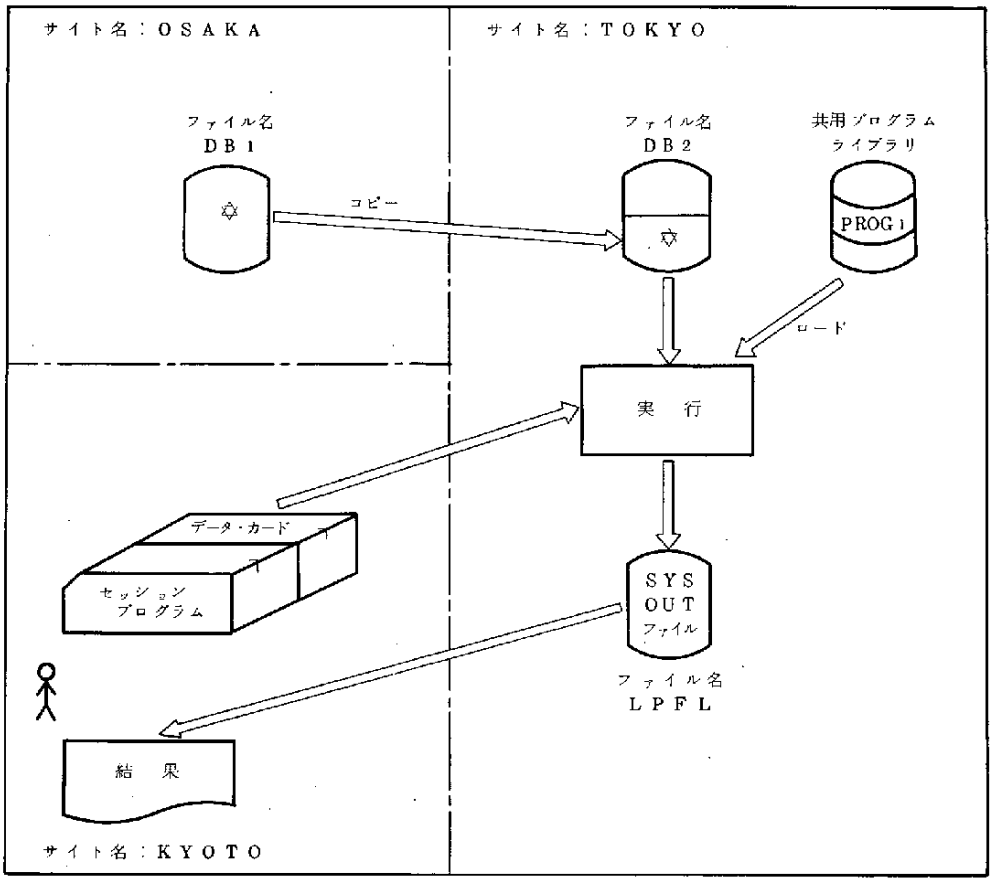
図６－２は、京都（ＫＹＯＴＯ）にいるユーザがセッションを制御し、大阪（ＯＳＡＫＡ）にあるファイルＤＢ１を東京（ＴＯＫＹＯ）にあるファイルＤＢ２に追加モードでコピーし、そのファイルを使用して、プログ

ラムＰＲＯＧ１を東京で実行させ、処理結果（ファイルＬＰＦＬ）を京都のラインプリンタに出力させる例である。

①は、セッション・プログラムの開始を示すＳＥＳＳＩＯＮ文である。ＮＪＰはセッション名、ＩＤのカッコ中のパラメータはそれぞれユーザ名、パスワード、課金番号である。

②は、セッション中で使用する整数型の変数を宣言するＩＮＴ文である。

③は、ＣＯＰＹ文であり、大阪にあるファイルＤＢ１の内容を東京にあるファイルＤＢ２にコピーする。但し、ＡＤＤ＝ＹＥＳが指定されているので、既にあるファイルの後に追加される。



(1)　ネットワーク・ジョブの構成例

```
SESSION NJP ID(NAME,PSWD,ACCNT);                          ①
INT CC (* Declare integer variable *);                    ②
COPY DB1/OSAKA TO DB2/TOKYO REPLY(CC)                     ③
     WHERE(ADD=YES);
IF CC <> 0 THEN GO FIN;                                   ④
EXECUTE @PROG1 AT TOKYO;                                  ⑤
%SYSIN: DATA WHERE(ESC='****');                           ⑥
    データ・カード                                        ⑦
****  (* End-of-data mark *)                             ⑧
%SYSLST: FILE LPFL WHERE(SYSOUT=PRINT);                   ⑨
%SYSDB: FILE DB2;                                         ⑩
FIN: ENDSESS (* End-of-session *);                        ⑪
```

(2) セッション・プログラムの例

図6－2 ネットワーク・ジョブ処理の例

④は、③のＣＯＰＹ文の実行が成功したか、不成功に終ったかが変数ＣＣにセットされているので、ＩＦ文によってそれをチェックしている。＜＞は不等号記号である。

⑤は、東京にあるプログラムＰＲＯＧ１をその場所で実行することを指令しているＥＸＥＣＵＴＥ文である。

⑥は、以下にデータ・カードが続くことを示すＤＡＴＡ文である。データ・カードの終りには、１カラム目から＊＊＊＊がさん孔されたカードが置いてある。％ＳＹＳＩＮは仮ファイル名であり、この名前を使って、プログラムＰＲＯＧ１より参照される。

⑦は、データ・カードである。

⑧は、データ・カードの終りを示すカードである。

⑨は、ファイルＬＰＦＬを宣言するＦＩＬＥ文である。このファイルに

対しては、ＳＹＳＯＵＴ＝ＰＲＩＮＴが指定されているので、実行完了後、京都にあるラインプリンタ装置に出力される。

⑩は、ファイルＤＢ２を宣言するＦＩＬＥ文である。

⑪は、セッション・プログラムの実行の終りと、記述の終りを示すＥＮＤＳＥＳＳ文である。名札ＦＩＮは、④のＩＦ文によって参照される。

このようにして、プログラムの実行をネットワーク内の望む所で行うことができる。

以上の例から解かるように、ＮＪＣＬは、ハイ・レベルのプログラミング言語と同じように、変数とＩＦ文を使用して、プログラムの実行順序を制御することが可能である。

6.2.4　プログラムの実行形態

プログラム実行文は、ユーザ・プログラムやシステム・プログラムの実行を指令する。

プログラムの実行形態としては、次の３つのモード（ mode ）がある。

① ＳＴＥＰ

② ＰＲＯＣＥＳＳ

③ ＪＯＢ

ＳＴＥＰモードは、その文によりプログラムが起動され、終了すると次の文の実行を開始する。

一方、ＰＲＯＣＥＳＳモードは、その文によりプログラムが起動されると、直ちに次の文の実行が開始され、以後併行処理が行われる。そのプログラムの実行が終了した時、指定した状態型変数にポストされ、ＷＡＩＴ文により終了の同期を取ることができる。

ＪＯＢモードは、その文によりプログラムが直ぐには実行されず、いったん実行サイトのジョブ実行待ちキューに登録され、このセッション・プログラムとは独立に実行される。

プログラムの実行形態は、次に示すＷＨＥＲＥ節中のＭＯＤＥパラメータで指定する。

$$\text{WHERE}(\cdots\cdots, \text{MODE}=\left\{\begin{array}{l}\underline{\text{STEP}}\\ \text{PROCESS}\\ \text{JOB}\end{array}\right\}, \cdots\cdots)$$

〔例〕

ＥＸＥＣＵＴＥ　ＡＢＣ　ＷＨＥＲＥ（ＭＯＤＥ＝ＪＯＢ）；

これは、ＡＢＣと呼ばれるファイルの中に入っているプログラムを、ＪＯＢモードで実行することを指令する例である。

6.2.5　プログラムの実行環境

プログラムを実行する時の実行環境を設定することができる。実行環境としては、実行優先権（ priority）、ＣＰＵ使用時間の上限（ processing　time）、経過時間の上限（ elapsed　time）、使用主記憶容量の上限（ core）、言語（ language）またはサブシステム指定などである。

プログラムの実行環境は、次に示すＷＨＥＲＥ節中のパラメータ（実行属性式と呼ばれる）で指定する。

ＷＨＥＲＥ（……，ＰＲＴＹ＝１～ｎの優先権番号

，ＰＴＩＭＥ＝t_1（単位：Million Instruction）

，ＥＴＩＭＥ＝t_2（単位：分）

，ＣＯＲＥ＝ｍ（単位：Kilo-Byte）

，ＬＡＮＧ＝言語名，……）

〔例〕

ＥＸＥＣＵＴＥ　ＡＢＣ　ＷＨＥＲＥ（ＰＲＴＹ＝１，ＰＴＩＭＥ＝６０）；

これは、ＡＢＣを実行する時、実行優先権を１、ＣＰＵ使用時間の上限を６０ＭＩ（Million Instruction）とすることを指令する例である。

実行属性がどの文で指定されたかによって、その有効範囲が決められる。ＳＥＳＳＩＯＮ文ではセッション全体を有効範囲とする実行属性を指定する。ＢＥＧＩＮ文ではサブセッション・ブロックを有効範囲とする実行属性を指定する。プログラム実行文ではその実行文の中だけの実行属性を指定する。同じ実行属性が指定された場合、ＰＲＴＹ、ＰＴＩＭＥ、ＥＴＩＭＥ、ＣＯＲＥの実行属性に関しては外側（広域）の指定が優先され、その他の実行属性に関しては内側（局所）の指定が優先される。

6.2.6　プログラム実行の完了コード

プログラムが、ＳＴＥＰまたはＰＲＯＣＥＳＳモードで実行された場合、プログラムの実行終了時に完了コード（completion code）がＲＥＰＬＹ項で指定された変数にセットされる。

この完了コードをＩＦ文でチエックし、次にどのようなアクションを取るかを決めることができる。２つ以上のプログラムの実行からなるセッションでは、二番目以降のプログラムの実行ができるかどうかは、先行のプログラムが正常に実行されたかに依存する。例えば、コンパイル／リンク／実行という一連の作業から成るセッションでは、先行するコンパイルとリンクが正常に終了しない場合、一般に実行しても意味がない。

完了コードは、次に示すＲＥＰＬＹ項で指定した変数にセットされる。

ＲＥＰＬＹ（｛＜整数型変数名＞／＜状態型変数名＞｝）

〔例〕

```
INT CC1, CC2;
COMPILE SRCFL TO OBJFL REPLY(CC1)
        WHERE(MODE=STEP);
IF CC1 <> 0 THEN GO ERR;
LINK OBJFL TO LOADFL REPLY(CC2)
```

```
        WHERE(MODE=STEP);
    IF CC2<>0 THEN GO ERR;
    EXECUTE LOADFL;
```

上の例では、まず最初にソース・ファイルＳＲＣＦＬをコンパイルし、コンパイルの完了コードが整数型変数ＣＣ１にセットされる。コンパイルが正常に終了すると、ＣＣ１にはゼロがセットされるので、ＩＦ文によってＣＣ１の内容をチェックし、異常終了した場合、後続の作業をスキップする。

プログラムがＪＯＢモードで実行された場合、実行サイトのジョブ実行待ちキューに登録完了後、ジョブ名と実行サイト名がＲＥＰＬＹ項で指定された変数にセットされる。

このジョブ名と実行サイト名を使用して、ジョブの実行状況の問合せ（ＡＳＫ文）、ジョブの取消し（ＣＡＮＣＥＬ文）、ジョブの出力指令（ＰＲＩＮＴ文とＰＵＮＣＨ文）などが可能となる。

ＲＥＰＬＹ項の形式は、次の通りである。

ＲＥＰＬＹ（＜文字型変数名１＞〔，＜文字型変数名２＞〕）

文字型変数名１にはジョブ名、文字型変数名２には実行サイト名がそれぞれセットされる。

〔例〕

```
    CHAR JOBNAME;
    EXECUTE JOBFILE REPLY(JOBNAME)
            WHERE(MODE=JOB);
```

これは、ファイルＪＯＢＦＩＬＥをＪＯＢモードで実行する例である。文字型変数ＪＯＢＮＡＭＥにはジョブ名がセットされる。

6.2.7　サイト指定

コンピュータ・ネットワークの環境下では、複数のサイトが存在してお

り、使用しようとしているファイルがどのサイトにあるか、プログラムをどのサイトで実行するかの指定ができなければならない。

サイト指定に関しては、ユーザがいちいち指定する繁雑さから開放するため、適切な省略時解釈がなされなければならない。ファイルのサイト指定の場合の省略時解釈は、そのＦＩＬＥ文が属するブロックを実行しているサイトを指定したと見なす。

プログラム実行文のサイト指定は、次に示すＡＴ句で指定する。

ＡＴ { ＜サイト名＞ | ?＜文字型変数名＞ | ＣＵＲＳＩＴＥ | ＢＥＳＴＳＩＴＥ }

サイト名はネットワーク中で定義されているサイトの名前である。

文字型変数名の場合には、変数の内容としてサイト名が入っていなければならない。

ＣＵＲＳＩＴＥはCurrent Site の略であり、セッション・プログラムを投入したサイトか、あるいはサブセッション・ブロックを実行しているサイトである。サイト指定が省略された場合、これが指定されたと見なされる。

ＢＥＳＴＳＩＴＥは、ネットワーク中の各ホストのロードと、そのプログラムの実行に必要なデータ・カードやファイルがどのサイトにあるかなどを調べ、最適なサイトを選択することを指令する。

ＦＩＬＥ文のサイト指定は、次に示すように指定する。

ＦＩＬＥ ＜ファイル名＞〔（＜メンバ名＞）〕〔／＜サイト名＞〕 …；

＜サイト名＞はネットワーク中で定義されたサイトの名前である。

サイト名が省略された場合、全域的ＦＩＬＥ文の時はセッション・プログラムが投入されたサイト、その他のＦＩＬＥ文はプログラム実行ブロックを実行しているサイトが指定されたと見なされる。

"／"は英語の" is on" の略記号である。

6.3 文

ＮＪＣＬの文には、次の４種類がある。

① プログラム構造文

② 宣言文

③ プログラム実行文

④ インストラクション実行文

以下に各文の機能について述べる。

文の構文に関しては、簡略化しており、厳密な構文は次節で示す。

6.3.1 プログラム構造文

セッション・プログラムは、次に示すＳＥＳＳＩＯＮ文で始まり、ＥＮＤＳＥＳＳ文で終了する。

```
SESSION ＜セッション名＞ ＩＤ(＜ユーザ名＞,
    ＜パスワード＞, ＜課金番号＞)
  〔WHERE(＜セッション実行属性式＞, ……)〕;
```

```
ENDSESS 〔＜セッション名＞〕;
```

サブセッション・ブロックは、次に示すＢＥＧＩＮ文で始まり、ＥＮＤ文で終了する。

```
BEGIN 〔ＡＴ＜サイト指定子＞〕
  〔WHERE(＜セッション実行属性式＞, ……)〕;
```

```
END 〔＜名札＞〕;
```

〔例〕

```
BEGIN  AT  OSAKA;
  :
END;
```

これは、ＢＥＧＩＮ文とＥＮＤ文で囲まれた文をＯＳＡＫＡで実行させる例である。

6.3.2 宣　言　文

(1) 型宣言文

セッション・プログラムの中で使う変数名や配列名は、次に示す文で宣言を行う。

```
{ INT  }{ <変数名> }
{ CHAR }{ <配列名> }, ……;
STATUS  <変数名>, ……;
```

ＩＮＴは整数型、ＣＨＡＲは文字型の変数及び配列を宣言し、ＳＴＡＴＵＳは状態型の変数を宣言する。変数名および配列名は８文字以下である。配列は１次元配列のみであり、添字の下限は１である。文字型変数が値としてとる文字列の長さは、８文字以内とする。

〔例〕

```
INT  A, BB(2);
STATUS  EVENT, ECB;
```

(2) ＤＡＴＡ／ＦＩＬＥ文

プログラム実行文で使用するデータ・カードやファイルは、次に示す文で宣言を行う。

〔＜仮ファイル名＞：〕 ＤＡＴＡ 〔＜ファイル名＞〕
〔ＷＨＥＲＥ（＜データ属性式＞，……）〕；

＜仮ファイル名＞： ＦＩＬＥ ＜ファイル名＞
〔（＜メンバ名＞）〕〔／＜サイト名＞〕
〔ＷＨＥＲＥ（＜ファイル属性式＞，……）〕；

〔例〕

```
%ＳＯＵＲＣＥ： ＤＡＴＡ
    WHERE(ＥＳＣ='＊ＥＯＤ＊'，ＣＯＤＥ＝ＢＣＤ)；
```

ＣＯＢＯＬソース・プログラム

＊ＥＯＤ＊

これは、ＣＯＢＯＬソース・プログラムをデータ・カードとする例である。ソース・プログラムはＢＣＤコード即ちＩＢＭ２６コードでパンチされており、ソース・プログラムの終りには＊ＥＯＤ＊カードが添えてある。

6.3.3 プログラム実行文

プログラム実行文は、ユーザ・プログラム、言語プロセッサ、ユーティリティなどを実行する。

プログラム実行文としては、以下で述べる１４種類がある。

(1) ＣＯＭＰＩＬＥ文

ＣＯＭＰＩＬＥ文は、言語プロセッサを実行させて、プログラムをソース形式からオブジェクト形式に変換する。

ＣＯＭＰＩＬＥ文の形式は、次の通りである。

```
COMPILE [ { <ソース・ファイル名>[/<サイト名1>] } TO ]
        [ { <ソース仮ファイル名>                } ]
[ { <オブジェクト・ファイル名>[/<サイト名2>] } ][AT<サイト指定子3>]
[ { <オブジェクト仮ファイル名>               } ]
[REPLY( { <整数型変数名>                                  } )]
        { <文字型変数名1>[, <文字型変数名2>]              }
[WHERE(<コンパイル実行属性式>, ……)];
```

〔例〕

```
COMPILE %SRC TO %OBJ
   WHERE(LANG=FORTRAN, LIST=NO);
%SRC:DATA WHERE(ESC='EOS');
```

FORTRANソース・プログラム

```
EOS (* END-OF-SOURCE *)
%OBJ:FILE OBJFL WHERE(……);
   ⋮
```

これは、仮ファイル名%SRCに入っているFORTRANソース・プログラムをコンパイルして、オブジェクト・プログラムを仮ファイル名%OBJに入れる例である。言語（LANG）はFORTRAN、ソース・リスト（LIST）は不必要である。

(2) LINK文

LINK文は、リンケージ・エディタを実行させて、プログラムをオブジェクト形式からロード形式（実行可能形式）に変換する。

LINK文の形式は、次の通りである。

```
LINK [ { <オブジェクト・ファイル名>[/<サイト名1>] } TO ]
     [ { <オブジェクト仮ファイル名>                 }    ]
[ [ { <ロード・ファイル名>[/<サイト名2>] } ] ] [AT<サイト指定子3>]
     [ { <ロード仮ファイル名>             }   ]
[REPLY( { <整数型変数名>                                    } )]
        { <文字型変数名1>[, <文字型変数名2>]                }
[WHERE(<リンク実行属性式>, ……)];
```

〔例〕

LINK OBJFL TO PROGLIB WHERE(MAP=YES);

これは、ファイル名ＯＢＪＦＬに入っているオブジェクト形式のプログラムをロード形式に編集して、ファイル名ＰＲＯＧＬＩＢに入れる例である。言語は既に定義されているとし、マップ・リスト（ＭＡＰ）は必要である。

(3) ＥＸＥＣＵＴＥ文

ＥＸＥＣＵＴＥ文は、ファイルからロード形式のプログラムを主記憶にロードし、そのプログラムを実行させる。ファイル名が省略された場合、指定されたプログラムはそのサイトの共用プログラム・ライブラリに入っていると仮定する。

ＥＸＥＣＵＴＥ文の形式は、次の通りである。

```
EXECUTE [ { <ロード・ファイル名>[(<メンバ名>)][/<サイト名1>] } ]
        [ { <ロード仮ファイル名>                              }   ]
        [ { <プログラム名>[/<サイト名1>]                      }   ]
[AT<サイト指定子2>][REPLY( { <整数型変数名>                                  } )]
                           { <状態型変数名>                                  }
                           { <文字型変数名1>[,<文字型変数名2>]               }
[WHERE(<実行属性式>, ……)];
```

〔例〕

EXECUTE @INQ／CTR1 AT CTR2；

これは、サイト名CTR1の共用プログラム・ライブラリにあるプログラム名INQを持って来て、サイト名CTR2で実行する例である。

(4) RUN文

RUN文は、COMPILE文、LINK文、EXECUTE文の3つの一連作業を行う。

RUN文の形式は、次の通りである。

```
RUN [{<ソース・ファイル名>〔／<サイト名1>〕
      <ソース仮ファイル名>}]
〔AT<サイト指定子2>〕〔REPLY( {<整数型変数名>
                              <文字型変数名1>〔,<文字型変数名2>〕} )〕
〔WHERE〔(<コンパイル実行属性式>, ……)〕,
〔(<リンク実行属性式>, ……)〕, 〔(<実行属性式>, ……)〕〕;
```

〔例〕

RUN MATRIX AT JIPDEC；

これは、JIPDECにあるファイル名MATRIXに入っているソース形式のプログラムをサイト名JIPDECで、コンパイル・リンクしてそのプログラムを実行する例である。

(5) COPY文

COPY文は、ファイル1の内容をファイル2にコピーする。

COPY文の形式は、次の通りである。

```
COPY [{<ファイル名1>〔／<サイト名1>
      <仮ファイル名1>}TO ]
```

```
{ <ファイル名2>〔/<サイト名2>〕 }
{ <仮ファイル名2>                }
〔REPLY({ <整数型変数名>                          }
        { <文字型変数名1>〔,<文字型変数名2>〕 }))〕
〔WHERE(<コピー実行属性式>, ……)〕;
```

〔例〕

COPY YRFILE/OSAKA TO MYFILE;

これは、サイト名OSAKAにあるファイル名YRFILEの内容をカレント・サイトにあるファイル名MYFILEを(そのファイルにすでに中味があればそれを消去し)コピーする例である。

(6) RENAME文

RENAME文は、ファイルまたはメンバの名前を変更する。

RENAME文の形式は、次の通りである。

```
RENAME{ <旧ファイル名>〔/<サイト名>〕TO<新ファイル名>                }
      { <ファイル名>(<旧メンバ名>)〔/<サイト名>〕TO<新メンバ名>      }
〔REPLY({ <整数型変数名>                          }
        { <文字型変数名1>〔, <文字型変数名2>〕 }))〕
〔WHERE(<リネーム実行属性式>, ……)〕;
```

〔例〕

RENAME FICHIER(ABC) TO XYZ;

これは、ファイル名FICHIER中のメンバ名ABCを新しいメンバ名XYZに変更する例である。

(7) DELETE文

DELETE文は、ファイルまたはメンバを消去する。

DELETE文の形式は、次の通りである。

```
DELETE <ファイル名>[(<メンバ名>)][/<サイト名>]
[REPLY(<整数型変数名>
       <文字型変数名1>[,<文字型変数名2>])]
[WHERE(<デリート実行属性式>,……)];
```

〔例1〕

```
DELETE OWNERFL/KYOTO;
```

これは、サイト名KYOTOにあるファイル名OWNERFLを消去する例である。

〔例2〕

```
DELETE PAYROLL(SATO);
```

これは、ファイル名PAYROLL中のメンバ名SATOを消去する例である。

(8) DESCRIBE文

DESCRIBE文は、カタログ(catalog)に登録されているファイルの名前の一覧表を、またはファイルの属性表を作成する。

DESCRIBE文の形式は、次の通りである。

```
DESCRIBE{CATLG[(<サイト名>)]
         FILE(<ファイル名>[/<サイト名>])}
[REPLY(<整数型変数名>
       <文字型変数名1>[,<文字型変数名2>])]
[WHERE(<ディスクライブ実行属性式>,……)];
```

〔例〕

```
DESCRIBE CATLG(NAGOYA);
```

これは、サイト名NAGOYAのカタログに登録されているすべての自分のファイルの名前の一覧表を作成する例である。

(9) ＥＤＩＴ文

ＥＤＩＴ文は、カード・ストリューム形式のソース・ファイルをライブラリ形式のソース・ファイルに編集したり、更新カードによりライブラリ形式のソース・ファイルを更新する。

ＥＤＩＴ文の形式は、次の通りである。

```
EDIT [{<ソース・ファイル名1>〔/<サイト名1>〕
      <仮ファイル名1>} TO]
  {<ソース・ファイル名2>〔/<サイト名2>〕
   <仮ファイル名2>}
  〔REPLY({<整数型変数名>
          <文字型変数名1>〔, <文字型変数名2>〕})〕
  〔WHERE(<エディト実行属性式>, ……)〕;
```

〔例〕

```
EDIT SRCFL TO SRCLIB/TOKYO;
```

これは、ユーザがいるサイトにあるカード・ストリューム形式のソース・ファイル名ＳＲＣＦＬの内容をサイト名ＴＯＫＹＯにあるファイル名ＳＲＣＬＩＢにライブラリ形式に編集して登録する例である。

(10) ＰＲＩＮＴ文

ＰＲＩＮＴ文は、ＪＯＢモードで実行したプログラムの印刷結果を、またはソース・ファイルの内容をラインプリンタ装置に出力する。

ＰＲＩＮＴ文の形式は、次の通りである。

```
PRINT [{JOB({<ジョブ名>
            ?<文字型変数名>}〔, <サイト指定子1>〕)
        FILE({<ソース・ファイル名>〔(<メンバ名>)〕〔/<サイト名1>〕
              <ソース仮ファイル名>})}]
```

〔AT＜サイト指定子2＞〕〔REPLY(＜整数型変数名＞)〕
〔WHERE(＜プリント実行属性式＞, ……)〕;

〔例〕

PRINT JOB(RB0001, OSAKA) AT CURSITE;

これは、サイト名OSAKAで実行したジョブ名RB0001をサイト名CURSITE(ユーザがいる場所の意味)にあるラインプリンタ装置に印刷結果を出力する例である。

(11) PUNCH文

PUNCH文は、JOBモードで実行したプログラムのさん孔結果を、またはソース・ファイルの内容をカードさん孔装置に出力する。

PUNCH文の形式は、次の通りである。

PUNCH{JOB({＜ジョブ名＞ / ?＜文字型変数名＞}〔, ＜サイト指定子1＞〕) / FILE({＜ソース・ファイル名＞〔(＜メンバ名＞)〕〔/＜サイト名1＞〕 / ＜ソース仮ファイル名＞})}
〔AT＜サイト指定子2＞〕〔REPLY(＜整数型変数名＞)〕
〔WHERE(＜パンチ実行属性式＞, ……)〕;

〔例〕

PUNCH FILE(PUNCHFL) AT NAGOYA;

これは、NAGOYAにあるファイル名PUNCHFLの内容をサイト名NAGOYAにあるカードさん孔装置に出力する例である。

(12) XBATCH文

XBATCH文は、Transparent Batchの略であり、ソース・ファイルを実行サイトのローカル・バッチ・モードで実行する。ソース・ファイル中のJCLは実行サイトの既存のそれであり、ジョブ・デック

の構成順序も既存のそれである。

ＸＢＡＴＣＨ文の形式は、次の通りである。

```
XBATCH [{<ソース・ファイル名>〔／<サイト名１>〕}]
       [{<ソース仮ファイル名>                  }]
〔ＡＴ<サイト指定子２>〕
〔REPLY(<文字型変数名１>〔,<文字型変数名２>〕)〕;
```

〔例〕

```
XBATCH %SYSIN AT TOKYO;
%SYSIN:DATA;
```

東京にあるコンピュータのＪＣＬで 書かれたジョブ・デック

```
＊ＥＯＤ＊
```

これは、サイト名ＴＯＫＹＯにあるコンピュータのＪＣＬで書かれたジョブ・デックをＴＯＫＹＯでローカル・バッチ・モードで実行する例である。＊ＥＯＤ＊カードはデータ・カードの終りを示す。

(13) ＡＳＫ文

ＡＳＫ文は、ＪＯＢモードで実行したプログラムの実行状況を問合せる。実行状況はＲＥＰＬＹ項で指定した変数に以下に示す整数値でセットされる。

値	意　味
−1	ＡＳＫ文が異常終了した
0	該当ジョブがない
1	ジョブは入力中である
2	ジョブは実行待である
3	ジョブは実行中である

4	ジョブは出力待である
5	ジョブは出力中である

ＡＳＫ文の形式は、次の通りである。

```
ASK JOB( { <ジョブ名>          } [, <サイト指定子>] )
         { ?<文字型変数名> }
[REPLY(<整数型変数名>)];
```

〔例〕

```
ASK JOB(NET0001, OSAKA);
```

これは、サイト名ＯＳＡＫＡで実行したジョブ名ＮＥＴ０００１の実行状況について問合せる例である。ＲＥＰＬＹ項が省略されているので、ユーザがこの文を投入した端末にそのジョブの実行状況が表示される。

(14) ＣＡＮＣＥＬ文

ＣＡＮＣＥＬ文は、ＪＯＢモードで実行したプログラムの実行あるいは出力を取り消す。取消しが成功したかどうかは、ＲＥＰＬＹ項で指定した変数に以下に示す整数値がセットされる。

値	意　味
－1	ＣＡＮＣＥＬ文が異常終了した
0	該当ジョブがない
1	ジョブは入力中である
5	ジョブは出力中である
6	ジョブの実行または出力が取り消された

ＣＡＮＣＥＬ文の形式は、次の通りである。

```
CANCEL JOB( { <ジョブ名>          } [, <サイト指定子>] )
          { ?<文字型変数名> }
[REPLY(<整数型変数名>)];
```

〔例〕

CANCEL JOB(NET0001, OSAKA) REPLY(STATE);

これは、サイト名ＯＳＡＫＡで実行しているジョブ名ＮＥＴ０００１を取り消す例である。整数型変数ＳＴＡＴＥには、取り消しが成功したかどうかがセットされる。

6.3.4 インストラクション実行文

インストラクション実行文は、主にセッションの流れを制御する。

インストラクション実行文としては、以下で述べる6種類がある。

(1) 代 入 文

代入文は、変数の値を変更する。

代入文の形式は、次の通りである。

{ { <整数型変数名> / <整数型配列要素> } =<算術式> / { <文字型変数名> / <文字型配列要素> } =<文字列式> } ;

まず、右辺の式の値を求める。算術式は値として整数をとり、文字列式は値として文字列をとるものである。

算術式の形式は、次の通りである。

{ 〔－〕{ <符号のない整数> / <整数型変数名> / <整数型配列要素> } / { <符号のない整数> / <整数型変数名> / <整数型配列要素> } { ＋ / － / ＊ / ／ } { <符号のない整数> / <整数型変数名> / <整数型配列要素> } }

文字列式の形式は、次の通りである。

```
{ <文字列>
  <文字型変数名>
  <文字型配列要素> }
```

〔例〕

```
I＝－２０；
K（３）＝X＋１；
NAME＝'NAMAE'；
```

この例のように、配列要素は配列名（添字式）という形に書き、文字列は引用符（'）で囲む。

(2) ＧＯ文

ＧＯ文は、指定された名札へ制御を移行する。

ＧＯ文の形式は、次の通りである。

```
ＧＯ　＜名札＞；
```

〔例〕

```
ＧＯ　Ｌ；
  ⋮
Ｌ：　ＲＵＮ　Ｘ；
```

(3) ＩＦ文

ＩＦ文は、判定を行なう。

ＩＦ文の形式は、次の通りである。

```
ＩＦ　＜比較式＞　ＴＨＥＮ　＜文＞　〔ＥＬＳＥ＜文＞〕
```

まず、比較式を評価する。この値が真ならばＴＨＥＮ以下の文を実行し、偽ならばＥＬＳＥ句が指定されていればその文を実行し、ＥＬＳＥ

句がないときにはそのまま次に進む。

比較式の形式は、次の通りである。

```
{ <算術式><比較作用素><算術式>
{ <文字列式> { = } <文字列式> }
             { <> }
             { >< }
```

比較作用素としては、＝（等しい）、＜＞または＞＜（等しくない）、＞（大きい）、＜（小さい）、＞＝または＝＞（大きいか等しい）、＜＝または＝＜（小さいか等しい）である。

〔例１〕

```
IF CC<>0 THEN GO LABEL;
```

これは、ＣＣ≒０のときは、名札ＬＡＢＥＬへ制御を移行し、ＣＣ＝０のときは何もしないで次の文へ進む例である。

〔例２〕

```
IF STATE>0 THEN DISPLAY('ERROR');
    ELSE EXECUTE STEP2;
```

これは、ＳＴＡＴＥ＞０のときは、コンソールに'ＥＲＲＯＲ'を表示し、ＳＴＡＴＥ≦０のときは、ファイルＳＴＥＰ２を実行する例である。

(4) ＤＩＳＰＬＡＹ文

ＤＩＳＰＬＡＹ文は、文字列をオペレータ・コンソールに表示する。

ＤＩＳＰＬＡＹ文の形式は、次の通りである。

```
DISPLAY(<文字列式>)〔REPLY(<整数型変数名>)〕
〔AT<サイト指定子>〕;
```

〔例〕

DISPLAY('P1 START') AT OSAKA;

これは、文字列'P1 START'をサイト名OSAKAのオペレータ・コンソールに表示する例である。

(5) DELAY文

DELAY文は、セッション・プログラムの実行を指定された時間（秒）だけ中止する。

DELAY文の形式は、次の通りである。

DELAY(<算術式>);

〔例〕

DELAY(60);

これは、セッション・プログラムの実行を60秒間だけ中止する例である。

(6) WAIT文

WAIT文は、PROCESSモードで実行されているプログラムの実行が終了するまで待つ。

WAIT文の形式は、次の通りである。

WAIT(<状態型変数名>, ……);

〔例〕

WAIT(ECB1, ECB2);

これは、状態型変数名ECB1で指定されたプログラムと、状態型変数名ECB2で指定されたプログラムの実行が終了するまで待つ例である。

6.4 ＮＪＣＬの構文法

文法を記述する方法にはいろいろあるが、ここでは直観的で見やすい図式（Syntax Diagram or Syntax Chart）で記述する。

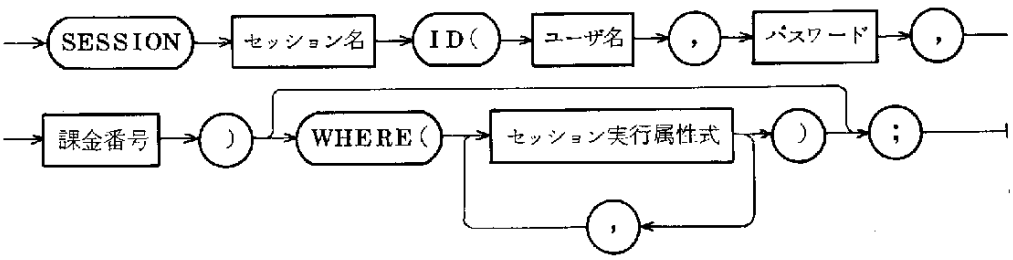
この図式では、入口からはいり、流れ線に沿って進んで出口までに出合うものを順に並べると、文法的に正しいすべての構文が得られるようになっている。円または楕円で囲んだ文字列は文字列そのものを示す。長方形で囲んだ超変数は更に別の箇所で定義されている。⌒ に書かれた数字はその線を通過できる最大回数を示す。

なお、煩雑になるので、ここでは、注釈に関する文法は省略してある。

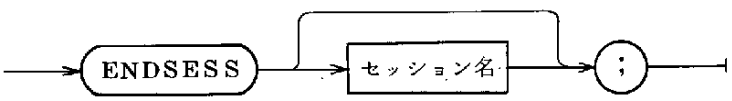
それぞれの属性式に関する文法は、紙面のつごうで省略する。

プログラム

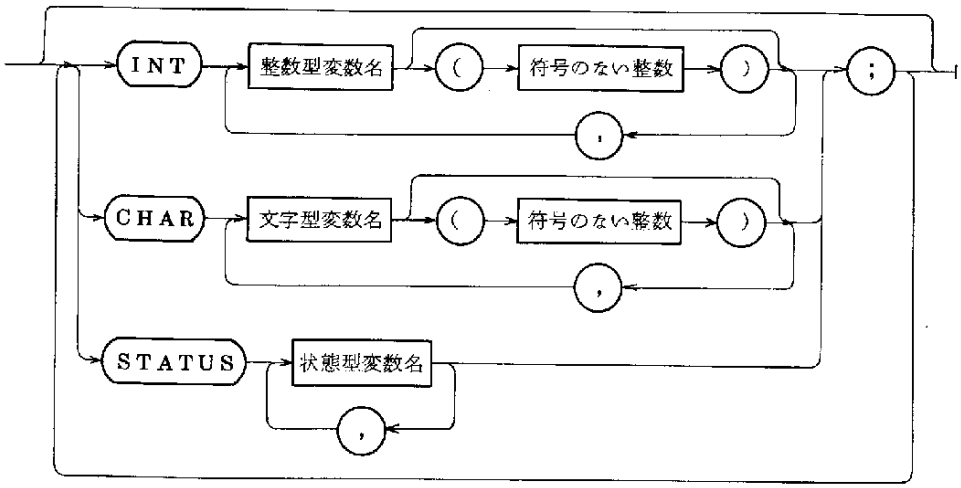
SESSION文



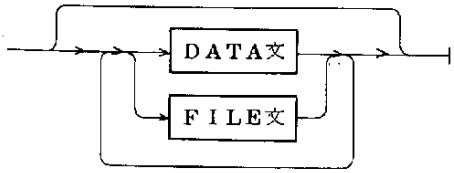
ENDSESS文



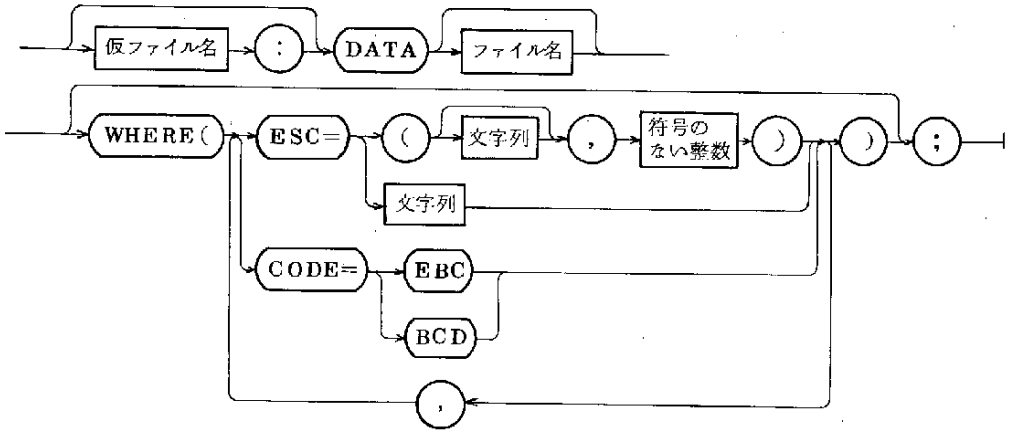
変数の宣言



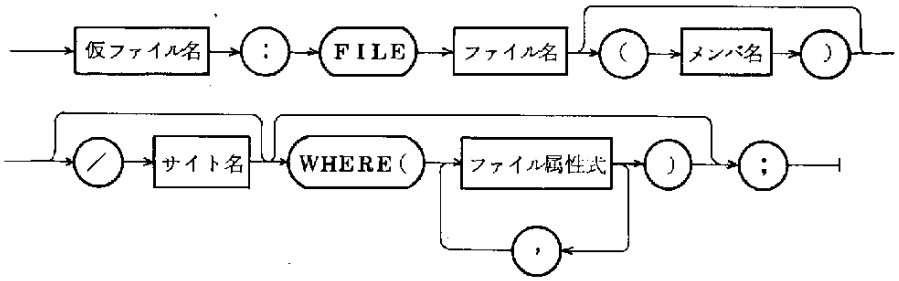
全域的DATA／FILEの宣言



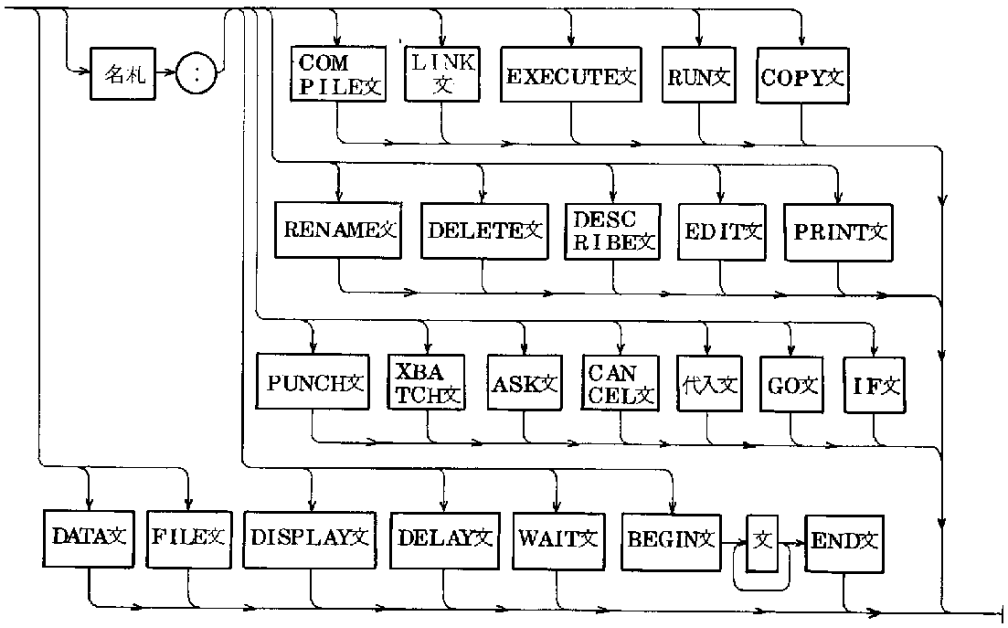
DATA文



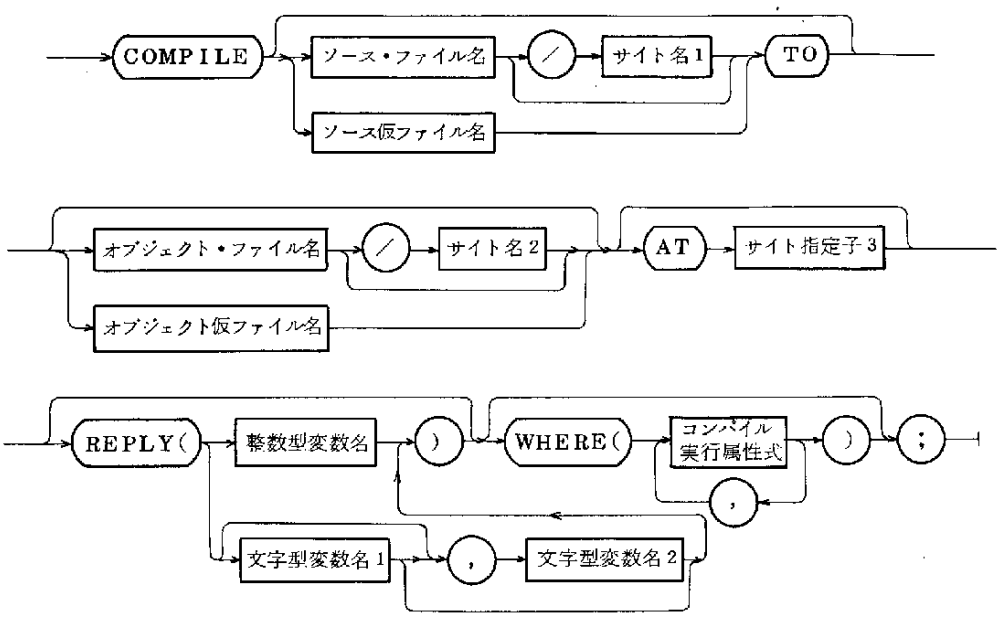
FILE文


仮ファイル名
:
FILE
ファイル名
(
メンバ名
)
/
サイト名
WHERE (
ファイル属性式
)
;
,

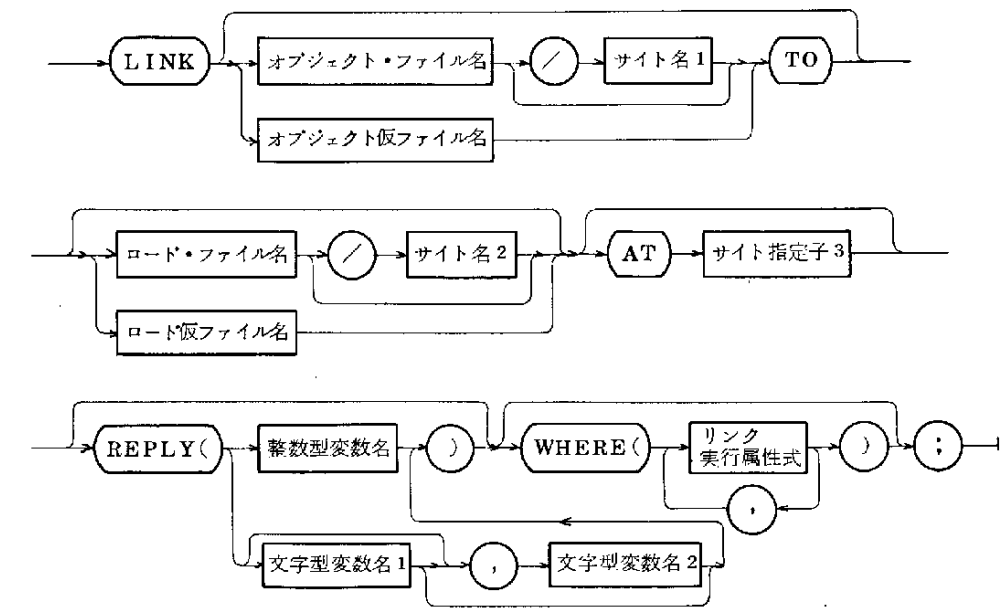
文


名札
:
COM PILE文
LINK 文
EXECUTE文
RUN文
COPY文
RENAME文
DELETE文
DESC RIBE文
EDIT文
PRINT文
PUNCH文
XBA TCH文
ASK文
CAN CEL文
代入文
GO文
IF文
DATA文
FILE文
DISPLAY文
DELAY文
WAIT文
BEGIN文
文
END文

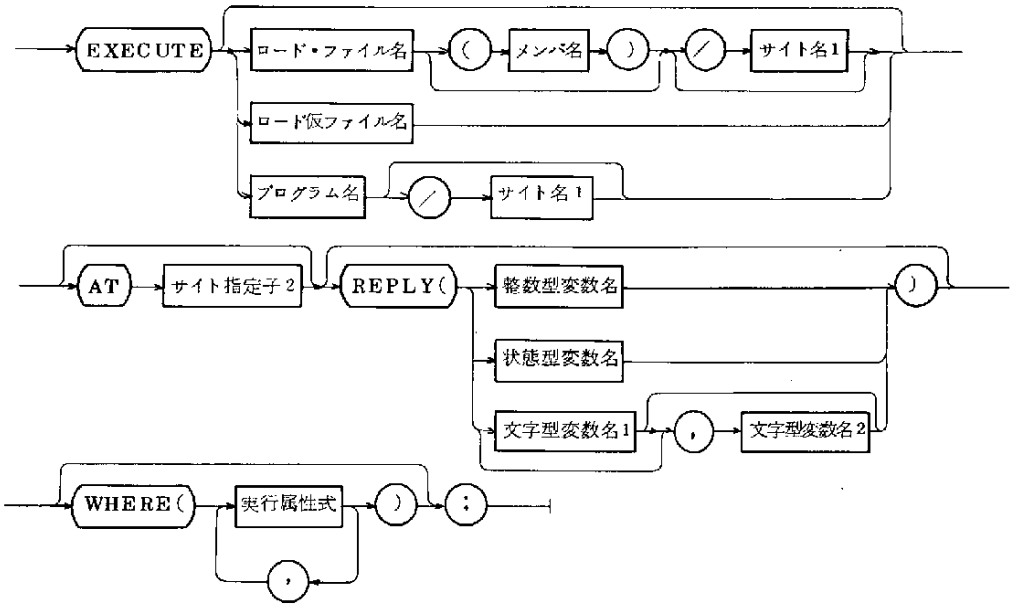
COMPILE文



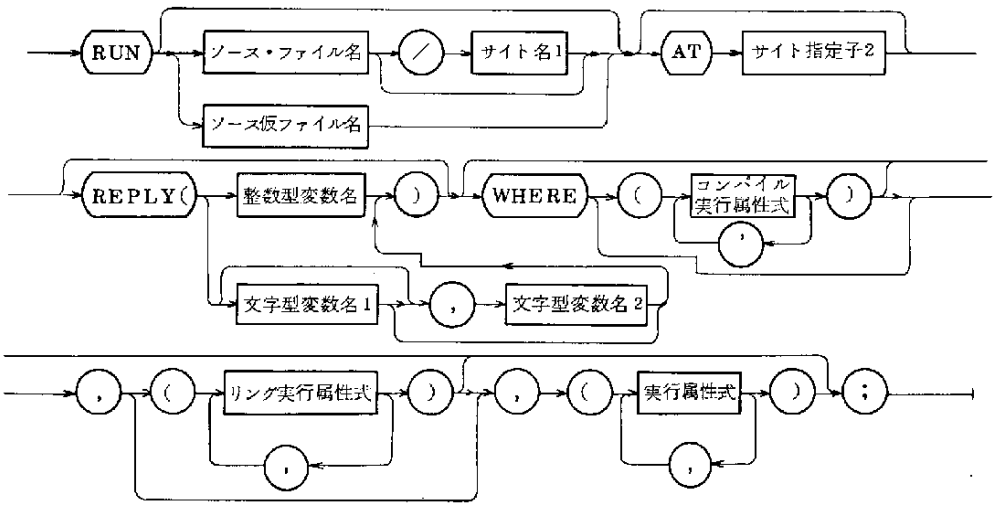
LINK文



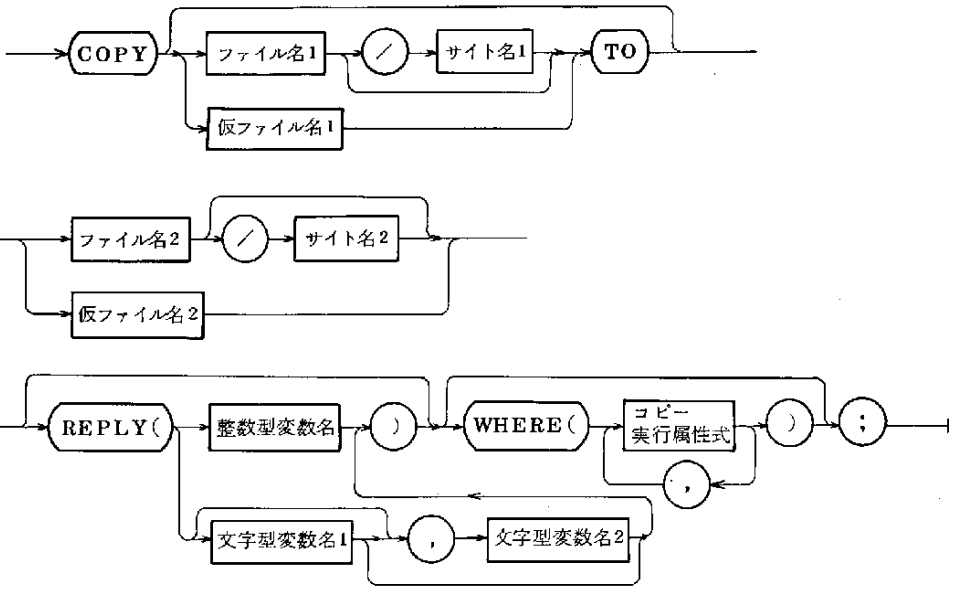
EXECUTE文



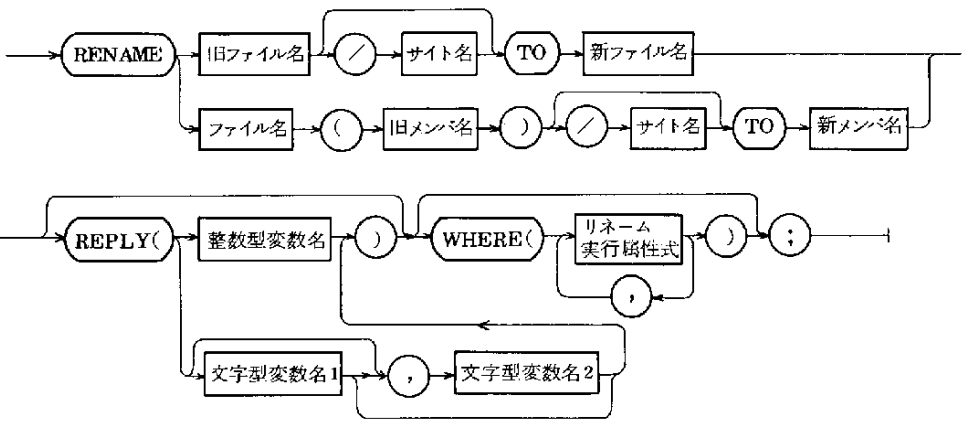
RUN文



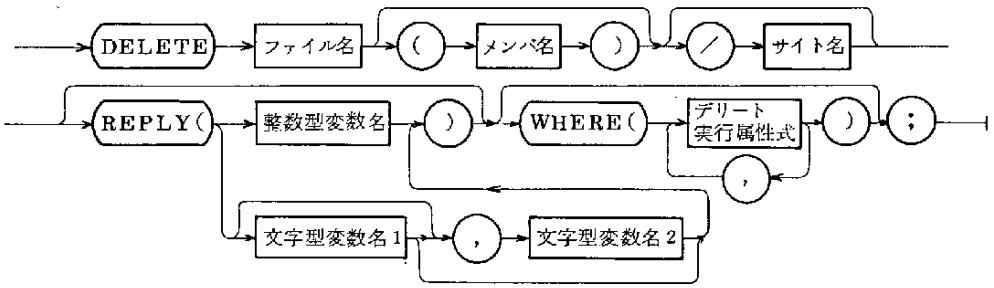
COPY文



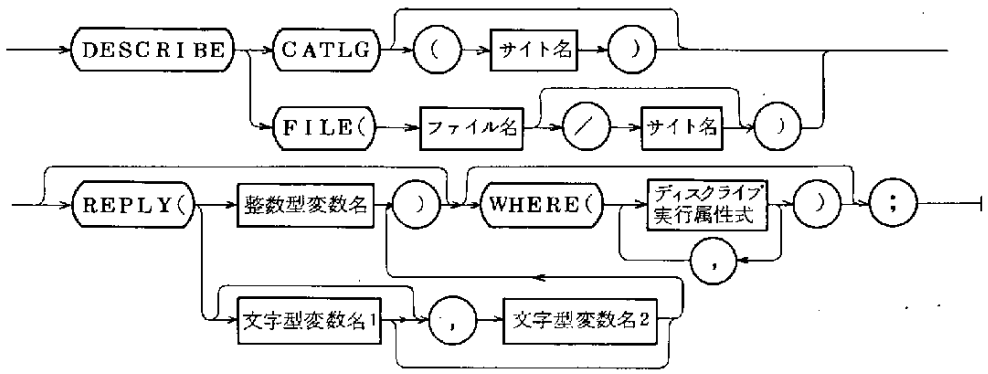
RENAME文



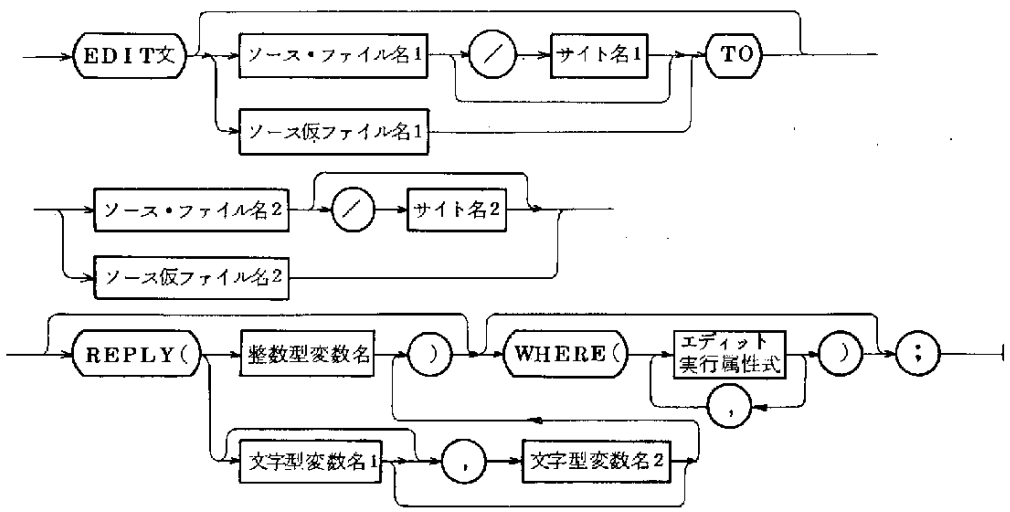
DELETE文



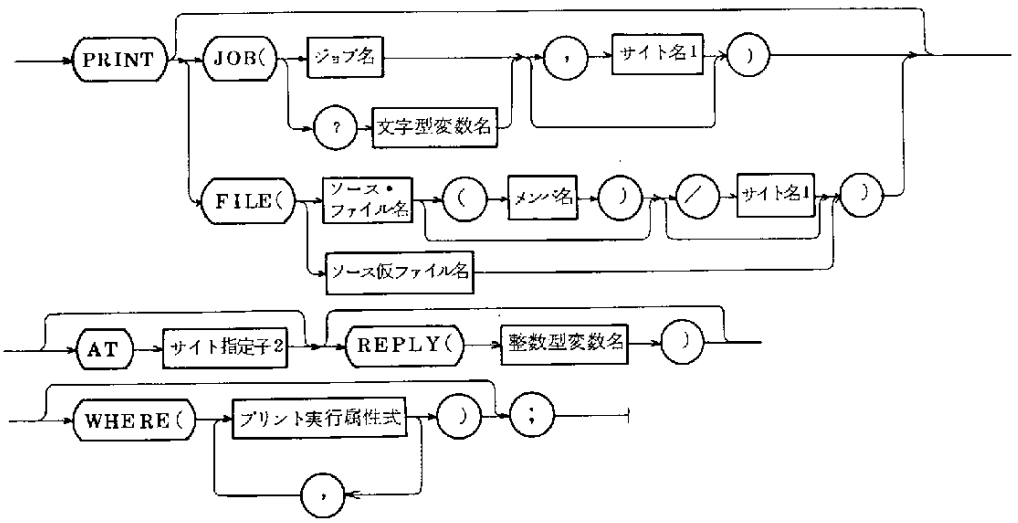
DESCRIBE文



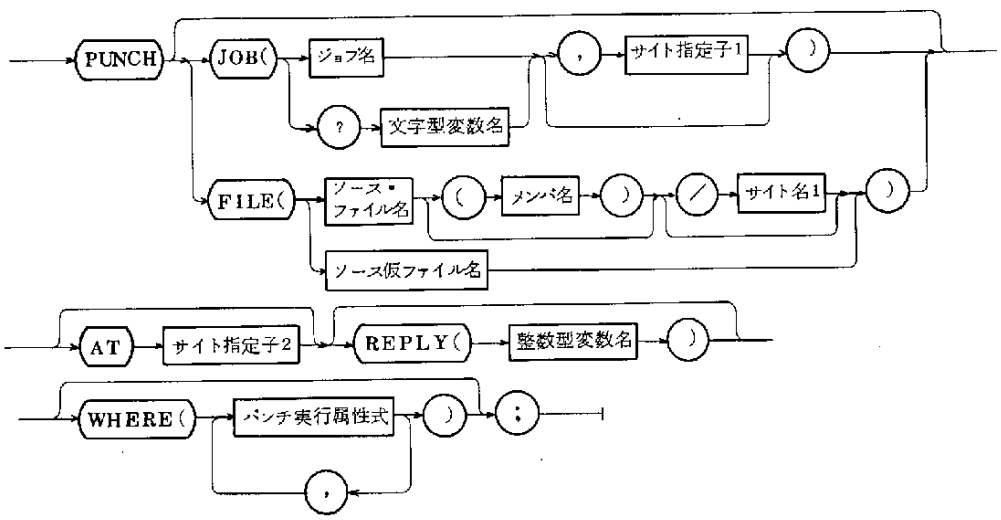
EDIT文



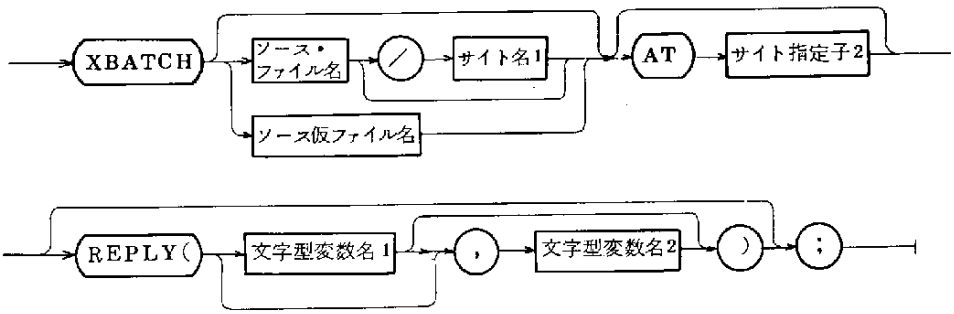
PRINT文



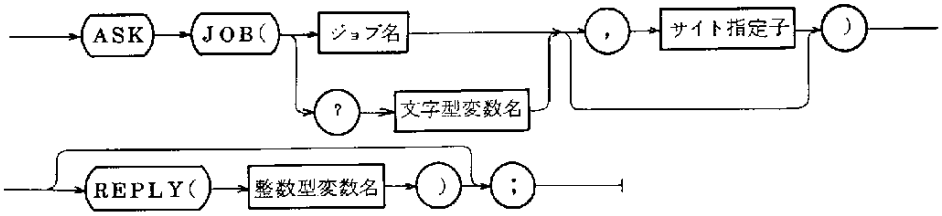
PUNCH文



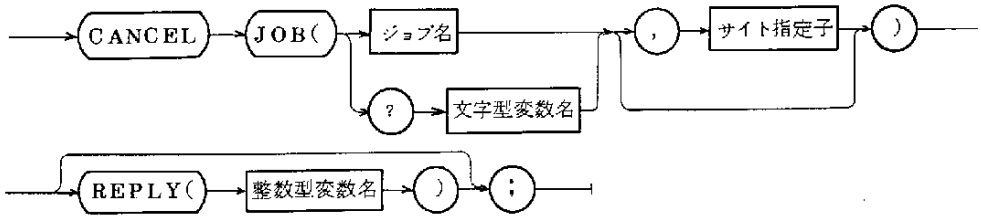
XBATCH文



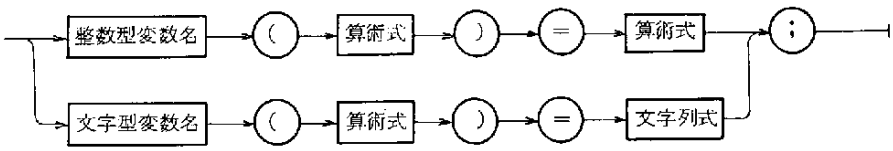
ASK文



CANCEL文

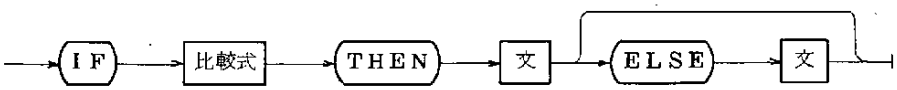


代入文

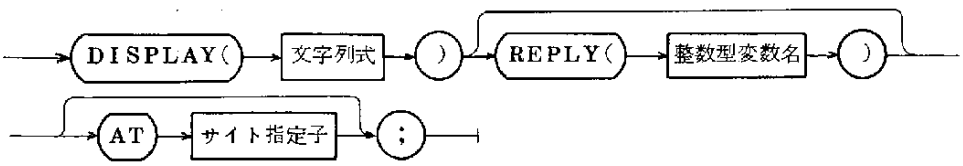


GO文

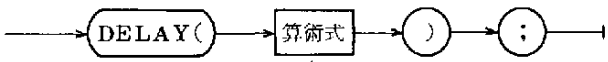
IF文



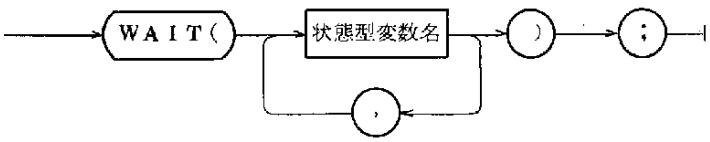
DISPLAY文



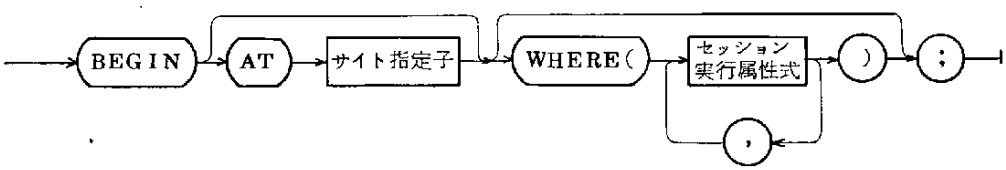
DELAY文



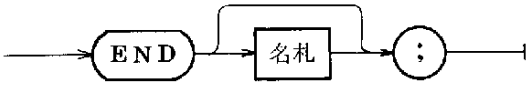
WAIT文



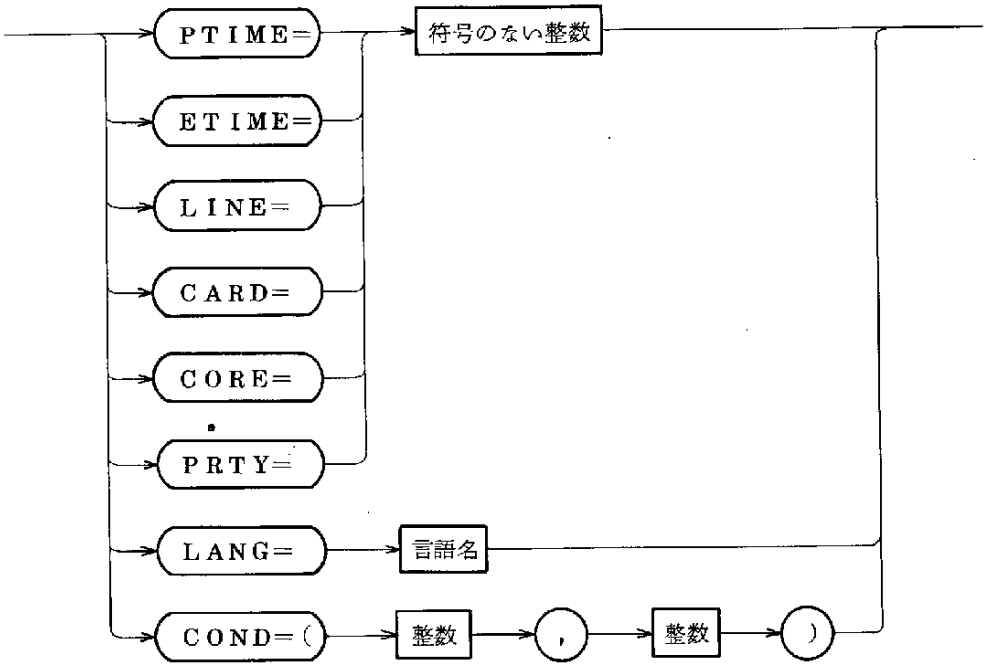
BEGIN文



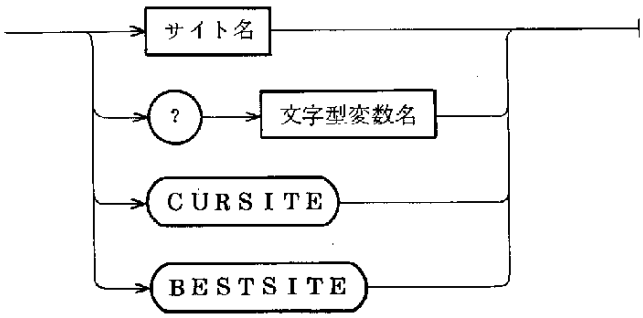
END文



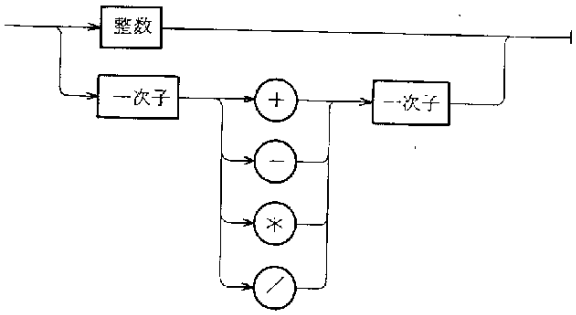
セッション実行属性式



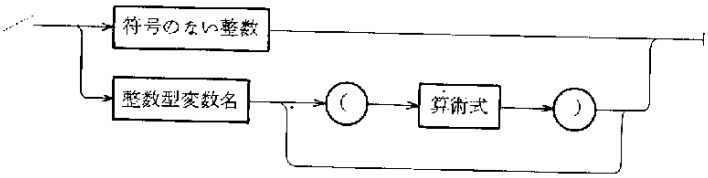
サイト指定子



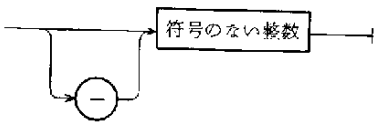
算術式



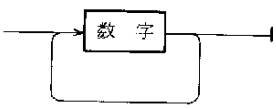
一次式



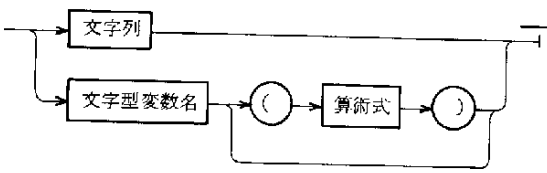
整　数



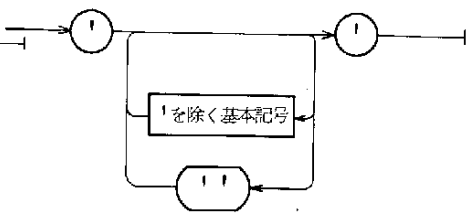
符号のない整数



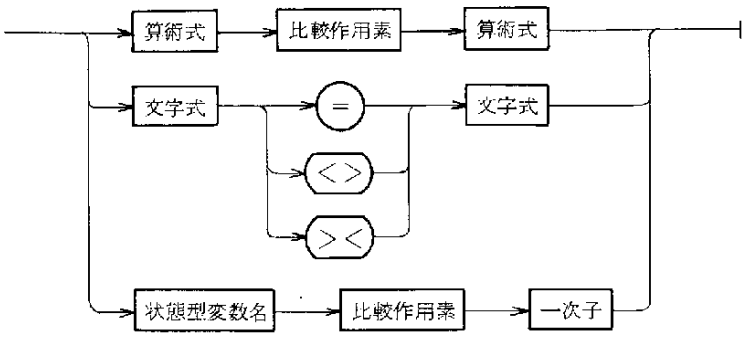
文字式



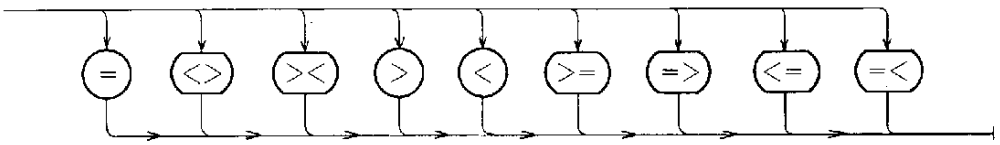
文字列



比較式

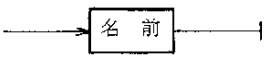


比較作用素

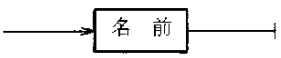


セッション名

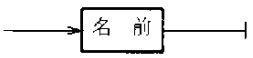
ジョブ名



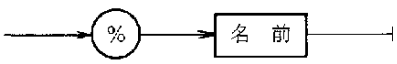
変数名



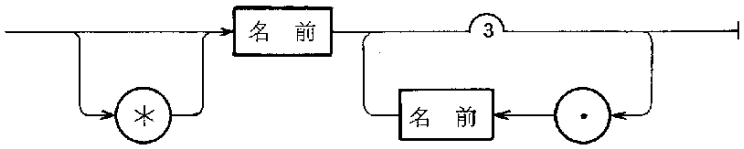
名札



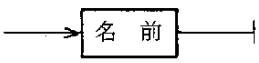
仮ファイル名



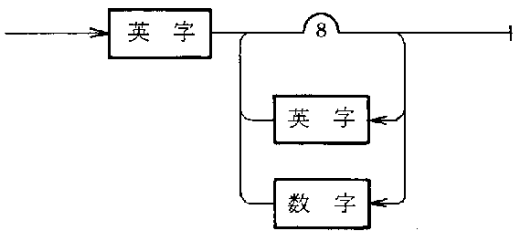
ファイル名



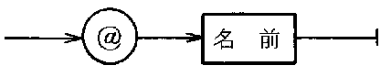
メンバ名



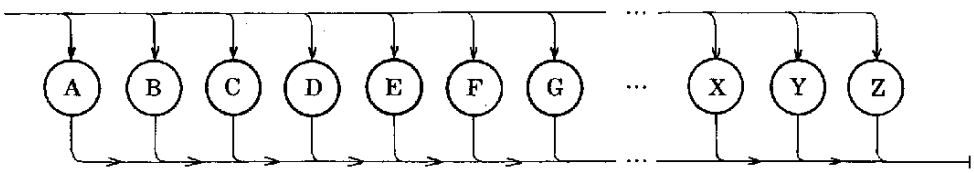
名 前



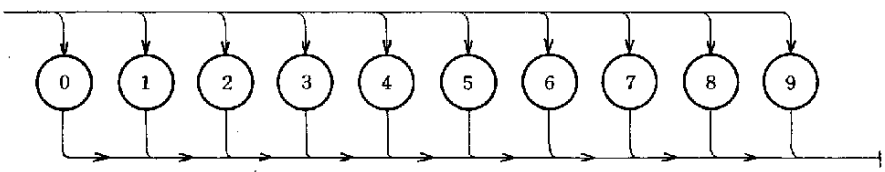
プログラム名



英 字



数 字



6.5 インプリメントの方法

6.5.1 ネットワーク・オペレーティング・システム

ネットワーク・オペレーティング・システム（ＮＯＳ＝Network Operating System）は、ユーザに対してＮＪＣＬを介して仮想ネットワーク環境をサポートする。即ち、ＮＯＳはネットワーク中の各ホストにそれぞれ存在するリソースを一元的に管理し、ユーザにはそのリソースがどのサイトに存在するかを意識させず、かつ同一の手続でアクセス可能にする。

ＮＯＳは、次に示す３つの主要な構成要素からなる（図６－３参照）。

- NFES（Network Front-End System）
- NCIC（Network Control/Information Center）
- NIP（Network Interface Program）

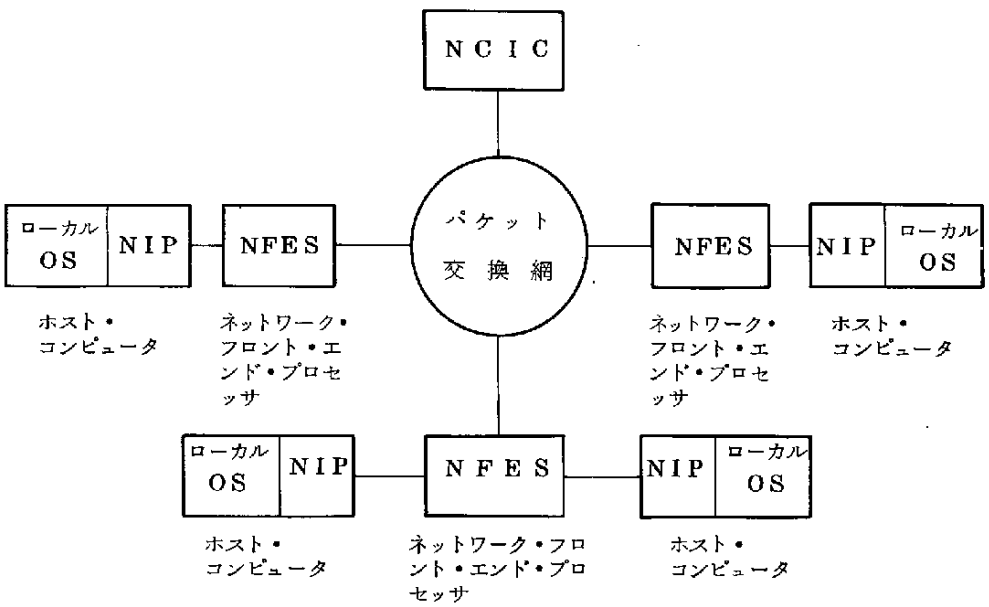


図６－３　ＮＯＳの構成要素

ＮＦＥＳは、ホスト・コンピュータとパケット交換網の間に設置されたＮＦＥＰ（Network Front-End Processor）と呼ばれるミニコンピュータの制御プログラムである。ユーザはこのＮＦＥＰに接続された端末やカード読取装置からセッション・プログラムを投入する。即ち、ＮＦＥＰはユーザに対するネットワークの出入口である。

ＮＣＩＣは、ネットワーク中に唯一つ存在し、各ホストのリソース情報、ロード状況、アカウンティング情報などを刻々収集し、管理を行っている。ＮＦＥＳは、セッション・プログラムが投入された時、サイト指定がＢＥＳＴＳＩＴＥの場合、このＮＣＩＣに問合せ、最適なサイトを紹介してもらう。

ＮＩＰは、ホスト・コンピュータ内に常駐し、ネットワーク・サービスを行なうため、既存のＯＳとのインタフェースを取り、ネットワークとの交信を司どる。ＮＩＰは、ホストがネットワーク・サービスを行なうために追加された制御プログラムである（ここでは、既存のＯＳには手を入れないと仮定する）。ＮＩＰはＮＦＥＳから既存のＪＣＬで書かれたジョブ・デックを受け取り、ジョブ・デックをリモート・バッチ管理などに渡して実行する。また、ハイ・レベル・プロトコル（ファイル転送、データ・アクセスなど）機能に関しては、自分自身でサービスを行なう。

ホスト・コンピュータとＮＦＥＰは、図６－４に示す形態で接続されると仮定する。

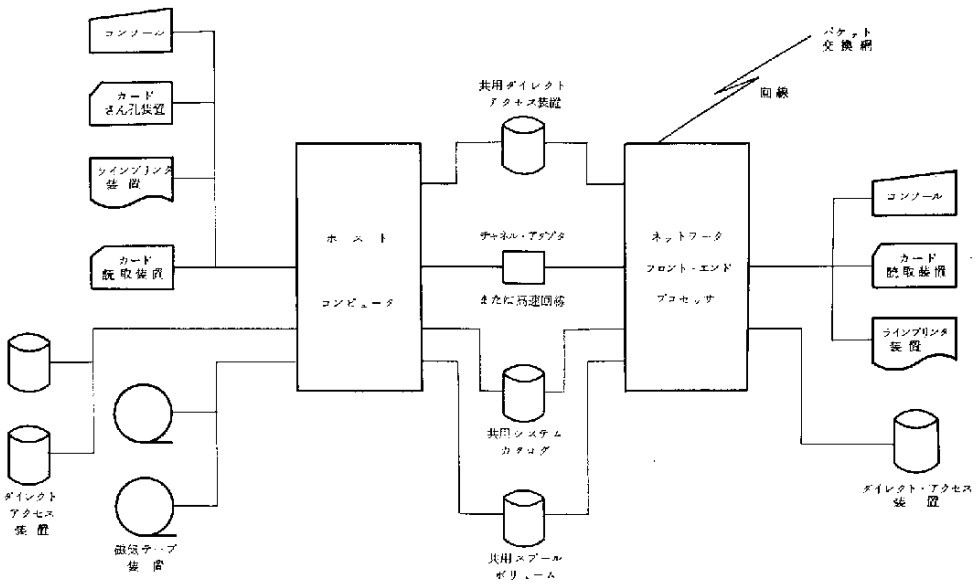


図6－4　ホスト・コンピュータとNFEPの接続例

6.5.2　ネットワーク・フロント・エンド・システム

ＮＦＥＳは、投入されたセッション・プログラムに対して、以下に示す一連の作業を行なう。

- ＮＪＣＬで書かれたセッション・プログラムをＩＬＳＣ（Intermediate Language for Session Control）と呼ばれるインタプリート可能な中間言語にコンパイルする。
- 実行サイトに接続されている相手ＮＦＥＳにこのＩＬＳＣプログラムの一部を転送する。
- このＩＬＳＣプログラムを受け取った相手ＮＦＥＳは、そのホスト用のローカルＪＣＬに変換し、ＪＣＬストリームをホストに転送する。
- プログラムの実行結果をユーザが指定したサイトのラインプリンタ装

置などに出力する。

ＮＦＥＳは、ユーザとはＮＪＣＬで、ＮＦＥＳとはＩＬＳＣで、ホスト・コンピュータとはローカルＪＣＬでそれぞれ会話を行なう（図６－５参照）。ユーザはＮＪＣＬ以外に、特定のホスト・コンピュータを指定して、そのホストのローカルＪＣＬで会話をすることも可能である。

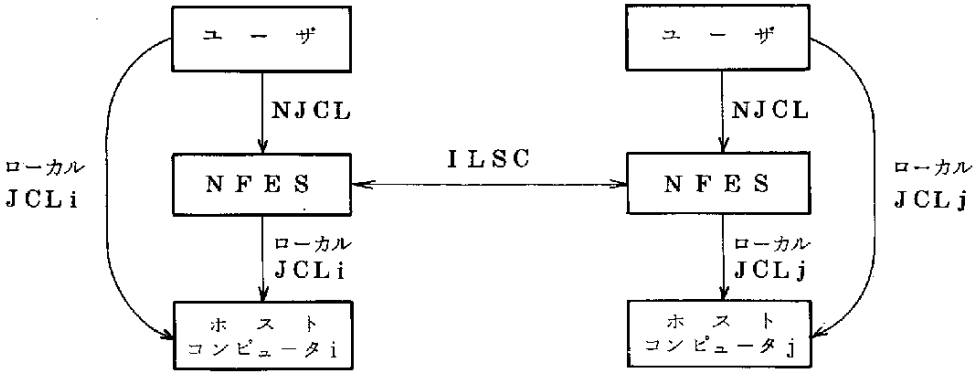


図６－５　各レベルでの会話用制御言語

ＮＦＥＳは、以下に示す構成要素からなる（図６－６参照）。

- Kernelは、ＮＦＥＳ全体の制御・管理を行なう。
- Network Interfaceは、パケット交換網との通信インタフェースである。
- Host Interfaceは、ローカル・ホストとの通信インタフェースである。
- Terminal Interfaceは、カード読取装置、ライン・プリンタ装置、端末装置との通信インタフェースである。
- Loggerは、他のＮＦＥＳからのプロセス起動用のオーダ（order）を受信し、指定されたプロセスを起動する。

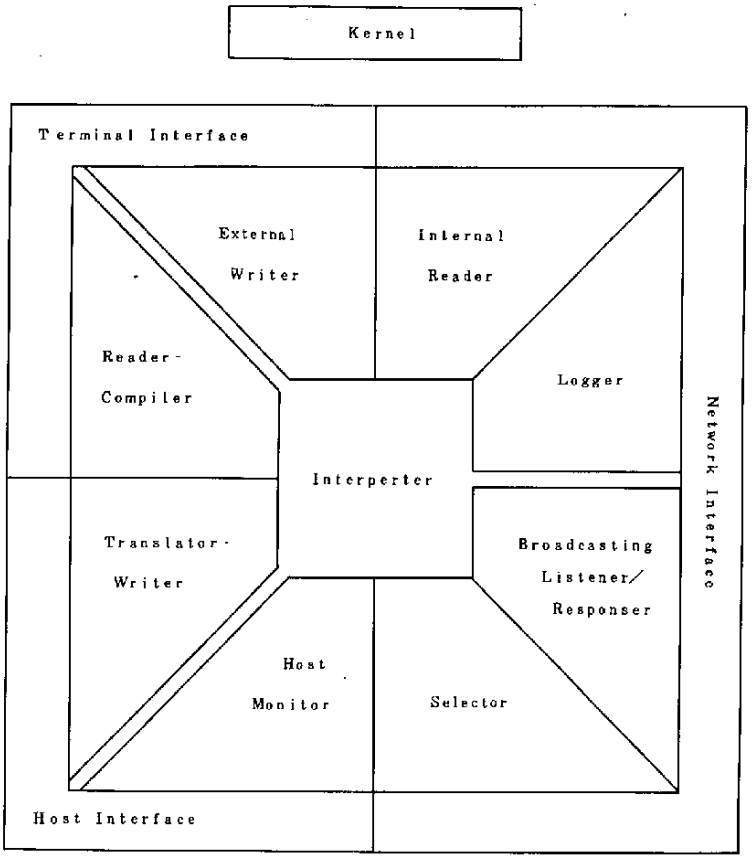


図6－6　ＮＦＥＳの構成

- Host Monitorは、ローカル・ホストのロード状況などをモニタリングする。
- Broadcasting Listener／Responserは、ＮＣＩＣからの問合せを受信し、Host Monitorに問合せてその応答をＮＣＩＣに返信する。

以上のプログラムは、常駐プロセスである。

- Reader—Compilerは、カード読取装置などよりＮＪＣＬで書かれたセッション・プログラムを読み込んで、ＩＬＳＣプログラムにコンパイ

ルする。

- Internal Reader は、別のＮＦＥＳから送られて来たＩＬＳＣで書かれたプログラム実行ブロックやサブセッション・ブロックを読み込む。
- Translator−Writerは、ＩＬＳＣで書かれたプログラム実行ブロックやサブセッション・ブロックをローカルのＪＣＬに変換し、ホストへ書き出す。
- External Writer は、プログラムの実行結果をライン・プリンタ装置などに出力する。
- Interpreterは、ＩＬＳＣプログラムを解読し、実行する。

以上のプログラムは、非常駐プロセスである。

- Selectorは、ロード状況をHost Monitorと ＮＣＩＣへ問合せ、実行サイトを決定するサブルーチンである。

Reader−Compilerは、入力ストリューム中に新しいセッション・プログラムを検出した時、ＰＳＣＴ（Permanent Session Control Table）の中にエントリを創成する。このエントリは、セッション・プログラムの実行が終了し、実行結果をライン・プリンタ装置やカードさん孔装置などに出力し終った時に、削除される。

ＰＳＣＴの中に新しいエントリを創成した後、セッション・プログラムはＩＬＳＣプログラムにコンパイルされる。コンパイル終了後、Interpreterを起動し、ＩＬＳＣプログラムのインタプリテーションを開始する。

ＩＬＳＣオーダがＲＦＰ（Request For Process）の場合、リモート（相手）側のLoggerにそのオーダを送る。これを受信したリモート側Loggerは、ＳＳＣＢ（Sub−Session Control Block）を創成し、Internal Reader を起動する。ソース側のInterpreterはリモート側のInternal Reader と同期を取った後、実行に必要な一連のＩＬＳＣオーダを送信する。リモート側では、一連のＩＬＳＣオーダを受信後、

Interpreterを起動し、インタプリテーションを開始する。ソース側のInterpreterとリモート側のInterpreterは、お互に同期を取りながら、それぞれのＩＬＳＣオーダを解読・実行する。

この過程を図６－７に示す。

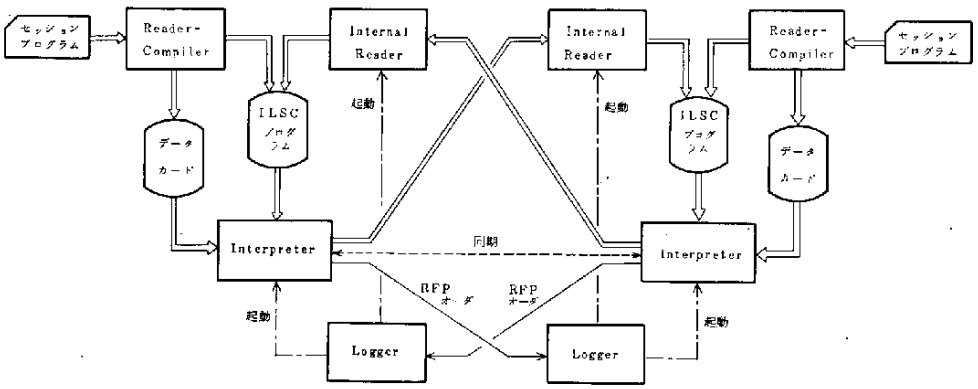


図６－７　セッション・プログラムの実行過程

6.5.3　ＩＬＳＣプログラム

セッション・プログラムは、Reader－CompilerによってＩＬＳＣプログラムにコンパイルされる。ＩＬＳＣプログラムは、図６－８に示すように、大きく５つのパートに分かれている。

辞　書　パ　ー　ト
セッション・パート
変数ディスクリプタ パ　ー　ト
ＤＡＴＡ／ＦＩＬＥ パ　ー　ト
オ　ー　ダ　・　パ　ー　ト

図６－８　ＩＬＳＣプログラムの構造

(1) 辞書パート

辞書パートには、次に示すものが含まれている。

- ＩＬＳＣプログラムの全長
- セッション・パートの開始位置
- 変数ディスクリプタ・パートの開始位置
- ＤＡＴＡ／ＦＩＬＥパートの開始位置
- オーダ・パートの開始位置
- そ　の　他

(2) セッション・パート

セッション・パートはＳＥＳＳＩＯＮ文によって作成される。

(3) 変数ディスクリプタ・パート

変数ディスクリプタ・パートは、セッション・プログラム中でユーザが宣言した変数の変数ディスクリプタが格納されている。

変数ディスクリプタのフォーマットを図６－９に示す。

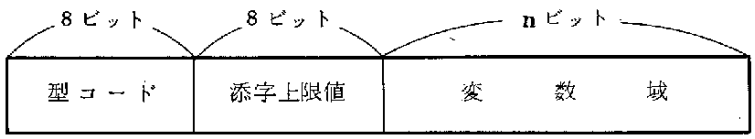


図６－９　変数ディスクリプタのフォーマット

型コードを次に示す。

型コード	意　味
1	整数型
2	文字型
3	状態型

添字上限値は、宣言された添字の上限値であり、単純変数の場合には０がセットされる。

変数域には、その変数に対する領域が確保される。変数1個につき、次に示す大きさの領域が確保される。

型	領域の大きさ
整数型	16ビット
文字型	64ビット
状態型	32ビット

〔例〕

```
INT  I, K(2);
CHAR  STRING;
```

に対しては、次に示す変数ディスクリプタが作成される。

(4) DATA／FILEパート

DATA／FILEパートは、DATA文またはFILE文によって作成される。

(5) オーダ・パート

オーダ・パートは、プログラム実行文とインストラクション実行文によって作成される。

オーダのフォーマットを図6－10に示す。

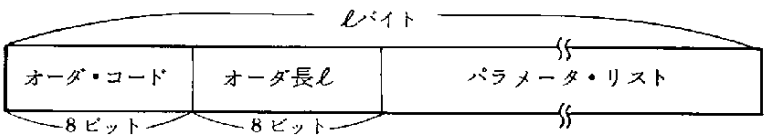


図6－10　オーダのフォーマット

オーダとしては、

RFP（Request For Process）

RFC（Request For Connection）

CLS（Close）

XORD（Transfer Order）

などがある。

6.5.4 ロード・レベリング

ネットワーク中には、処理能力以上のジョブが待っているホストもあれば、一方、処理すべきジョブがほとんどないホストもあり、ある特定のホストにジョブ・ロードが集中する傾向にある。その結果、オーバロード（overload）状態にあるホストのサービス率は低下し、一方、アンダロード（underload）状態にあるホストの効率が低下する。オーバロード状態にあるホストのロードの一部をアンダロード状態にある別のホストに転送することによって、効率とサービス率の両方が改善される。

これは、ロード・レベリング（load−leveling）またはロード・シェアリング（load sharing）と呼ばれる。

投入されたジョブをどのサイトで実行するのが最適かは、次に示すパラメータによって決定される（図6−11参照）。

① ジョブ・モデル

- 実行量：$\hat{X}_P$（MI＝Million Instruction）
- 入力データ長：$\underline{X}_I$（ビット）
- 出力データ長：$\underline{X}_O$（ビット）

ここで、

$$\underline{X}_I = \begin{bmatrix} X_{I_1} \\ X_{I_2} \\ \vdots \\ X_{I_N} \end{bmatrix} \quad , \quad \underline{X}_O = \begin{bmatrix} X_{O_1} \\ X_{O_2} \\ \vdots \\ X_{O_N} \end{bmatrix}$$

である。X_{I_i}はサイトｉにある入力データであり、サイトｉに入力データがない場合は、$X_{I_i}=0$である。X_{O_i}はサイトｉへの出力データであり、サイトｉへの出力データがない場合は、$X_{O_i}=0$である。Ｎはネットワーク中の総サイト数である。

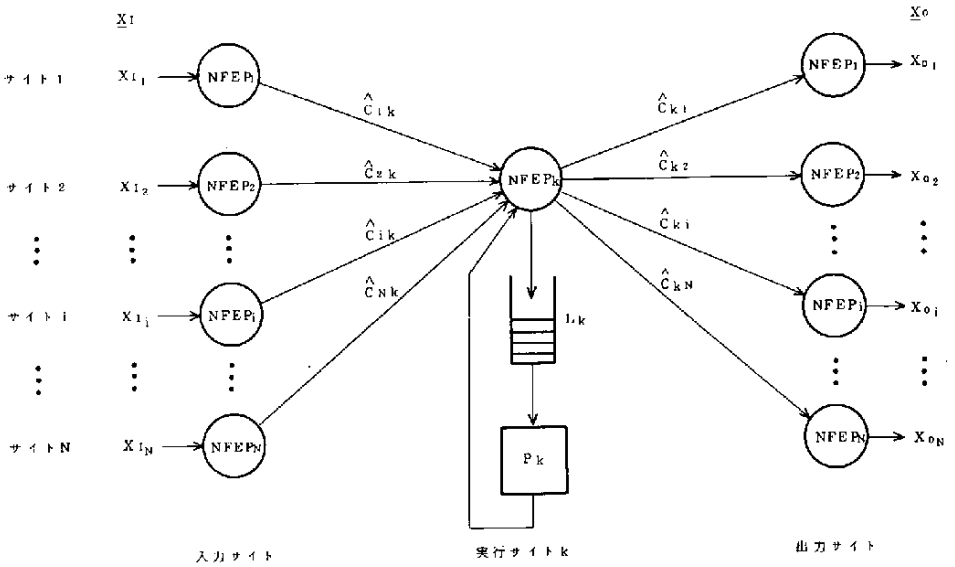


図６－１１　ロード・レベリング・モデル

②　実行サイト・モデル

・処理能力：P_k（ＭＩ／秒）

・ジョブ実行待行列長：$\hat{L}_k$（ＭＩ）

③　伝送システム・モデル

・伝送容量：$\hat{C}_{ij}$（ビット／秒）

ここで、$\hat{C}_{ij}$はサイトｉからサイトｊまでの推定伝送容量であり、$\hat{C}_{ii}=\infty$と仮定する（実用上は、$\hat{C}_{ij}=C$（一定）（$i \neq j$）としても

よい）。

サイトkでジョブを実行した場合の推定ターン・アラウンド時間$\hat{T}_k$は、次式より求まる。

$$\hat{T}_k = \hat{T}_I + \hat{T}_P + \hat{T}_O \qquad (1)$$

ここで、$\hat{T}_I$ は入力データの実行サイトへの総伝送時間、$\hat{T}_P$ は実行サイトでの実行待時間と実行時間の和、そして$\hat{T}_O$ は出力データの出力サイトへの総伝送時間である。

$$\hat{T}_I = \sum_{i=1}^{N} \frac{X_{I_i}}{\hat{C}_{ik}} \qquad (2)$$

$$\hat{T}_P = \frac{\hat{L}_k + \hat{X}_P}{P_k} \qquad (3)$$

（$\hat{L}_k$：実行待時間、$\hat{X}_P$：実行時間）

$$\hat{T}_O = \sum_{i=1}^{N} \frac{X_{O_i}}{\hat{C}_{ki}} \qquad (4)$$

$$\hat{T}_{min} = \min \{ \hat{T}_k \mid k = 1, 2, \cdots\cdots, N \} \qquad (5)$$

この$\hat{T}_{min}$になるサイトが最適サイトである。

一般に、ターン・アラウンド時間を短くするためには、入力データが大量の場合、即ち入力データの伝送時間が長い場合には、入力データのあるサイトへプログラムを転送して実行し、出力データの伝送時間が長い場合には、出力サイトへプログラムを転送して実行する。

ＮＦＥＳは、プログラム実行文のＡＴ句にBESTSITEが指定された場合、そのプログラムに対する｛$\hat{X}_P$，$\underline{X}_I$，$\underline{X}_O$｝の推定値と自サイトの$\hat{L}_k$とをＮＣＩＣへ送り、最適サイトを紹介してもらう。

この要求を受けたＮＣＩＣは、ソース・サイト以外のすべての適切な（互換性のある、あるいは変換可能な）サイトのＮＦＥＳへＲＦＳ

(Request For Service)コマンドをブロードキャストする。

このコマンドを受信したすべてのＮＦＥＳは、自サイトの$\hat{L}_k$ を返信する。ＮＣＩＣは、すべてのサイトの$\hat{L}_k$ を収集後、各サイト毎に$\hat{T}_k$ を計算し、$\hat{T}_k$ が最小である最適サイトを見つけ、要求元のＮＦＥＳに報告する。

ＮＦＥＳは、最適な実行サイトにジョブと入力データを転送して、実行し、実行完了後、出力データを回収する。

6.5.5 データ・マイグレーション

データ・マイグレーション(Data Migration)は、あるサイトで実行しているプログラムが別のサイトにあるデータ・ファイルをアクセスできるようにする基本的な機能である。

ＤＡＴＡ文で作成されたデータ・カードは、プログラムの実行直前に実行サイトに転送され、プログラムからローカルなアクセスを可能にする。

ＦＩＬＥ文のＷＨＥＲＥ節中でＭＯＶＥ＝ＹＥＳと指定されたファイルは、データ・カードと同様にファイル全体を実行サイトに転送する。

ＭＯＶＥ＝ＮＯあるいはＭＯＶＥ属性式が省略されたファイルは、プログラムの実行直前には実行サイトに転送されず、プログラムからのデータアクセスの要求が発せられる毎に、ＮＤＡＭ(Network Data Access Method)を介してデータ操作が行なわれる。

転送データは、ソース側でローカル表現形式からネットワーク上での標準表現形式ＥＳＤＩＦ(Encoded Standard Data Interchange Form)に変換され、ターゲット側でＥＳＤＩＦからそのサイトのローカル表現形式に変換される。但し、ファイルには、そのファイルに関するデータ構造やデータ属性を述べているディスクリプタ(Descriptor)あるいはスキーマ(Schema)が添付されていると仮定する。

第6章の参考文献

(1) Elovitz, H.S. and Heitmeyer, C.L.
What is a Computer Network ?, National Telecommunications Conference, Dec. 1974, pp. 1007—1014

(2) Kimbleton, S.R. and Schneider, G.M.
Computer Communication Networks : Approaches, Objectives, and Performance Considerations, Computing Surveys, Vol. 7, No. 3, Sept. 1975, pp. 129—173

(3) Whitby-Strevens, C.
Current Research in Computer Networks, ACM Computer Communication Review, Vol. 6, No. 2, Apr. 1976, pp. 13—40

(4) Kimbleton, S.R. and Mandell, R.L.
A Perspective on Network Operating Systems, Proc. AFIPS NCC, June 1976, pp. 551—559

(5) Rosenthal, R.
Network Access Techniques—A Review, Proc. AFIPS, NCC, June 1976, pp. 495—500

(6) Benoit, J.W. and Graf—Webster, E.
Evolution of Network User Services—the Network Resource Manager, Proceedings of the 1974 Symposium on Computer Networks : Trends and Application, May 1974, pp. 21—24

(7) Rosenthal, R.
Accessing Online Network Resources with a Network Access Machine, IEEE Intercon Conference Record, Apr. 1975, pp. 25/3 : 1—4

(8) Marcus, R.S.
Network Access for the Information Retrieval Application, IEEE Intercon Conference Record, Apr. 1975, pp. 25/4:1-7

(9) Bouknigt, W.J., Grossman, G.R., and Grothe, D.M.
The ARPA Network Terminal System-A New Approach to Network Access, Proceedings DATACOMM Nov. 1973, pp. 73-79

(10) Retz, D.L.
ELF-A System for Network Access, IEEE Intercon Conference Record, Apr. 1975, pp. 25/2:1-5

(11) Farber, D.J. and Larson, K.C.
The System Architecture of the Distributed Computer System-The Communications System, Proc. Symposium on Computer-Communicatins Networks and Teletraffic, Apr. 1972, pp. 21-27

(12) Farber, D.J. and Larson, K.C.
The Structure of the Distributed Computing System -Software, Proc. Symposium on Computer-Communications Networks and Teletraffic, Apr. 1972, pp. 539-545

(13) Farber, D.J. and Heinrich, F.R.
The Structure of a Distributed Comptuer System-The Distributed File System, The First International Conference on Computer Communication : Impacts and Implications, Oct. 1972, pp. 364-370

(14) Farber, D.J., Feldman, J., Heinrich, F.R., Hopwood, M.D., Larson, K.C., Loomis, D.C., and Rowe, L.A. The Distributed Computing System, Proceedings of Seventh Annual IEEE Computer Society International Conference, Feb. 1973, pp. 31—34

(15) Thomas, R.H. A Resource Sharing Exective for the ARPANET, Proc. AFIPS NCC, June 1973, pp. 155—163

(16) BBN Distributed Computation and Tenex—Related Activities, Quarterly Progress Report No.1, 1 Nov. 1974 to 31 Jan. 1975, BBN Report No. 3012, Feb. 1975, pp. 36

(17) OSCLTG(presented by E.H. Sibly) The Operating Systems Command Language Task Group: Technical Report Dec. 1973, Proc. of the IFIP Working Conference on Command Language, July 1974, pp. 353—388

(18) Sibly, E.H. Economic Justification of an OSCL/OSRL, ACM Operating System Review, Vol. 10, No. 4, Oct. 1976, pp. 7—15

(19) Sayani, H.H. A Basis for the Design of Operating System Command and Response Languages, Proc. of the ACM Annual Conference, Oct. 1976, pp. 373—380

(20) Cowan, R.M.
Burroughs B6700/B7700 Work Flow Language, Proc. of the IFIP Working Conference on Command Language, July 1974, pp. 153—166

(21) Burroughs
B6700/B7700 Work Flow Management Reference Manunal, Form No.5001175, Apr. 1976

(22) Somia, M.
The Approach of Software Problems in the SOC Experimental Computer Network, The First International Conference on Computer Communication : Impacts and Implications, Oct. 1972, pp. 390—396

(23) du Masle, J.
An Evaluation of the LE/1 Network Command Language Designed for The SOC Network, Proc. of the IFIP Working Conference on Command Language, July 1974, pp. 319—326

(24) Raymond, J. and du Masle, J.
NJCL, A Network Job Control Language, IFIP Information Processing, Aug. 1974, pp. 301—303

(25) Chupin, J.C.
Command Languages and Heterogeneous Networks, Proc. of the IFIP Working Conference on Command Language, July 1974, pp. 311—318

(26) Chupin, J.C.
Control Concepts of a Logical Network Machine for Data Banks, IFIP Information Processing, Aug.

1974, pp. 291－295

(27) Chupin, J.C., Seguin, J., and Sergeant, G.
Distributed Applications on Heterogeneous Networks, Paper Submitted to the IEEE/ACM Third Annual Symposium on Computer Architecture, Jan, 1976, pp. 20

(28) Lagasse, J.P., Artaud, G, and Cabanel, J.P.
ARAMIS－A Processing Network with User Data Bases Interactive Systems, Proc. IEEE COMPCON, Sept. 1975, pp. 213－216

(29) Schicker, P., Baechi, W., and Duenki, A.
Job Control in a Heterogeneous Computer Network, Proc. European Computing Conference on Communications Networks, Sept. 1975, pp. 537－547

(30) Schicker, P. and Duenki, A.
Network Job Control and its Supporting Services, Proc. Third International Conference on Computer Communication, Aug. 1976, pp. 303－307; also INWG Protocol 33

(31) Huckle, B.A.
Creating a Uniform User Interface to a Heterogeneous Computer Network, INWG General 129, Sept. 1976, pp. 10

(32) Rayner, D.
Network Job Control in the NPL Context, INWG General 130, July 1976, pp. 7

(33) Gerla, M.
Deterministic and Adaptive Routing Policies in Packet-Switched Computer Networks, Proceedings DATACOMM, Nov. 1973, pp. 23-28

(34) Roome, W.D. and Torng, H.C.
Modeling and Design of Computer Networks with Distributed Computation Facilites, Proceedings of the 1974 Symposium on Computer Networks: Trends and Applications, May 1974, pp. 30-38

(35) McGregor, P.V. and Boorstyn, R.R.
Optimal Load Sharing in a Computer Network, IEEE International Conference on Communications, June 1975, pp. 41-14-41-19

(36) Howell, R.H.
The Integrated Computer Network System, The First International Conference on Computer Communication: Impacts and Implications, Oct. 1972, pp. 214-219

(37) Cerf, V.G., Harslem, E.F., Heafner, J.F., Metcalfee, R.M., and White, J.E.
An Experimental Service for Adaptable Data Reconfiguration, IEEE Transaction on Communications, Vol. COM-20, No. 3, June 1972, pp. 557-564

(38) Schneider, G.M.
DSCL-A Data Specification and Conversion Language for Networks, Proc. ACM SIGMOD Conf. May 1975, pp. 139-148

(39) Birss, E.W. and Fry, J.P.
Generalized Software for Translating Data, Proc. AFIPS NCC, June 1976, pp. 889-897

(40) White, J.E.
A High-level Framework for Network-based Resource Sharing, Proc. AFIPS NCC, June 1976, pp. 561-570

7．分散型データ・ベース

7．分散型データ・ベース

7.1 はじめに

最初に開発されたデータベース管理システム（ＤＢＭＳ）とみなされているのは、ＧＥ社が１９６３年に開発したIntegrated Data Store(IDS)である。このＤＢＭＳはデータ構造としてネットワークをサポートし、ダイレクト・アクセス装置に蓄積される情報の構成、格納、検索、更新および削除ができるシステムであり、主にホスト言語をもちいてとりあつかうように作られていた。

ＩＤＳ以降、６０年代から７０年にかけて種々のＤＢＭＳが開発され、利用されている。

これらのシステム開発、導入の要因としては、データあるいはファイルの重複による無駄を避けるということであった。すなわち、各業務ごとにファイルを作成し処理するよりは、ファイルを集中化、共有化することによって重複投資をさけることにあった。

このようなファイルの共有化を達成するためのＤＢＭＳは、次のような機能を持つ必要がある。

(1) データのアプリケーションプログラムからの独立

複数のアプリケーションプログラムで共有されるために、でき得れば、すべてのアプリケーションにとって最適なデータ構造を持つ（データ構造を持たない場合も含めて）ことが必要となる。

(2) データの保全性（Data Integrity）

正しいデータが高い信頼性で利用できなければならない。これは、関連するすべてのアプリケーションにおいて共有されているデータであることから、データが乱された場合の影響が非常に大きいことによる。

(3) データの機密保護（Data Security）

多数のアプリケーション、利用者がデータベースを共有しているために、故意、不注意、無意識など目的、方法を問わず、正当でない利用者が、データへの侵害、および破壊をおこない得ることになる。これを適当な手段を講じて守る必要がある。

現在までに開発されたこれらの機能を持ったＤＢＭＳは非常に多数にのぼっているが、特に注目しなければならないのは、CODASYLのData Base Task Group（DBTG）の仕様および、ＩＢＭのF.E. Coddのリレーショナルモデルである。

前者は「データベース作業班からCODASYLプログラム言語委員会への報告１９７１年４月」および「CODASYLデータ記述言語の開発報告１９７３年６月」の２つであってデータベース界にあたえた影響は大きい。

後者は、データベースを厳密に扱う上での理論的基盤を求める努力の中から生れてきたものであって、高度のデータ独立性を保証するアプローチとして重要視されている。

このように、１９６３年に最初のＤＢＭＳが開発されて以来、約１０年にして理論的位置づけ、共通化のための活動の成果が報告され、１５年経過した今日では、大きな勢力を持ったＤＢＭＳを中心とした戦国時代にあるといってよい。この時代は同時に、普及期でもある。例えば、ソフトウェアパッケージの分野では、データベース管理パッケージの売上げは、ファイル管理パッケージに次いで第２位にランクされており、どのソフトウェア・ハウスもデータ・ベース管理パッケージで一様に成功を収めていると報告されている。

データ・ベース技術とは独立に、１９６８年よりパケット交換技術の基礎に上にたって異機種分散型のコンピュータ・ネットワークの開発が開始された。

このコンピュータ・ネットワークの最終目標は、いかなるリソースもローカルな利用者が使うのと同じように、遠隔地から利用されることにある。

このようにして共有されるリソースはハードウェアはもちろんのことソフトウェアやデータも含まれる。それまでタイム・シェアリング・システムによって局所的には、ソフトウェアの共有が可能になっていた。コンピュータ・ネットワークは、タイム・シェアリング・システムにおける局所的な制約を取り外し、規模および距離の面での飛躍的な拡大をはたすことになった。

コンピュータ・ネットワークの初期のものとしては、ＡＲＰＡネットワーク、ＮＰＬネットワークがあるが、特に前者の功積が大きい。

ＡＲＰＡネットワークは、１９６９年末に米国西海岸のＳＲＩ、ＵＣＳＢ、ＵＣＬＡ、ＵＴＡＨの４ケ所のノードおよびホストからはじまって、５０を超えるノードと１００台を超えるホストからなるシステムに発展した。そのカバーする範囲も、ハワイ、アメリカ本土、ノルウェー、英国となり、衛星通信まで利用している。

ＡＲＰＡネットワークに対する批判は数多くあるが、最初にして最大のシステムが出来上ったこと、このコンピュータ・ネットワークが以降にでてくるネットワークにあたえた影響という意味では大成功であったということができる。

ＡＲＰＡネットワークは、以降のネットワークにあたえた影響としては、次のものがあげられる。

(1) コンピュータ通信の手段としてのパケット交換網の実現

(2) ミニコンをノードとした高速、高信頼性の通信手段の実現

(3) プロセス間通信によるリソース共有の一般的アプローチ

(4) 階層構造を持ったプロトコルによるネットワーク作成者側からの標準手法の実現

(5) メッセージのラウンド・トリップ時間などの一般的な標準値の設定

次の世代にでてきたネットワークは、ＡＲＰＡネットワークの実験成果を生かして開発された。このコンピュータ・ネットワークを代表するのが、フランスのCycladesである。

Cyclades は、１９７２年初頭より情報代表部（Délégation a Informatique：DI）の指導のもとに開始された。これの目的は、種々の新らしい分野での実験道具となるプロトタイプのコンピュータ・ネットワークになることであり、特に次の６つの分野が考えられた。

(1) データ・コミュニケーション

(2) コンピュータの相互会話

(3) 協同ジョブの研究

(4) 分散型データ・ベース

(5) インタフェースおよび共通言語

(6) リソースの共有

Cyclades の成果としては、上記の如くアプリケーションの実験が表面にあらわれたこと、パケット交換網とホストの役割を明確にしたこと、プロトコルの重要性を再認識し国際標準化への動きに対して大きな役割をはたしたことをあげることができる。

Cyclades 以降、同様なネットワークが次々と作られて、実験、限定的な実用化がなされた。

Cyclades の次の世代を考えるとき、コンピュータ・ネットワークには２つの分野での異ったグループの動きを見ることができる。１つは、コンピュータ・サイドに立ったグループであり、もう一つは、コミュニケーション・サイドに立ったグループである。

ＡＲＰＡネットワーク、Cyclades ほか種々のコンピュータ・ネットワークの成果、独自の研究開発の成果をもとに、それぞれのコンピュータ・メーカーがネットワーク・アーキテクチャの作成、確立という方向に進んできている。

これの口火になったのは、いうまでもなくＩＢＭのＳＮＡである。日本においても国産各社のネットワーク・アーキテクチャが構想の段階とはいえ、発表されつつある。

コミュニケーション・サイドの動きとしては、ARPAネットワークの提案者である、L.G. Robertsが社長となって作ったTELENETによるパケットを交換サービス、英国郵政省のEPSS、フランスのTranspac、カナダのDatapac、日本電信電話公社の新データ網など、商用のパケット交換網サービスの動向をあげることができる。これらの一部はすでにサービスを開始している。

このように、データ・ベースとコンピュータ・ネットワーク（分散処理）が同じような時期に、発展期をむかえたことになった。そして、次のステップとして、データ・ベースと分散処理とを一体にした形での利用が考えられるようになった。これがそもそもの分散型データ・ベースの発端であるということができる。

7.2 分散型データ・ベースの定義

分散型データ・ベースはコンピュータ・ネットワークを基礎とする分散処理技術と、データ・ベース技術が一体となったものであるということができる。このような立場から分散型データ・ベースを定義するとすれば"ネットワーク内に散在するデータを、複数のアプリケーションが存在場所を意識することなく、共有できるシステム"ということができるだろう。

しかしながら分散型データ・ベースは現在のところでは、概念的にとらえられているだけで、共通の定義があるわけではない。ここでその定義のいくつかを紹介する。

(1) Boothの定義

広い意味でいえば、種々の場所でデータが蓄積されているとき、分散形データ・ベースを構成すると考えることができる。しかしながら実際には各地のデータ要素が相互に関係しているか、ある所で実行されているプロセスが他で蓄積されているデータにアクセスする必要があるときに、分散型データ・ベースが存在すると考えられる。

分散型データ・ベースには、あるデータのセットのコピーを複数個のサイトで持つ形式と、各地のデータが一つのセットを構成する場合がある。

(2) Farber の定義

ファイルとは、一つのサイトにおけるデータの物理的な集合である。データベースは、データ処理において使われるデータの論理的な集団である。データ・ベースを構成するファイル群が異なった場所にあるとき、分散型データ・ベースという。

(3) Ramamoorthy and Krishnarao

分散型データ・ベースは、種々の処理装置にあるデータ・ベースを論理的に統合したものである。

(4) Deppe & Fry の定義

分散型データ・ベースとは、統合的なアクセスが可能となっているコンピュータ群に、物理的なデータ・ベースを分割したものである。

(5) Chupin の定義

データ・ベースは、利用者、データ・ベース管理システム、データ・ベースより構成される。分散型データ・ベースとは、一つ以上のアクセス法が分散されたデータ・ベースである。

以上に例示したように、分散型データ・ベースに関する限り、確定した定義はない。ここでは、非常に広い意味で分散型データ・ベースというものをとらえて、そのニーズ、技術的背景、動向をとらえてゆくことにする。

この項のはじめにおこなった定義は、将来の目標としての分散型データ・ベースであって、現状にこれをあてはめて考えてみると、ほとんど例をみつけられない。この意味もあって、どちらかというと分散型データ管理システムと呼ぶにふさわしいものまで含めて調査してゆく。分散型データ管理システムとは、以下に示すようなものである。

データ管理システムは、図7－1のように次の3つのセットから構成される。

(1) 利用者（ターミナル、利用者プログラムなど）

(2) データ管理プログラム

(3) 物理的なデータ

ここでデータ管理プログラムとは、利用者がデータをアクセスする場合に利用できるプログラムすべてを言うこととする。

データ管理システムが、複数個の独立なコンピュータにある物理的なデータをアクセスできるとき、このようなシステムを分散型データ管理システムと呼ぶ。

分散型データ管理システムは図７－２に示すような構成となる。

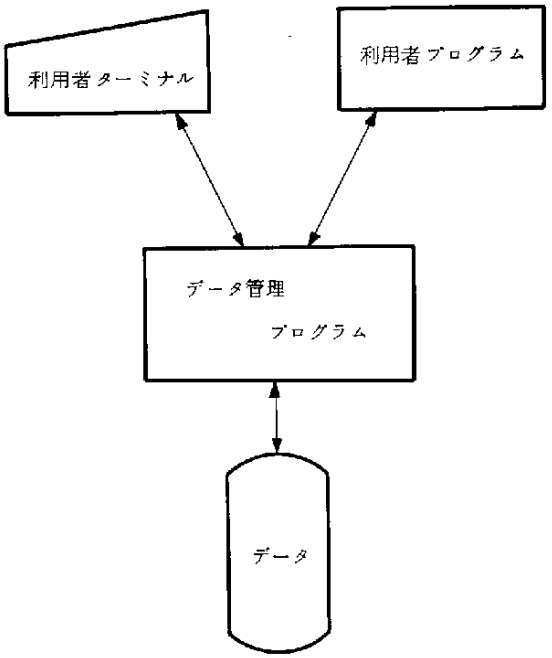


図７－１　データ管理システムのモデル

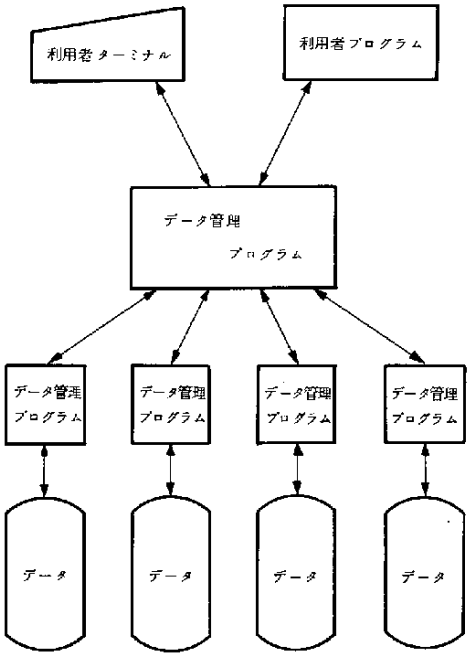


図7－2　分散型データ管理システムのモデル

7.3　分散型データ・ベースの背景

7.3.1　利用者のニーズ

分散型データ・ベースに対するニーズは、異った組織あるいは異ったコンピュータを利用しているグループ間でデータ交換をおこなっている場合には、潜在的に分散型データ・ベースのニーズが存在すると見るべきである。現在、利用者の責任あるいは手間によって、複数のデータ・ベースを個々に利用しているところにもニーズがあると考えてよいであろう。

データを共有するグループのメンバー相互の関係としては、同一企業内、系列企業間、官庁間あるいは国際的なレベルなどがある。

それぞれの範ちゅうでどのような実例あるいは計画、構想があるかをみてみよう。

A　同一企業内

単に同一企業内といってもその規模はさまざまであり、比較的小規模のものとしては、研究室あるいはグループ内で同一のファイルを使用するものから複数の研究室あるいは、企業で言えば部門間にまたがった程度のものがあろう。

このレベルの例としては、電子技術総合研究所でおこなっているパターン情報処理システムEPICS(Electro technical Laboratory's Pattern Information Computing System)におけるオンライン・パーマネント・ファイルをあげることができる。このシステムの基本思想は、中央に主計算機(Tosbac 5600/170)をおき、チャネル経由で副計算機(FACOM 230/35、HITAC 8350、Tosbac 3400/41、PDP−11、NEAC 3200/50)と結合し、主計算機はコンピューティング能力および大容量ファイル、副計算機では種々の特殊装置および局所処理を担当させるものである。

企業全体にわたる業務において、データの共有をおこなっている例としては、住友銀行におけるオンライン・バンキング・システムなどがある。

全国規模にわたる同一企業内のネットワークとしては、旭化成のACTをあげることができる。

ＡＣＴの背景と目的は次のようなものである。

旭化成は、札幌から宮崎にいたるまで2本社、１４事務所、7研究所、３１工場を持ち、全事業部が全国にちらばっており、これらの事業所が多種多様な商品をとりあつかっている。これらの事務所における情報処理は自然発生的に生成してきた。その結果として、多種多様の汎用および制御コンピュータが各地に導入されてきた。

また社内情報の伝送のために多数のテレタイプを使用していたが、中継など途中で人が介在するために能率が良くなかった。

このような状況の下で、ＡＣＴは2つの目的を持って開発されること

になった。

第一番目は、全国規模の業務を東京、大阪の本社コンピュータに集中し、生産関係の業務は工場が属する地区の汎用コンピュータに集中することである。すなわち、ＡＣＴは、全国に散在する汎用コンピュータ相互間の通信路および同一地区内の汎用コンピュータと制御用コンピュータまたはミニ・コンピュータ相互間の通信路となってファイル転送、データのコード変換をする役割をはたすことである。

第二番目は、テレタイプ網の中継の自動化をはかり、省力化および業務の質の向上をめざすものである。

B　同種企業間

同一企業内での情報処理のコンピュータ化がうまくいったときに、より高度なサービスあるいは迅速、的確な情報交換のためにとり上げられたのが同種企業間のデータ共有である。比較的早い時期に作られたものとしては、地銀協の為替交換システムがある。また近畿日本ツーリスト・オンライン・ネットワークおよび日本キャッシュサービスのシステムもこれに該当するものである。

前者のシステムは、日本航空（ＪＡＬ）のＩＢＭ360/65、全日本空輸（ＡＮＡ）のＨＩＴＡＣ8500、東亜国内航空（ＴＤＡ）のＮＥＡＣ2200−500、近畿日本鉄道のＵＮＩＶＡＣ418Ⅱの４台のコンピュータがそれぞれおこなっている座席予約サービスを近畿日本ツーリストの端末から利用しようとするものである。

後者のシステムは、キャッシュ・ディスペンサーの共通利用を目的とするものである。すなわち都市銀行１３行、地方銀行１１行、相互銀行１２行の各コンピュータ・センターと端末とをオンラインで結合し、参加銀行のＣＤカードのいずれであっても、その銀行のコンピュータ・センターの預金ファイルをチェックすることによって、現金の支払いをするものである。

同種企業間でのデータの共有は、サービス・レベルでは非常に進んでいるが、技術レベルでは端末装置レベルの共有をおこなっているだけである。技術レベルとしては低いところにおさえてある理由としては、現状のサービス・システムを変更しないで最も簡単におこなう方法としては端末装置によるデータの共有が最適であることによる。

C　サービス会社と利用者

種々の利用者がデータの共有をおこなう場合に、直接利用者間でおこなうこともあるが、特定のサービス会社を経由してたがいにデータを共有する方法もある。

同種企業間でのデータの共有の例としてとりあげた、地銀協システムなどはこの例である。これ以外にも、ＴＳＳベンダーを利用してのデータの共有が考えられる。

ここでは米国のＴＳＳサービス会社であるTYMNETにおける利用例をみてみよう。この例の中で最も興味深いのは、医療データ・バンクをＴＳＳによって利用できることである。

TYMNETは、TYMSHAREという名のタイムシェアリング会社が運用サービスをしている分散形コンピュータ・ネットワークである。

TYMNETは2つの目的を持っている。

(1)　TYMSHARE の商用タイムシェアリング・サービスに遠隔地からのアクセスをサポートする。

(2)　ユーザが自分自身のコンピュータに端末からアクセスできるようにする。

このネットワークは、７０都市に９０台のノードを持っており、その一部はパリ、ブリュッセル、ロンドンまでサービスされている。このネットワークの特徴は、７つのメーカの１５種類のコンピュータを設置しており、利用者はこのうちのどれでも使用することができる。

TYMNETは科学技術計算、オーダ・エントリ、情報検索のために利

用できる。

情報検索についていえば、LockheedのDIALOG、System Development CorporationのORBIT、National Library of MedecineのMEDLINE、Battelle Memorial InstituteのBASISという米国における主要なシステムがTYMENETに結合されている。

これらの文献検索システムは独立な利用体系によって使用されているが、全体を一つのデータ・ベースとして利用するとか、同一の利用体系によって使用するとかの要望がでてくることも考えられる。

D　大学・研究所間

昭和48年度より3年間継続の特定研究として8億5,000万円の研究費と毎年20をこえる研究班、3年目には219名の教授、助教授、研究者が協同して進められたプロジェクトとして、「広域・大量情報の高次処理」があった。この研究の主旨は次のとうりである。

研究・調査等における学術情報、すなわちデータならびに文献情報が急速に大量、広般になってきている。専門または地域において広域の多数の利用者によるこれら学術情報の効果的な利用のためには、効率的な高次情報処理を行なう巨大情報システムの研究開発が必要である。すなわち、学術情報のシステム化、情報資源共有ネットワーク（resource sharing network）の形成および高水準の構造解析にもとづく高度の情報処理システムの研究が要求されている。これらは、学界の緊急な課題であり、かつ、情報化社会の礎石となるべきものである。ここでは、そのための基礎理論的、および実験的（システム設計を含む）研究を対象とし、主として次の研究項目例にかかわるものに重点をおく。

（研究項目例）

① 巨大情報システムの体系化基礎論

② データおよびデータベースの構造解析

③ 巨大情報システムにおける言語問題

④　巨大情報システムにおける情報ネットワーク

⑤　学術情報のシステム化

データ・バンクの共同利用を、コンピュータ・ネットワークを通じて行なうことは、3年間の特定研究の仕事としては、実現しえなかった。しかし、このための準備をしたこと、研究者として科学技術情報の共有の重要性についての認識があることから、この種のデータ・ベースの共有は今後の大きなニーズとしてとらえることができる。

E　官庁間

行政官庁では、それぞれ個別に情報処理機能を持っており、主として個別業務あるいは統計集計といった作業を行なっていたが、官庁相互のタテ割り意識が、お互の情報流通を妨げてきたため、政策立案に必要な情報を迅速的確に他機関から入手することは困難であった。このため、同じような調査・集計を複数の省庁で行なうなどの無駄も多かった。

このような背景はあるにしろ、省庁間におけるデータの共有（データの交換）は、表7－1に示すような状況で存在している。

官庁間の個別の情報処理機能は、行政サービスの簡素化と質の向上という面、政策情報の必要性という2つの面から、複合化という構想が生じてきた。

これらの構想は、各省庁において異なったものになっているが、大別すると4つの形になると考えられる。

第一番目は、行政管理庁の提案している行政情報通信ネットワーク（Administrative Information Communication Network：AICON）に代表されるものである。これは、データ交換網の共有化を最初におこなっていくものである。データ交換網の共有化は、通信費用の軽減をはかるばかりでなく、同一の網に各省庁のオンライン網を結合することとなり、コンピュータの共同利用、データの共有などを推進する大きな力となるであろう。

表7－1　官公庁でのデータの交換例

データ使用官庁	業務名	データ名	データの入手先	オンライン方式による	オフライン方式による			
					磁気テープ	紙テープ	カード	その他
警察庁	自動車登録番号照会	自動車登録ファイル	運輸省自動車局	○（予定）				
経済企画庁	経済分析業務	交通モデル	運輸省					○
〃	〃	産業連関データ	通商産業省					○
北海道開発庁		道路関係データ	建設省、北海道庁		○			
科学技術防災センター	地震解析業務	強震計データ	強震計設置場所					○
〃〃 航空宇宙研究所	機体実験業務	機体関係データ	東京大学金属材料技術研究所					
〃放射医学総合研究所	がん治療業務	病歴ファイル	国立ガンセンター				○	
法務省	日本人出帰国管理業務	旅券発給データ	外務省		○			
外務省	貿易統計検索	大蔵省通関統計	大蔵省関税局		○			
大蔵省	貿易統計業務	通産省貿易統計	通商産業省		○			
社会保険庁	社会保険料徴収業務	被験者資格記録データ	東京都社会保険事務所			○		
通商産業省	通商白書作成	通関統計	大蔵省		○			
〃	情報蓄積加工サービス	卸売物価指数	日本銀行		○			
〃	〃	米国輸入、海外市場	JETRO		○			
〃	〃	国連統計、OECD統計	アジア経済研究所		○			
特許庁	特許情報検索	検索用蓄積データ	外国特許局		○		○	
文部省緯度観測所	星座の編集	星の位置	東大東京天文台 海上保安庁		○			
運輸省情報管理部	内航海運輸送統計	内航海運輸送データ	総理府統計局		○			
〃	観光出入国統計	観光出入国データ	〃		○			
〃 東京航空管制部	航空管制業務	気象観測データ	気象庁					○
気象庁	気象予報業務	気象データ	外国気象機関	○				
海上保安庁	天体位置計算	星のカタログ	米国天文台		○			
〃	水路編暦	海洋観測データ	気象庁、水産庁					○
〃	天体観測	天体観測データ	天文台、国土地理院					○
郵政省	電波監理業務	国際周波数登録データ	国連国際電気通信連合		○			
建設省	工業用水解析	工業統計	通商産業省		○			
建設省国土地理院	天文測量計算	星表	アメリカスミソニアン天文台		○			
〃	地形補正	地形データ	東大地震研究所				○	
〃 建築研究所	構造物の地震応答	各地強震記録	防災センター				○	
自治省	交付税計算	国調データ	総理府統計局					○
〃	〃	道路河川台帳	建設省（予定）					○

第二番目は、総理府統計局などが提案しているデータ・バンク構想である。行政官庁では、各種統計データを持っており、これらを各官庁間で総合的に利用しようとする手段をあたえることである。この構想は、各省庁などがそれぞれ保有するデータを個別のデータ・バンクという形でとらえ、これをネットワーク化することによって、総合的なデータ・バンク化を目ざすものである。このために、個別情報の総合化の手順として分散形ネットワーク、ネットワークにおけるデータ・バンク情報センター（クリアリング・センター）という方向を持つものである。

第三番目は、既にデータ交換をおこなっている省庁間で、ネットワーク化をはかってゆくものである。これらの動きとしては、通産省と経済企画庁の間などにみられる。

第四番目として、工業技術院内にある「電子計算機利用に関する技術研究会」がおこなっているものであり、５０年度より、リソースシェアリング研究班を発足させ、省庁間におけるコンピュータ・ネットワークの形成に意欲的に取り組んでいる。

F　国際的なレベル

国際的なレベルでのデータの共有としては、気象観測データの交換などがある。

現在では、気温、湿度、雨量、風向・風速などは、地上気象観測装置と称するパネルに記録部・指示部が収納され、観測机の傍で読み取りや監視が容易にできるようになっている。

これらの観測値はADESS（Automatic Data Editing and Switching System）と称する気象資料自動編集中継装置により全国的に集配信されるとともに、国際的にも高速回線で各国に配信されている。

すなわち、ADESSは、世界気象データ交換網における主幹線の一中枢としてワシントン・メルボルン・北京等と結ばれ、オンライン・リアルタイムの２４時間年中無休の運用となる。

その取り扱いデータ量はＦＡＸ以外で受信6,380,000桁／日、送信25,480,000桁／日に及び、なお増加中である。

一方、それら観測値は日表、旬表、月表、年表等として大型電子計算機により各種統計がなされ、印刷物として公表される外、その資料はＭＴとしても保存され、データ・バンク的機能をもち、一般の利用に供せられる。

もう一つの例として、ヨーロッパにおいて、科学技術に関する情報ネットワークを作ろうとしているEURONETをあげておく。

EURONETは１９７１年に、ヨーロッパの閣僚会議において、科学技術情報やドキュメンテーションに関する協力についての合意がみられた。これにもとづいた検討の結果７５～７７年の間に、次の３つの分野の活動をすることが企画された。

(1) 種々の分野（農学、物理学など）におけるシステムの開発作成

(2) 情報処理のための物理的なネットワークの作成

(3) 情報工学における技術と道具の開発

EURONETは、長期間の計画であり、利用者数は１９７６年に６万、８０年に９６万、８５年に２３５万人に増加することを前提として作られている大がかりなものである。

7.3.2 技術的背景

情報は、社会・経済活動の発展とともに膨大化の一途をたどってきた。これらの情報を処理すべく、コンピュータ・システムも巨大化してきた。しかし、コンピュータ・システムの巨大化は、マルチジョブ処理、マルチタスク処理、マルチＣＰＵによる処理というソフトウェア技術にささえられて効率を上げてきた。これらのソフトウェア技術の向上に対して、ＩＣ、ＬＳＩそして超ＬＳＩ技術という方向に発展しつつあるハードウェア技術の進歩は、グロッシュの法則にのっとって大型化してきたシステムに対し

て、この法則にしたがわないコンピュータの出現を可能にした。

すなわち、ミニコンピュータ、つづいてマイクロプロセッサの出現である。

ハードウェア技術の向上によって、コンピュータの処理コストは大巾にさがったのに反して、データ伝送の主役となっている回線コストは高速化、デジタル化によるコスト減はあるが、コンピュータの処理コストほどの大巾なコスト減の見通しはない。

このような環境的条件が整ったことによって、従来の処理を集中化して、コストを削減するという方向以外に、データの発生点あるいは、データを最もよく使用するところに分散して蓄積し、コストを削減する方向が新しく生まれたことになる。

このような分散処理の考え方は、前項で述べたような異った組織間におけるデータの共有のための技術的バックグランドとなっている。

データ・ベースシステム技術は着実に進歩しているが、Very Large Data Base（VLDB）においてその有効性が疑問視されている。ここでいうＶＬＤＢというのはデータ量が大きいばかりでなく複雑多様な目的に利用されることを意味している。この理由は以下のとうりである。

(1) ＶＬＤＢはデータ・ベース管理者が予想し得る以外のアプリケーションが多種出現し、データ構造の最適化が困難となる。このため、データ・ベースの利用の拡大が困難となる。

(2) ＶＬＤＢは、再生が困難なために、保全性へのリスクが大きくなる。

(3) ＶＬＤＢは同一のデータ・ベースを使用するアプリケーションの増大をきたすために、ファイル、中央処理装置の高速化を追求することになるが、これはコスト的に見合わなくなりつつある。

(4) ＶＬＤＢは、地域的に広い範囲の利用者に使用されるが、集中方式は、通信コストの増大をまねくことがある。

(5) ＶＬＤＢは、組織的に広い範囲にまたがった利用者を含むことになる

可能性が大きい。このときには集中管理をおこなうことは政治的に困難となる。

このような背景からも、将来のデータ・ベースを目ざして種々の研究開発がおこなわれており、たとえば、リレーショナルモデルのように構造を持たないデータ・ベースの研究や分散管理する方式に関する研究などがある。

このように、在来のデータ・ベースは大型化、複雑化することによって次々と問題点がでてきた。このような状況下で、一つの別なアプローチとしてでてきたのが分散型データ・ベースの考え方であって、この方法によって問題がすべて解決するわけではない。むしろ、分散処理によって新らたな問題、より複雑化した問題が発生してきたというべきである。

分散型データ・ベースの問題点

すでに述べてきたように、分散型データ・ベースの研究開発については、解決しなければならない問題点が多くある。これには、分散によって新らたに発生したものと、分散によって、問題点がよりクローズアップされたものがある。以下にこれを列挙する。

(1) データの分散方法

- データの最適配置
- データのコピーをどのくらい保持するか。複数個保持したときには、その間の一致性をどう保障するか。

(2) 名前のつけ方、参照方法

- データ・ベース名、ファイル名、データ名の存在場所をホスト名まで含めたフルネームで指定するのか。
- フルネームで指定しないとすれば、その管理方法は。
  - ネットワーク・カタログを分散保持。
  - 中央で情報を保持するホストを作る。
  - 放送機能などを利用して全ホストに問合せ。

(3) 管理、運営上の責任

- データ・ベースシステムは、運用管理上の責任体制が明確であることが前提となっているが、分散形においては、この前提をおくことが困難。このとき、このデータ・ベースは誰が責任を持つか。

(4) データ変換

- 浮動小数点数、固定小数点数、文字コードの内部表現の違いをどのようにとりあつかうか。

(5) 分散させる機能

- データ・ベース管理システム、ファイル管理システムのどの機能を分散させるのか。

  - 物理アクセス法の分散
  - 論理アクセス法の分散
  - 利用者アクセス法の分散

(6) リソースの競合

- データ・ベース、ファイル、データ項目などを排他的に専有するプロセス間でのデットロックの防止などの対処をどのようにするのか。

(7) 秘密保護

- 利用者の確認手続、方法をどのようにするか。一ケ所で集中しておこなうのか、各ホストの責任でおこなうのか。

7.4 分散型データ・ベースのとらえ方

7.4.1 機能面での分類

ここでは分散型データ・ベースはどのような観点から分類できるかについて述べてみる。

これは、次の4つの側面からとらえるのが妥当である。

(1) 制御および管理

(2) 均 質 性

(3) データの冗長性

(4) データの関係

これらの側面を述べたのち、既存のシステムおよびデータの共有に関する種々の提案を、この側面から分類する。

A 制御および管理

制御および管理とは、コンピュータ・ネットワークの種々なホストに分散しているデータ・ベースをどのように制御、管理するかということである。

分散型データ・ベースにおいては、次の制御、管理形態が考えられる。

(1) 統一管理

- 集中管理方式
- 分散管理方式

(2) 局所管理

統一集中管理方式

統一管理とは、コンピュータ・ネットワークの種々のホストに分散したデータ・ベースを全体的な見地から管理する方式をいう。統一集中管理とは、データ・ベース自体は種々のホストに分散しているが、それの制御および管理は、特定のホストにおいて集中的におこなう方式であり、図7－3の如き形態となる。

この形態では、利用者がデータにアクセスする場合には、必ず管理しているホストを経由しなければならなくなる。このためにデータ・ベースのアクセス（要求）が管理をおこなっているホストに集中してしまうことになる。それ故に機能分散によるメリットは、データ・アクセスの処理を複数のコンピュータで同時におこない得ること以外は全くないといってよい。

ホストが通信回線のようにせいぜい50～200Kbit/secの容量のもので結合されているときには、データの伝送にかかわる処理および伝

送時間がかかるデメリットがあるために、分散型データ・ベース・システムの実験としての意味しかない。

ホスト間がバス結合のように1 megabit/sec以上になれば,伝送時間等が無視できるので、複数個のホスト／CPUにおいて同時処理をおこなうことができるメリットのみが残ることになり有効である。

この意味において、統一集中管理方式の有効性は、コンピュータ・ネットワークというように自律的なコンピュータが結合したものにおいて見い出されるというよりはむしろ、コンピュータ・コンプレックスのように強い結合形態あるいはデータ・ベース・マシンのように、データ・ベース・マシンのように、データ・ベース・アクセス部分を独立させることによって同時処理のメリットを求めるものに見い出される。

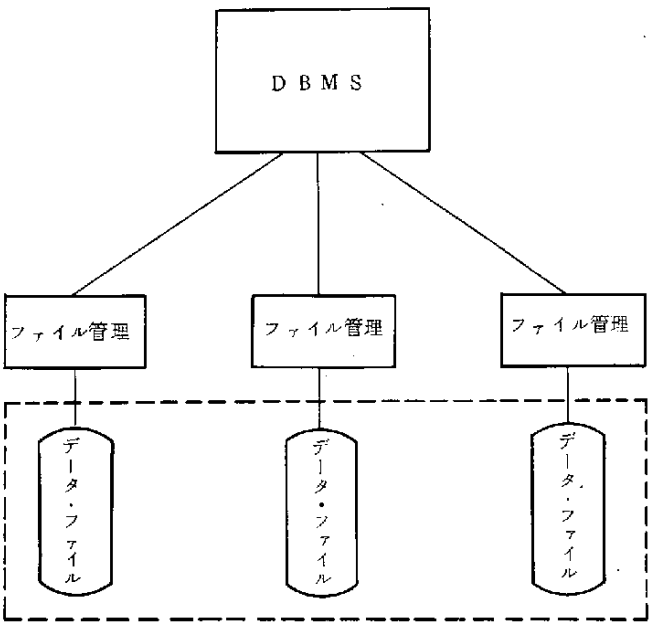


図7－3　統一集中管理方式

統一分散管理方式

この方式は、図７－４の如く、データ・ベースのあるホストにそれぞれＤＢＭＳをおき、これらが協同で統一的なデータ・ベースを管理するものである。これは統一集中管理方式における唯一のＤＢＭＳを、各ホストに分散させたものであると考えることができる。

この方式では、利用者がデータをアクセスする場合には、基本的には利用者のいるホストとデータの存在するホストだけがデータアクセスに関係するようにすることが可能なはずである。

もちろん、データの同時更新などのチエックなどを大がかりにやろうとすれば、制御情報が各ＤＢＭＳ間を往き来することになるのでいちがいにこのようなことは言えない。

しかしながら、これは分散型データ・ベース・システムの終局的な形態の一つのパターンとして種々の問題を解決し、実現のための研究、実験が進められていくであろう。

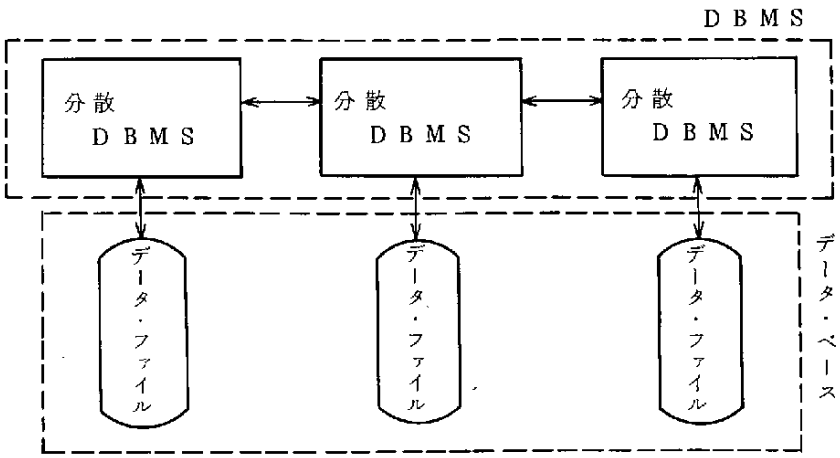


図７－４　統一分散管理方式

局所管理方式

データ・ベースは、それぞれのホストに分散されている場合に、統一管理方式においては、全体的な見地から全体を統一的に管理している。局所管理方式は、全体的な管理は一切おこなわないで、それぞれのホストにおいて自分のところにあるデータ・ベースを各自、独立して管理するものである。

元来、それぞれのホストのＤＢＭＳは、自分のデータ・ベースに関しては十分な管理をおこなっているのであるから、それぞれのデータ・ベースに関しては、有効かつ問題なく利用できる。

利用上問題になるのは、複数個のホストを利用するユーザが、それぞれのデータ・ベースをどのようにアクセスすべきかということである。

まず第一に、ユーザはどのような言語でデータ・ベースをアクセスするかということである。すなわち同じアクセス言語を持ったデータ・ベースならは、同一の言語で利用できるが、同一のアクセス言語を持たない場合に、もともとのシステムが持っている言語でアクセスをするか、それとも共通言語を決めて、それによってアクセスするかということである。

制度、機構上の問題から異ったアクセス言語を持つデータ・ベースへのアクセスは、ひんぱんに起こると考えるべきなので、利用者に負担をかけない共通言語によるアクセス方式を検討しなければならない。

第二は、異ったデータ・ベース間のデータの構造（リレーション）が表現できないために、ユーザ側でこれを解決しなければならないことになる。すなわち、データ・ベースを利用するシステムにおいては、プログラムあるいは利用者のコマンドは、データの存在場所などを意識する必要がないことがメリットであるが、このような複数個のデータ・ベースを利用する場合には、このようなメリットが半減すること

になる。

第三は、排他的にデータを利用する場合に、データが異なったデータ・ベースに存在したとすれば、デッドロックがチェックできないことである。

このように種々の問題をかかえているが、コンピュータ・ネットワークにおける各ホストの自律性とデータ・ベース管理における統一的なものとの妥協点、あるいは機構上の問題から、あるいはオーバヘッドと利用者のサービスのトレードオフなどの種々の点から考えて、現在においては、プリミティブな形で使用され、将来は、共通言語という形で最もひんぱんに利用される形態であろう。

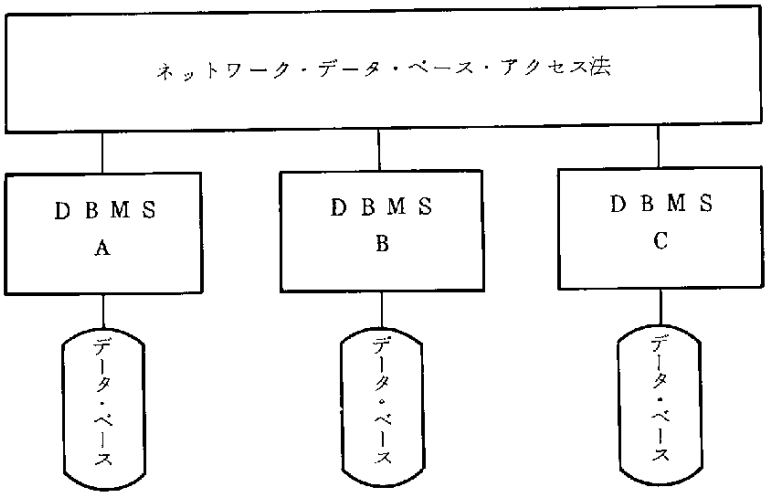


図7－5　局所管理方式

B　均　質　性

分散型データ・ベース・システムを構成する各ホストにあるDBMSが均質であるかどうかということである。ここで均質とは、2つの意味を持っている。第一はデータ構造の表現、データの操作など、ユーザに公開されている言語が同一であることを意味し、もう一つは、

ＤＢＭＳを分散させることを考えてみたとき、分散されるＤＢＭＳ間でのインターフェースあるいはプロトコルが同じであることを意味する。

本来は、両方の意味で均質であるときに、均質なデータ・ベース・システムであるということができるが、ここでは広い意味にとらえて前者が満足されていれば、均質であると考える。

コンピュータ・ネットワークのホストが同機種あるいは同じファミリーに属する場合、非常に強力なＤＢＭＳが出現し、データ・ベースに関する共通化、標準化がおこなわれた場合あるいは全く新しくデータ・ベース・システムをインプリメントする場合には均質データ・ベースであり得る。

上記以外の場合には、既存のデータ・ベース・システムを結合することになるとで不均質になると考えるのが妥当である。

C　データの冗長性

データ・ベースは、似かよったファイルを統合して、データを共通化して、重複を避け無駄を省いている。このため、データの冗長性を持たないのが原則となっている。

しかしながら、コンピュータ・ネットワークにおいては以下に述べる理由により、同一のデータあるいは同一のファイルを複数個持たせることがある。

(1)　応答性の向上

複数のサイトにおいて同一のデータを参照できることから、同じデータに対する参照の問合せがそれぞれのサイトに分散する。また非常に近い場所のサイトに問合せができることから応答性の向上が期待できる。

(2)　伝送コストの削減

データの問合せ場所から近いサイトにデータを容易に設置できる

ことから、問合せ、応答メッセージの伝送コストの削減が期待できる。

(3) 保 全 性

同一データを複数個持っていれば、一部のサイトにおいて、データが破壊されたとしても容易に復旧できる。

(4) 信 頼 性

一部のサイトの障害あるいは通信網の障害によってアクセスできないサイトがあっても、他のサイトにおいて同一のデータをアクセス可能

同一データを複数個持った場合には、どのサイトにそれを蓄積するかという問題が発生する。これに関しては、Caseyの同一ファイルのコピーをどのように配置すればトータルコストを最少にできるかという研究、MorganとLevinのプログラムの蓄積、実行場所も可変と考えたときに、データをどこに配置したらよいかという研究などがある。

コピーを持たせた場合には、応答性などの向上をさせることができるが、データがアップデートされたときに各データ項目をどのようにして一致させるかと問題が別に発生することになる。

D データの関係

データ・ベースは、データ項目間の関係を何かの形で持っている。一方ファイルは、データ項目の並びとしてレコード、ブロックなどを構成している。この意味で、データ間の関係づけが可能であるということは、ホストにおいて通常使われている意味でのデータ・ベースへのアクセスを対象としていることを示し、データ間の関係づけが不可能とは、ファイルのアクセスのみを対象としていることを示している。

これまでに述べてきた機能面での分類にしたがって既存のシステムおよびデータ共有の方法に関する提案を一覧表にしたのが表7－2である。ここで述べたシステムおよび提案は、次項で解説する。

表７－２　データ・ベースシステムおよび提案の項目別分類

システム or提案名 ＼ 項目	制御および管理	均質性	データの冗長性	データの関係	備考
Socrates	統一集中管理	均質	不明（たぶんなし）	持つ	
ARAMIS	統一分散管理	均質	なし	持つ	ミニコンネットワーク
CONIT	局所管理	不均質	なし	持つ	文献検索システムの共有
RSEXEC	統一分散管理	均質	なし	持たない	
TIPログインシステム	統一分散管理	均質	あり	持たない	TENEXにおけるTIPアカウント情報システム
Cashinの提案	局所管理	不均質	なし	持つ	関係モデルのデータ・ベースの共有の提案
Peeblesの提案	局所管理	不均質	なし	持たない	ファイル指定、アクセスコマンドの共通化の提案
du Masleの提案	局所管理	不均質	なし	持つ	ＪＣＬのプログラム言語の統一化の提案

7.4.2　分散型データ・ベースシステムの紹介

ここでは、現在までに発表されている分散型データ・ベースシステムの概要を紹介する。

A　SOCRATES

もともとSocratesは、ＣＩＩのコンピュータのデータ・ベースシステムとして開発されたものである。このシステムは、トリー構造の関係を表現でき、それらのデータを操作することができる。

このデータ・ベースに対するアクセス言語としては、会話形で利用する特別な言語がある、またCOBOLによってプログラムされたものからのアクセス方法もサポートされている。

Socratesは、Cyclades コンピュータ・ネットワークの一環として、ＣＩＩのほかにSIEMENSやＩＢＭコンピュータに対してもインプリメントされた。また、これらの複数のコンピュータ間に分散されたデータ・ベースへのアクセスを可能にするネットワークアクセス法もインプ

リメントされてきている。

Cyclades における異機種分散型処理は、アプリケーションを直接容易にインプリメントするのに必要な道具としてのLogical Network Machine（LNM)と、それを利用するアプリケーションの２つから構成され、その関係は、図７－６のとうりである。

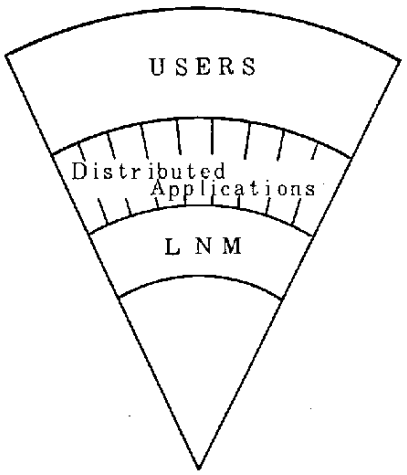


図７－６　分散型アプリケーションの構成

ＬＮＭは、異機種間の通信におけるデータリンクの設定などの問題を利用者に意識させる必要をなくするためのものである。

機能としては、ネットワーク間で通用する名前によるデータあるいは情報の交換と、ネットワークジョブなどのためのネットワーク言語の通訳などである。

上記の目的を達成するために、ＬＮＭのに共通な言語としてＩＧＯＲというインタープリートされる仮のコードが設定され、プログラムはこれによって作られることになっている。

ＩＧＯＲで作られたプログラムは、ＬＮＭのすべてのホストにおいて実行可能となる。

分散型データ・ベースのアプリケーションは、ＬＮＭの機能を利用することによっておこなわれる。

分散型データ・ベースは１個以上のアクセス法が分散したシステムと考えることができ、Socrates においては、次のアクセス法の分散が想定された。

(1) ネットワーク直接アクセス法

Socrates の本来の物理アクセス法である。ＩＢＭのＢＤＡＭに相当するアクセス法を、ネットワークアクセス法におきかえる。これによって図７－７に示すようなファイルの分散が可能になり、これから構成されるデータ・ベースを分散させることができる。

Cyclades の初期においては、この方法によって分散形データ・ベースの実現が考えられていた。

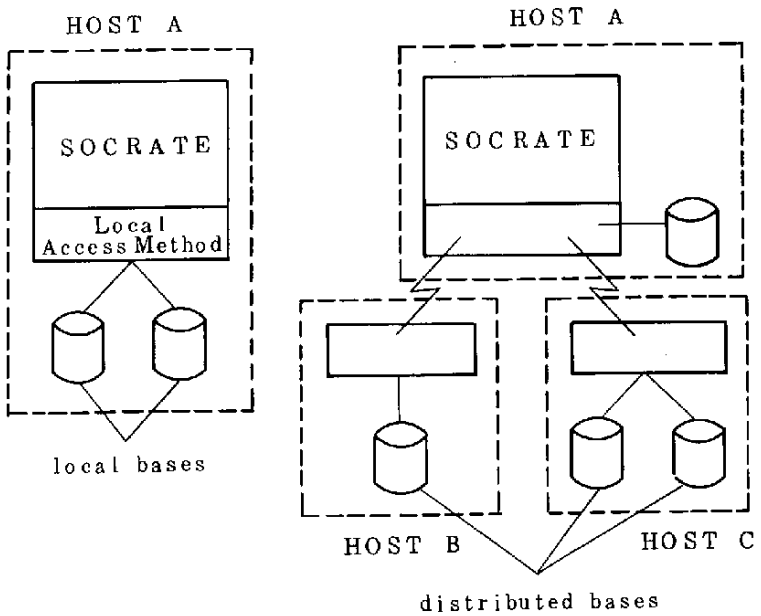


図7－7 ファイルの分散

(2) 論理アクセス法

Socratesの利用者言語は、編集、仮コードの発生、最終コードの発生という3つのステップによってコンパイルされる。

このコンパイルの過程において、仮コードをＬＮＭで述べたIGORに変換するか、あるいは、SocratesコンパイラにＩＧＯＲオブジェクトを出力させるように修正する。

仮コードのレベルは、データ・ベースの論理アクセス法に相当するので、この操作によって、論理的なレベルでのアクセスの分散が可能である。

Socratesにおいては、この方法で分散する実験もおこなうことが明らかにされている。

B ARAMIS

このプロジェクトはCycladesコンピュータ・ネットワークの一環として、ミニコンピュータにおいて自律性を失なわないでサービスを提供するようなデータ・ベースシステムの研究開発をおこなっているものである。

ARAMISネットワークは、小さな分散型データ・ベース管理システムを利用しようとする要請に合わせて設計されたものであり、次の3つの要素より構成されている。

(1) 各ホストにおいて、プロセス間コミュニケーションをサポートするＮＣＰ

(2) SYMBADデータ・ベース管理システム

(3) データ・ベースにアクセスを可能とするサブシステム

このシステムの特徴は

(1) データの存在場所は、放送機能を利用して、すべてのホストに対してデータの存在の有無を問合せをする。この応答によってその存在場所をを知ることができる。

(2) 問合せた結果、データが複数個のサイトに分散している場合は、次のような手順によってデータを集める。

利用者がいるサイトをmaster consumerとする。master consumerは、データに対する問合せメッセージを発信する。データを保有するサイトはproducerとなる。このproducerの中から、slave consumerを選び出し、ここにおいてデータの編集をおこなって、master consumerに送る。この関係を図7－8に示す。

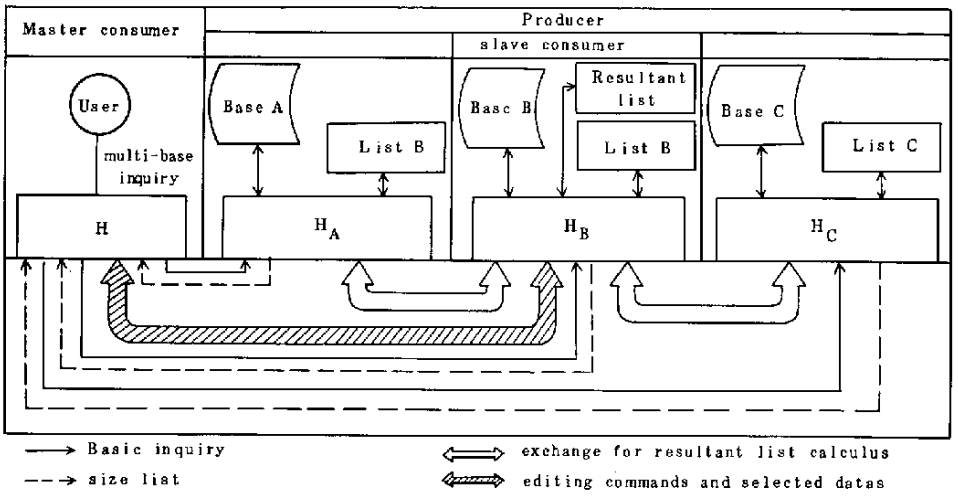


図7－8　ARAMISの制御分散

C　CONIT

（Connector for Network Information Transfer）

MITのMulticsを使って実験的に開発された文献検索インタフェースシステムである。

このシステムは、ARPAネットワークおよびTYMSHAREネットワークを通じて、MITのIntrex、NLMのMEDLINE、LockheedのDIALOG、SDCのORBITの4つの文献検索システムに共通コマンドによってアクセスが可能である。図7－9に、その構成図を示す。

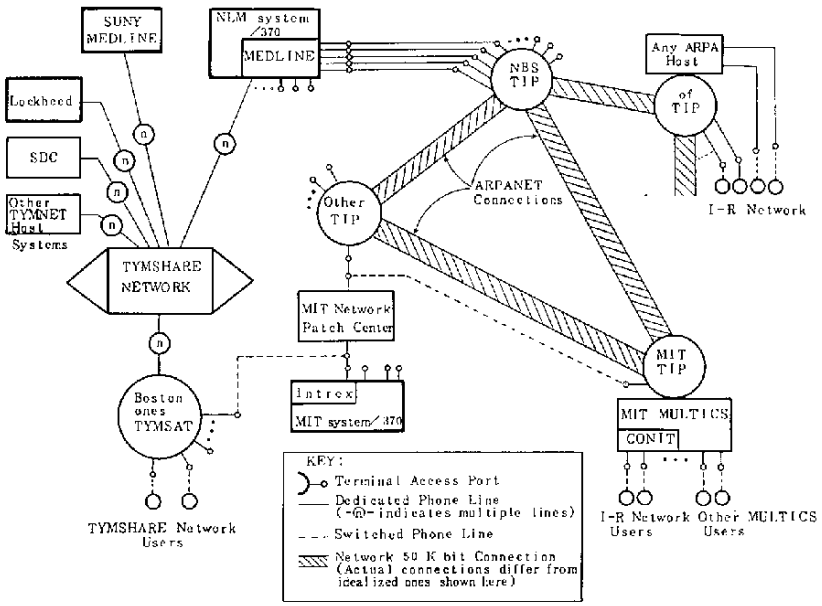


図7－9　CONITの構成

共通コマンド言語の翻訳システムであるCONITにアクセスすることによって、4つの文献検索システムのいずれに対しても、単にSELECTシステム名というコマンドによって、検索システムにログ・インでき、それを使用して必要な文献をさがしだすことができる。

このような共通言語システムを考えるに当って、必ず問題となるのは、共通言語が正しくローカルな言語（実際に利用するシステムの言語）に翻訳できるかどうかということ、およびそのために必要となってくるオーバヘッドである。

作成者によれば、共通言語からローカル言語への翻訳に関して、完全を期することは不可能であるが、ほぼ同等な機能への変換ならば十分にできることを指摘し、その程度で価値があると言っている。

またオーバヘッドに関しては、直接のアクセスに比して約25～50%のコストの追加があると考えられている。もし、シソーラスなどを利用

してより高度なサービスをおこなうとすれば５０～１００%以上のオーバヘッドの追加となると見込まれている。

これは個々のデータ・ベースは独立しているが、ユーザからみるとすべてのデータ・ベースが対象となり得るので機能的には分散した文献情報データ・ベースと考えることができる。

D　RSEXEC

RSEXECは分散された管理機能を持つシステムであって、ＡＲＰＡコンピュータ・ネットワークのTENEXホストコンピュータで動作する。TENEXホストとは、PDP－１０にページング・ハードウェアを追加したものである。

RSEXECをデータの共有という立場からみたときに注目しなければならないのは、ファイルをTENEXホストコンピュータ上ならばどこに保存していても利用できるという意味で、ファイルの分散に一応の成功をしたことである。この機能をＡＲＰＡネットワークのＴＩＰから自由に利用できるようにすることによって、ネットワークの利用を助けている。

機　　能

(1)　ファイルの存在場所

RSEXECにおいては、そのホストがローカルに管理しているファイルのディレクトリをそれぞれにおいて保持している。このディレクトリがRSEXEC全体のファイル・カタログの一部となるような形で構成されている。（図７－１０）参照

この機能によって、利用者がファイルの存在場所を指定しなくても、次々とカタログをたどることによって、その存在場所が明らかになる。

また存在の判明したファイルに対しては、レコードごとのアクセス要求をホスト間で交換しあうことによって、ファイルへのレコードごとのアクセスが可能である。これらの作業はすべてRSEXECの責任

においておこなわれる。

(2) アクセスできる範囲

・ RSEXECは前述の方法によって、分散されたファイルアクセスをサービスしているわけであるが、すべてのファイル操作をサポートしているわけではない。

それでも、TACO(Text Editor)、MACRO、LOADER などからの使用は可能であり、たとえば、TSSよりの edit、assemble、run という処理は、ソース・プログラム、実行形式プログラムなどが分散していたとしても実行可能となる。

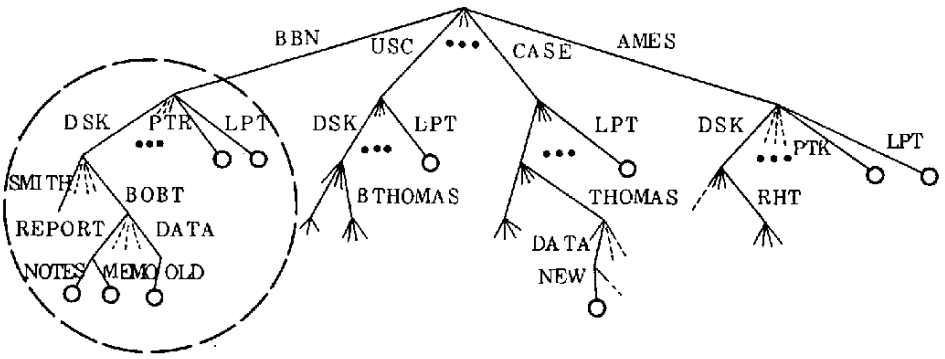


図7－10 RSEXECのファイル・ディレクトリ

E TIPログインシステム

TIPログインシステムは、ARPAネットワークにおいて、Terminal Interface Message Processor(TIP) からのログイン情報を複数個のPDP－10において管理している。

TIPログイン・システムは、ネットワーク・ユーザ・IDのデータ・ベースがすべてのTIPSER－RSEXECサイトにおいて保持されていることを要求する。

このデータ・ベースのそれぞれのコピーは、互いに独立のユーザ・エントリの集合である。このデータ・ベースに対する許される更新は、個個のユーザ・エントリの追加、修正、および消去である。

いかなるサイトにおいて初期化されようとも、更新を許し、すべてのデータ・ベースのコピーを一致するように編入することを保証するデータ・ベース・マネージメント技術が開発された。

"一致するように編入される"ことは、もしすべての更新アクティビティが中止されれば、すべてのコピーは最後には一意的なものとなることを意味する。

ユーザ・IDのデータ・ベースを保持するために使われた技術は、次の2つの部分より構成される。

(1) すべての更新がすべてのデータ・ベースのサイトに渡たる（一度そしてたゞ一度だけ）ことを保証するために、エントリを更新するサイトにおいて使われる信頼性のある、データに独立な更新の通知と分散メカニズム。

(2) 更新コマンドが到着したときは、データ・ベース・サイトにおいてアクティブになるデータに従属した更新処理手順。

ユーザ・IDのデータ・ベースに対しては、複雑な更新手続きが要求される。

データ・ベースのエントリに対する最新の更新は古いものにかぶせるという方法である。たとえば、ユーザのパスワードが変えられたならば古いパスワードは、新しいものと単純に入れ変えられる。

更新手続きは、異なったデータ・ベースのサイトのそれぞれにおいて、更新の事象の順に再構成し、動作するために、タイム・スタンプのメカニズムを利用するのが基本になっている。

さらに、データ・ベースのそれぞれのエントリ（と、修正可能なサブフィールド）は、現在の値にしたところの更新タイム・スタンプを保持

している。

ほとんどの更新コマンドがデータ・ベースのサイトに到着したとき、それが参照しているデータ・ベース・エントリのタイム・スタンプと単純に比較することにより編入あるいはリジェクトのいずれかを行うことができる。

エントリの消去と創成は少し違ったあつかいを必要とする。たとえば、もし、一つのエントリに対して創成と消去の両方のコマンドが異なったサイトで発生したならば、ネットワークあるいはシステムの機能が不完全であると三番目のサイトには、消去コマンドのあとに、創成コマンドがくるような事が起こる。このような場合をうまく処理するために、データ・ベースの更新手続は、消去コマンドが到着する以前に初期化されたエントリに対するすべての更新コマンドが確定するまで、消去コマンドに対する最終アクションが延期される。そして安全にデータ・ベースからエントリを消去できるのはその時点だけである。

ＴＩＰアカウント・システムの動作は、セグメント化されたデータ・ベースの創成と操作を生みだす。

アカウンティング・アプリケーションにおいて第一に関係するのは、データ・ベースの構成と便利なデータ・ベースのアクセスである。このデータ・ベースには、次のようなものが要求される。

(1) カタログ

データ・セグメント（インクリメンタル・アカウンティング・ファイル）をアクセスするために、それらの所在を知る必要がある。

カタログ機能は、RSEXEC分散形ファイルシステムによってあたえられる。

(2) エントリの重複がないことを保証する。

エントリは、アカウンティング情報を含むので、重複があってはならない。データの重複配置は行われているが、データ収集プロトコル

に於て、データ・エントリ自体の重複は存在しないように注意深く設計された。

(3) アカウンティング合計が作られるとき、それぞれのデータ・ベース・エントリはたゝ一度だけ処理されることを保証する。タイム・スタンプが、"たゞ一度だけ"の処理を保証する機能を持つということになる。

F Cashinの提案

複雑なデータ構造を持つデータ・ベースを共有するためには、それぞれのデータ・ベースにおいて使われるデータ・モデルとランゲージに関して、システム間で共通として使えるような合意が必要となる。

これらは、それぞれのデータ・ベースに使われている特定のアクセス法や記憶構造に従属したい十分ハイレベルな情報の操作を記述あるいは問合せする手段でなければならない。

現在のデータ・ベースシステムで使われている論理データ記述は、十分ハイレベルではない、というのは同じ情報が相いれない方法で論理的に構成される。たとえば、それぞれの在庫量に対して納入者を示す在庫リストと、納入者が納入した量を示す納入者リストを考えてみる。

同じ情報がアプリケーション側からの要求によって2つの異ったリストに保持されている。

英国のNational Physical Laboratory (NPL)では、共通データモデルによってハイレベルなデータ記述するための要求を調べることによってこれらの問題の調査がおこなわれている。もしこのようなモデルが合意されたら、システム間のすべての交信は、このモデルの言葉で表現される。この共通モデルの利用形態を図7－11に示す。

このようなモデルの特徴は、すべてのデータ・ベースシステムにおいてモデル化するすべての場合を表現できなければならない。一方、共通モデルはデータ・ベースとして直接インプリメントはされない。それは

存来のシステム間のコミュニケーションの手段としてのみ使われる。

リレーショナルモデルは、上記の要求を満たしている。このモデルの基礎は、いかなるデータ・ベースもいくつかのリレーションを持ったデータのセットで記述されることである。

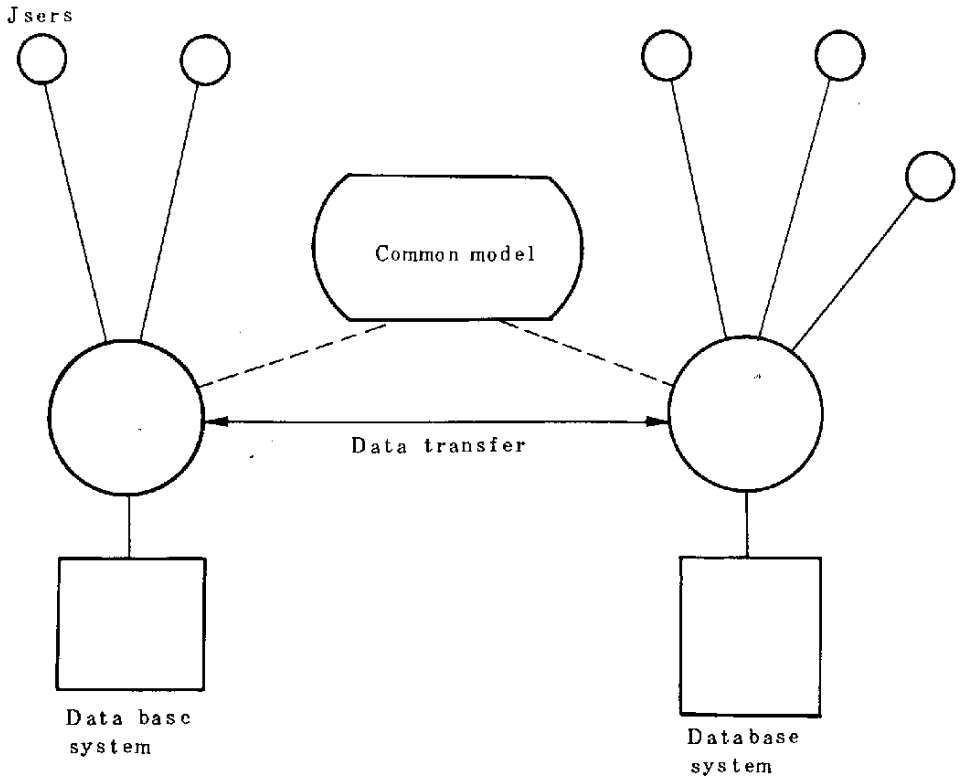


図7－11　共通モデルによるデータ・ベースの共有

G　Peeblesの提案

彼の提案では、ファイルのアクセス方法、ファイルに対する制御カードの指定事項は、それぞれのＯＳによって違いが見られるが、目的とすることは余り変わらないので統一的なネットワーク・コントロール言語（NLC）を作ることが可能であり、これを設定しようというものである。

彼は、ＩＢＭ３６０のOS/360、UNIVAC－1108のEXEC－８とを比較して、彼の設定したＮＣＬとこれらのＯＳジョブ制御言語の類似性を述べている。

表７－３に、ファイルを記述する制御言語の例を示すが、これらの類似性をふまえて、ＮＣＬをネットワーク・データ管理で解読し、それぞれ固有のジョブ制御言語に変換してアクセスできるようにしようとするものである。

表７－３　Peeblesの提案するＮＬＣ

FILE－REQ	DD(statement)	@ASG
filename	DSNAME	NAME
generation	(part of DSNAME)	(part of NAME)
node－name	－	－
buffer address	－	－
<s>	－	KEY1／KEY2
NEW	DISP＝NEW	(implied)
device	UNIT＝(type)	(implied)
space	space ＝	RESERVE
CATLG DECATLG DESTROY TEMP	CATLG UNCATLG DELETE DELETE	C－catalogue U－uncatalogue D －delete K －delete

表7－4　Peeblesの提案したコマンド

コマンド名	機　　能
Read－n	1個以上のレコードをリモートHOSTのファイルから読み出す。
Write－n	1個以上のレコードをリモートHOSTのファイルに書き出す。
Pelease－n	排他的制御のレコードをリソースする。
Open－n	利用者がファイルをアクティブにする。
Close－n	ファイル処理の終了を示す。
Eind－n	順インデックス・ファイルにおいて、特定のレコード位置に位置付けする。
Cntrl－n	リワインド、バックスペースなどの位置制御をおこなう。
Copy－n	同一HOSTにおいて一方から他方のファイルにコピーする。
File－req	リモート・ファイルのアクセス要求をする。
End－req	リモート・ファイルのアクセスが終了したことを示す。
Comac	分散してデータ・アクセスをするプロセス間で、コマンドの実行が正常に終了したことを通知する。
Syser	コマンドの実行が失敗したときエラー状態を通知する。
Terminate	データをアクセスするプロセスに対してファイル処理が終了したことを通知する。
Activate	将来の拡張のために、データ・アクセス・プロセスをユーザが作って呼び出せるようにしたもの。
Save	作成したアクセス・プロセスを保存することを指示。
Delete	アクセス・プロセスをファイルから消去する。
Describe／Get－description	ネットワーク・ライブラリに登録されたファイルのレコード構造も保存する。保存するコマンドがDescribe、読み出しをするのがGet－deseription である。

H　du Masleの提案

彼の意見はネットワーク・ジョブ制御言語（NJCL）の提案である。NJCLを作り上げるという意味では、Peebleのそれと全く同じであ

るが、アプローチが全く異なっている。すなわち、du Masleは幾種類かのＪＣＬを調査し、ネットワーク利用者の将来の必要性を評価したのち、新しい言語を作成するという方法を否定している。

彼の意見では、将来の利用者はＪＣＬに予期されていなかったサービスを期待することは確かであり、伝統的なＪＣＬは、機能を仕様決定時に固定されてしまい、利用者を満足させることはできない。このために、ＮＪＣＬのあるべき姿としては、利用者に高級プログラム言語の伝統的機能を提供できるような言語であると同時に、期待しうるサービスを具体的に定義できなければならない。

このような機能を持たせるためには、従来のパラメトリックなＪＣＬではなく、高級言語としてのＮＪＣＬにしなければならないことを強調している。これらを満足させる高級言語としてPASCALを利用することを提案している。その要因としては次のものがあげられている。

(1) 単純でかつ満足できる能力を有する。

(2) データタイプの定義機能が備わっている。

(3) 効率が良い。

(4) ポータビリティがある。

PASCALを使用することに、利用者はジョブ制御言語ばかりではなく、手続きも同時に記述できるために一つの言語ですべての用が足せることを利点として述べている。

7.5 分散型データ・ベースの現状および今後の方向

7.5.1 現　状

すでに述べてきたように、データ・ベースに関していえば、実用化の第一期をおえて、さらに統合された巨大化、複雑化したデータ・ベースを有効にとりあつかうことができるシステムを目ざして研究開発がおこなわれている。

CODASYL仕様、Coddのリレーショナル・モデルの両者に関して、種々の研究開発がおこなわれているとはいっても、実用化の第二期において主流となるデータ・ベースの思想という形では確立していない。分散型データ・ベースがその有効性を発揮するためには、実用化の第二期におけるデータ・ベースの思想というものが切望される。というのは、Cashinの提案におけるような共通データモデルに重要な影響をあたえるからである。

実用化第二期のデータ・ベースの思想が明らかになっていないこともあって、前項のデータ・ベースシステムの例にみられるように、分散型データ・ベースでは、どのような形式のデータ・ベースをどのようなアクセス法で共有化させるのかという基本的な方針が明確になっていない。

このことが現在における分散型データ・ベースの主要な問題点である。この問題は、分散型データ・ベースが独自に解決する性格のものではなく、在来のデータ・ベースの研究と歩調をあわせて解決されてゆくものである。この意味で、分散型データ・ベースにおいては、データ・ベースの研究成果待ちといった要素が多々あるといってよい。この種の問題は、データ・ベースの研究成果が、分散型データ・ベースの進歩をうながし、その結果が次の要求を生みだすというような経過をたどって進んでいくと考えられる。この例をあげるとすれば、統一分散管理方式のデータ・ベースの実用化であろう。

種々の問題点を含みながら、データ共有の要求を満たすための開発というのが、分散データ・ベース独自の努力によっておこなわれている。これは、CONITのように、利用者の立場に立って共通のコマンド設定をおこなうことによって分散型データ・ベースを実現しようとするアプローチ、あるいは、RSEXECのように、データ管理というレベルで分散処理をすることによって、データの共有を達成しようとするものである。

いずれにしても、いまだ実験段階のものであって、在来のデータ・ベー

スシステムが実用化の第一段階にいるのにくらべて遅れが目立っている。

しかしながら、データ・ベースの歴史が15年、分散処理の歴史が9年であることを考えれば、この程度のギャップはやむを得ないであろう。

最近になって、各コンピュータ・メーカが、ネットワーク・アーキテクチャと称して分散型処理技術の統一化を目指していることから、各メーカごとに、標準となった分散型データ・ベースというものが作られる可能性がでてきている。

いずれにしても、分散型データ・ベースに関する限り、ニーズは明らかにあるが、方向性は確立していないということができる。

7.5.2 今後の方向

前項のニーズに関する分類を整理してみると、次のようになる。

(1) 同一企業内のデータの共有

(2) 同種企業間のデータの共有

(3) サービス会社の利用者間でのデータの共有

(4) 大学・研究所間のデータの共有

(5) 官庁間のデータの共有

(6) 国際的レベルでのデータの共有

これらの内部におけるデータ処理システムは、それぞれの場所で必要性が大きいために個別に作られたものである。データの共有は、共有ということによってさらに次元の高いシステム、よりパフォーマレスのよいシステム、よりサービス性の高いシステムになり得る故に、おこなわれるわけである。

これらの個々のシステムのその性格からして、その対象分野内での効果最大を期して自然発生的に構成員の合意のもとに設置されたものであり、データの共有にあたって全体的にバランスをどうするかというように、全体的な立場から意図的、計画的に作られるものではない。

同一企業内におけるデータの共有システムにおいては、企業という一つの統一的な命令系統を持った組織において作成するわけであるので、利害の不一致があったとしても、全社的な見地に立った判断という形での合意をとりつけることは可能であろう。しかしながら、同一資本系列内、同種企業間、大学間、官庁間、国際的なレベルとなってくると、統一的、計画的な実行が要求される反面、管理、運営、責任などの体制に関して明確なコンセンサスと取り決めが必要である。

このように考えると、利用される分散型データ・ベースは、統一的な命令系統を持った組織においてのみ有用なものと、それ以外のものにおいても利用されるものと2種類考えられる。

また、分散型データ・ベースの想定される利用形態、現状における実験例、提案などを総合して考えてみると、今後コンピュータ・ネットワークにおける分散型データ・ベースシステムは次の2つの形態が主流となってゆくと思われる。

(1) 弱い結合形態の分散型データ・ベースシステム
(2) 統合管理された分散型データ・ベースシステム

弱い結合形態の分散型データ・ベース

弱い結合形態の分散型データ・ベースとは、前項の分類にしたがえば、CONITのように、局所管理、不均質、データの冗長性を持たないシステムのことである。

この形態は、次の点において非常に有効であると考えられる。

(1) コンピュータ・ネットワークの前提としている自律的なコンピュータの相互結合によるリソースの共有という考え方になじみやすい。

すなわち、ローカルなセンターの自治の下で、サービスだけを授受しあうという形態であって、各ローカルなホストのサービス方法、形態は各サイトの管理にまかされる形となるので、ネットワーク化の協力が得やすい。

(2) データを共有する組織あるいは機関が、機構上、政策上、運営上、従来からのサービスの継続など種々の理由によって、統一的、均質的なデータ・ベースの共有ができなくても利用が可能

現在のところデータの共有は、特定の端末から各データ・ベースに特有な個々のアクセス法を利用することによっておこなわれている。この形態のシステムの開発によって、サービス・ホストの許可があれば、任意の端末あるいはプロセスからデータ・ベースの共通が可能となる。

この方式の次のステップは、CONITあるいはCashin の提案にみられるように、共通言語によるアクセスあるいは共通のモデルを使用したデータ・ベース相互間のインタフェースおよび自分の欲するデータがどこに存在するかを適格に知らせることができるクリアリングセンターの構成法である。

前者は、共通言語あるいはモデルの機能範囲と処理効率が問題とされ後者は、利用者の欲する情報をどのようなレベルまでこまかく正確に回答できる構成法が問題とされる。

統合管理された分散型データ・ベース

統合管理された分散型データ・ベースとは、前項の分類でいえば、統一分散管理、均質で冗長性のあるデータをもサービスできるシステムである。この機能がすべて満足できるシステムは、いまのところ現われていない。しかし一部の機能の実験は、ＴＩＰログインシステム、SOC—RATES, ARAMISなどでおこなわれている。

この形態は、一つの企業内で、全国に分散された事業所が特定の機構の下に統括されている組織が新らたにシステムを作り上げる場合などに有効となるものである。

このシステムは、現在の技術でいえば、各事業所ごとにプログラムが存在し、メッセージ交換によって情報が交換されていたものが、分散型データ・ベースの出現によって、それらの処理がすべてプログラム側か

からシステム側に移されることになる。

この方式の成否は、各ホストの制御情報がいかに少なくできるかということ、利用者がいかにうまくデータ・ベースを分散できるかにかかってくるであろう。

第7章の参考文献

(1) Chupin J. C.
Control concepts of a logical network machine for data Banks. IFIPS 74 Stockholm. p. p. 291～295

(2) Chu W. W.
Optimal file allocation in a multicomputer information system. IFIPS 68 Edinburgh. pp. F80～F85.

(3) Casey R. G.
Allocation of copies of a file in an information network. SJCC 72. vol. 40. pp. 617～625.

(4) Casey R. G.
Design of tree networks for distributed data. NCC 73. vol. 42. pp. 251～257

(5) Booth G. M.
The use of distributed data bases in information networks. ICCC' 72. pp. 371～376.

(6) Morgan H. L, Levin K. D.
Optimal program and data locations in computer networks. AD/A—001008, 1974. P. 22.

(7) Bihan J. L.
A survey of problems in distributed data bases. Reseau Cyclades DAT510, July 1975, p. 6.

(8) BBN

Interface Message Processor for the ARPA computer network. Quarterly technical report №2. 1 April to 30 June. BBN report №3106. July 1975. p. 78.

(9) Farber D. J. Heinrich F. R.

The structure of a distributed computer system — The distributed file system, ICCC′ 72, pp. 364～370.

(10) Peebles R. W.

Design Considerations for a distributed data access system. AD775569. May 1973. p. 401.

(11) CII

SOCRATE. manual dutilisation, 1973, p. 124.

(12) Chupin J. C, Seguin J., Segeant G.

Distributed Application on Heterogeneous Networks. 3rd Annual Symposium on Computer Architecture Florida, 1976. 1.

(13) 名和小太郎

分散型ネットワークシステム── 旭化成のＡＣＴについて── 関東NEACユーザー会第4回例会資料, S. 49, p. 27.

(14) 名和小太郎

企業内ネットワークについて、情報処理に関するシンポジウム予稿、日本情報処理開発センター, S. 51, pp. 58～64.

(15) 上野　滋

官庁間コンピュータ・ネットワークのニーズと現在開発中、計画中のネットワーク・システムの概要について、情報処理に関するシンポジウム予稿、日本情報処理開発センター、S. 51, pp. 51～57.

(16) 日本情報処理開発センター

複合ＮＩＳ形成に関する調査研究報告書，47—R006，S.48，p.255.

(17) 電子技術総合研究所

パターン情報処理研究システム — 計算機複合体EPICS —，S.49，p.239.

(18) 野垣内　章

旅行業者と運輸・航空機関とのコンピュータ・リンク，ビジネス・コミュニケーション　75，vol.12，№2，pp.48～55.

(19) Combs B.

TYMNET：A distributed Network, DATAMATION 73, 7. pp. 40～43.

(20) Tymes L. R.

TYMNET—A terminal oriented communication network. SJCC' 71. pp. 211～216.

(21) 日本情報処理開発センター

米国におけるコンピュータ・ネットワークの技術動向，S.49，p.184.

(22) CODASYL

DATA BASE TASK GROUP REPORT ON THE CODASYL PROGRAMMING LANGUAGE COMMITTEE APRIL 1971, 71, p.273.

(23) Thomas R. H.

A resource sharing executive for the ARPANET NCC73, pp. 155～163.

(24) Cashin P. M.

Data Base Interworking, Network Systems and Software, Infortech State of the Art Report 24, pp. 435～459, 1975

(25) Shoshani A., Spiegler I.

The Integration of Data Management Systems on a Com—

puter Network. AIAA Computer Network Systems Conference, №73—417, p. 8, 1973.

(26) Booth G. M.
Distributed Information Systems, NCC'76, pp. 789～794

(27) Farber D. J.
Distributed Data Base—An Exploration, Proceedings of the 1975 Symposium Computer Networks : Trend and Applications. pp. 25～27

(28) Marcus R. S.
Network Access for the Information Retrieval Application, 1975 IEEE INTERCON Conference Record, 25/4, p7

(29) Lagasse J. P., Artaud G., Cabanel J. P.
ARAMIS—A Processing Network with User Data Bases Interactive System, COMP' 75, pp, 213～216

(30) Booth G. M.
Distributed Data Bases — Their Structure and Use, Distributed Systems, Infotech State of the Art Report, pp. 201～213, 1976

(31) Curtice R. M.
The Outlook for Data Base Management, DATAMATION April 1976, pp. 46～49

(32) Comba P. G.
Needed : Distributed Control, International Conference on Very Large Data Bases 1975, pp. 364～375

(33) Whitney K. M.
Fourth generation data management system, NCC' 73,

pp. 239～244

(34) Codd E. F.

A Relational Model of Data for Large Shared Data Banks. CACM Vol. 13, №6, pp. 377～387, 1970

(35) Abrial J. R.

Data Semantics, Data Base Management, pp. 1～60

(36) Codd E. F.

Recent Investigations in Relational Data Base Systems. Information Processing' 74, pp. 1017～1021

(37) Deppe M. E., Fry J. P.

Distributed data bases — A summaryof research, Computer Networks 1, pp130～138, 1976

(38) 相磯秀夫

マイクロコンピュータのアーキテクチャとシステム構成、情報処理 Vol. 17, №4, pp. 259～270, 1976

(39) 西野博二

データ・ベースシステムの研究開発動向、情報処理、Vol. 17, №10, pp. 878～882, 1976

(40) 土井喜一、関口　宏

分散型データベース、情報処理Vol. 17, №10, pp. 934～939, 1976

8．JIPNETの開発を終えて

8．JIPNETの開発を終えて

8.1　ＪＩＰＮＥＴの開発経過

ＪＩＰＮＥＴの開発経過を表８－１に示す。

開発初年度である昭和４８年度は、システムの基本構成を検討し、ネットワーク機能の検討およびハードウェア機器の仕様設定および導入を行ない、昭和４９年度にＪＩＰＮＥＴ第１バージョンのシステムをインプリメントした。第１バージョンは、基本的なものではあるが、コンピュータ・ネットワークとして最低限必要な、サブネット・コントロール、ホスト間通信、ＴＳＳ利用、ファイル転送、リモートジョブエントリの諸機能を有しており、このシステムを用いて実際のアプリケーション・プログラムによる利用実験を行った。

昭和５０、５１年度は、リプレースされた新ホストのＮＣＰを作成するとともに、第１バージョン・システムの評価により、改良版プロトコルを規定し、各ホストのＮＣＰおよび高位プロトコル処理プログラムの改良版である第２バージョン・システムをインプリメントし、総合的な評価をおこなった。とくに５１年度は、ＮＡＰ(Network Access Processor)のインプリメント、ＣＪＰ(Cooperated Job Processing)の実験等を行うとともに、ネットワーク利用の高度化を図る為の分散型データ・ベース(ＤＤＢＡ)やネットワーク・ジョブ制御言語(ＮＪＣＬ)の検討をおこなった。

表8－1　JIPNETの開発経過

時期（年・月）	内　　容
昭和48年度	（調査・検討段階）
48. 4	JIPNETの開発開始
48. 4～ 6	ネットワークの構成、結合方式および異機種結合上の問題点検討
48. 4～ 8	結合用ハードウェア機器選定
	IMP用に、N3200／50を採用　最新ターミナルを数台選定
48. 4～ 9	既存コンピュータ・ネットワークの技術調査
48. 6～ 9	JIPNETの基本機能設定
48. 7～ 9	結合用インタフェース・アダプタ機能検討
48. 7～11	ハードウェア・モニタにより既存ホストの効率測定
48. 9～49. 3	基本プロトコル（ホスト－ホスト、ホスト－IMP、IMP間）の検討
	NCP（FACOM、HITAC）の実現手段検討
	サブネット機能の論理設計
49. 1	ARPAのDr. R. E. Kahn によるレクチャー開催
49. 1	プログラミング・シンポジウムにて論文発表
49. 1～ 3	IMPおよびターミナルの設置、結合テスト
昭和49年度	（第1バージョンのインプリメント）
49. 4～ 8	高位プロトコルとしてDSP、FTP、RBPを規定
49. 4～10	サブネットのソフトウェア作成
49. 4～11	NCP（F、H）の作成
49. 6～ 7	アダプタの結合テスト
49. 8	ホスト・コンピュータF／60をF／75にリプレース
49. 8～11	F／75用サーバDSP、サーバFTP、ユーザRBP作成
	H-8450用サーバRBP、ユーザRBP作成
	TIP用ユーザDSP作成

時期(年・月)	内　　容
昭和４９年度	（第１バージョンのインプリメント）
49.12	ＪＩＰＮＥＴ第1バージョン完成
	工技院利用研に対しデモンストレーション
50. 1～ 2	サブネットの効率測定
50. 2	Ｎ／５００をＡＣＯＳ／７００にリプレース
50. 3	ＮＣＰの効率測定
	ＡＣＯＳ／７００の結合テスト
昭和５０年度	（評　価・改　良）
50. 4～ 6	ホストーホスト・プロトコルの改良
	ＤＳＰの改良
50. 4～12	海外の技術資料の調査
	（仮想マシーン、分散型データ・ベース等）
50. 6～11	改良版ＮＣＰ（Ｆ／75、Ｈ−8450）の作成
	ＡＣＯＳ　ＮＣＰの作成
	改良版ＴＩＰの作成（ユーザＤＳＰも含む）
	改良版ＤＳＰの作成
	Ｆ／７５用サーバＤＳＰ
50. 8	第２回ＵＪＣＣにて論文発表
50. 9	海外調査（米国、欧州）
50.11	情報処理学会　全国大会に７論文発表
50.12～51. 1	改良システムの効率評価（ＮＣＰ、ＤＳＰ）
50.12～51. 3	ＡＣＯＳ用サーバＤＳＰの作成
51. 1～51. 5	改良版ＦＴＰの作成
	Ｆ／７５用　　ユーザＦＴＰ、サーバＦＴＰ
	Ｈ−８４５０用　ユーザＦＴＰ、サーバＦＴＰ

時期(年・月)	内　　容
昭和５０年度	（評　価・改　良）
51. 1～51. 5	改良版ＤＳＰの作成 F/75用　ユーザDSP 改良版ＮＴＣＰの作成 F/75用　モニター組込
昭和５１年度	（総合評価、高度機能の検討）
51. 4～52. 1	高度機能（DDBA、ＮＪＣＬ）の技術調査および検討
51. 4～52. 3	NAP/NFEPの作成
51. 7～ 9	改良版システムの総合評価 ＮＣＰ、ＤＳＰ、ＦＴＰ
51. 7～10	改良版ＲＢＰの作成 F/75用　ユーザＲＢＰ H-8450用 サーバＲＢＰ
51. 7～12	ＮＪＣＬの言語仕様および実現性の検討
51. 9～52. 3	ＡＣＯＳ用ユーザＦＴＰの作成
51.11	情報処理学会全国大会にて4論文発表
51.11～52. 2	ＣＪＰによる実験

8.2　ＪＩＰＮＥＴの成果

ＪＩＰＮＥＴの研究開発は、本年度をもって一応の区切りがつけられることとなる。4年間に渡るＪＩＰＮＥＴ基本ファシリティの開発にあたっては、数多くの問題に遭遇し、各種の貴重な体験を得ることができた。計画当初の段階で設定したＪＩＰＮＥＴの実用、実験上の目的（1.1項参照）の大半は、この4年間の開発をとうして一通り達成し得たと言えるが、今後に残された問題としては、ホストおよび端末のネットワークへの結合に当って、より経済

的・効率的な技術の開発、国際的なプロトコル標準化動向への対応、ユーザのニーズに密着した新しいネットワーク利用技術の開発等、より高度なレベルの実現が期待される。

4年間に渡るＪＩＰＮＥＴの研究開発は、我国におけるコンピュータ・ネットワークの実現に対する先駆的実験研究の一つとして、十分その役割を果し、とくに異機種間結合にともなう問題点の解明、パケット交換網の伝送効率の定量評価、各種のプロトコル処理における既存ＯＳの問題点の解明、ネットワーク結合にともなうホスト・コンピュータの負荷と効率の定量化、NFEPを介したネットワーク結合の定量評価、ネットワーク・プロトコルの定量的・定性的評価、ネットワーク開発工数の定量化など、今後開発される各種ネットワーク・システムのよき参考となる事と信ずる。

なお、表８－３にＪＩＰＮＥＴ関係の外部発表論文を、表８－４に開発したソフトウェア群を示す。

表8－3　ＪＩＰＮＥＴの外部発表および論文

年月	会議名	発表内容および論文
49. 1	プログラミング・シンポジウム	ＪＩＰＤＥＣにおけるコンピュータ・ネットワーク・システム
49.10	電気四学会連合大会	コンピュータ・ネットワークにおけるソフトウェア
49.12	ＪＩＰＤＥＣ調査研究発表会	コンピュータ・ネットワーク・システムＪＩＰＮＥＴ
50. 2	情報処理学会ネットワーク講習会	コンピュータ・ネットワークにおけるＨＯＳＴ－ＨＯＳＴプロトコル
50. 3	東北大学通研シンポジウム	ＪＩＰＮＥＴの基本機能
50. 5	大阪大学シンポジウム	コンピュータ・ネットワークのソフトウェア
50. 8	ＰＡＣＮＥＴ国際会議	Computer Network System ＪＩＰＮＥＴ
50. 8	第２回日米コンピュータ会議	Packet Switching Subnet of ＪＩＰＮＥＴ System
50.10	情報処理学会ネットワーク講習会	・コンピュータ・ネットワークにおけるＨＯＳＴ－ＨＯＳＴプロトコル ・コンピュータ・ネットワークＪＩＰＮＥＴ
50.11	５０年度情報処理学会全国大会	・コンピュータ・ネットワークＪＩＰＮＥＴ ・ＪＩＰＮＥＴのＨＯＳＴ－ＨＯＳＴプロトコル ・ＪＩＰＮＥＴにおける高位プロトコル ・ＪＩＰＮＥＴにおけるＴＩＰソフトウェア ・ＪＩＰＮＥＴのＦＡＣＯＭ２３０／７５用ＮＣＰ ・ＪＩＰＮＥＴのＨＩＴＡＣ－８４５０用ＮＣＰ ・ＪＩＰＮＥＴのＡＣＯＳシステム７００用ＮＣＰ
50.12	ＪＩＰＤＥＣシンポジウム	・コンピュータ・ネットワークの現況
51. 1	情報処理学会コンピュータ・ネットワーク研究会	・コンピュータ・ネットワークにおけるＮＣＰ実現上の問題点 ・コンピュータ・ネットワークのＨＯＳＴ／ＩＭＰアダプタ
51. 9	情報処理学会ネットワーク講習会	・ＨＯＳＴ－ＨＯＳＴおよびハイレベル・プロトコルの動向
51.11	５１年度情報処理学会全国大会	・ＪＩＰＮＥＴにおけるホスト間通信の効率評価 ・ＪＩＰＮＥＴにおけるファイル転送の効率評価 ・ＪＩＰＮＥＴにおけるＴＳＳ利用プロトコルのインプリメントと効率評価 ・ネットワーク・ジョブ制御言語：ＮＪＣＬ

表８－４　ＪＩＰＮＥＴの開発ソフトウェア

ソ　フ　ト　ウ　ェ　ア	プログラム・サイズ（バッファ）
サブネット関係	
ＩＭＰ制御プログラム	１７　KW（　７　KW）
ＴＩＰ制御プログラム	１３.５KW（　４.５KW）
Ｎ３２００クロスアセンブラ（Ｆ／７５）	４６　KW
ACOSホスト関係	
ACOS ＮＣＰ	２１　KW（　８　KW）
サーバ　ＤＳＰ	４　KW（　１.３KW）
ユーザ　ＦＴＰ	１５.７KW（　４.３KW）
テクトロ用ＧＳＰ	２.２KW
Ｆ／７５ホスト関係	
Ｆ／７５　ＮＣＰ	２０.７KW（１５　KW）
ＮＴＣＰ	１.２KW
ユーザ　　ＤＳＰ	３.１KW
サーバ　　ＤＳＰ	２.５KW（　１　KW）
ユーザ　　ＦＴＰ	１１.４KW（　３　KW）
サーバ　　ＦＴＰ	１１.０KW（　５　KW）
ユーザ　　ＲＢＰ	９.８KW（　５　KW）
テクトロ用ＧＳＰ	３.２KW
Ｈ－８４５０　ホスト関係	
Ｈ－８４５０　ＮＣＰ	４５.６KB（２６.１KB）
ユーザ　　ＦＴＰ	６２.８KB（　５.８KB）
サーバ　　ＦＴＰ	５８.５KB（１７　KB）
サーバ　　ＲＢＰ	３６.３KB（１３.２KB）
ＮＡＰ関係	
ＮＡＰ制御プログラム	６０　KB（２４　KB）
Ｕ２００クロスアセンブラ	７５　KW（４０　KW）

（注）　表８－４は最終製品のみで改良前の旧バージョンのものは含まない

あ と が き

ARPANETの稼働が開始されてすでに6、7年の歳月が経過し、商用のコンピュータ・ネットワーク・サービスも徐々に軌道に乗り始めた。一方、現存のシステムは真に理想とするコンピュータ・ネットワークとは程遠く、ネットワークがその真価を発揮するには、現在のシステムになお多大の論理的機能を付加する必要があり、その実用化にはなお多年にわたる研究と実証の努力が重ねられねばならねことも指摘されている。

各種の異なったリソースの真に有効な共用を実現するには異機種間のデータやデータ・ベースの完全な互換性の確立、ハードウェアおよびソフトウェアの異機種間の自由なポータビリティやロードレベリングの実現、ユーザの操作性向上のためのネットワーク・アクセス言語等、解決せねばならぬ難題がなお多数存在している。

JIPNETの目標もこれら新技術の一角を実現し、その有効性の検証を試みることにあるが、その実験台としての基本的ネットワーク・ファシリティのインプリメントの段階に於ても幾多の問題に遭遇し、それはそれで貴重なる体験であった。

我が国に於いても、各コンピュータメーカのネットワークアーキテクチャ構想の確立を始め、近年急激にシステムのネットワーク化に対するニーズが高まりつつある。とくにリソースの分散とその共用はコンピュータ利用形態の必然的な要求として経済性、効率性、信頼性の優れた真の実用システムの出現が期待されている。

本プロジェクトは4年間の研究開発目標をほぼ達成し今年度をもって一応完結するが、本プロジェクトの研究開発の貴重な機会を与えられた事を感謝するとともにJIPNETに課せられた使命の重大さを痛感し、今後ともこの経験を踏み台として、更に新しい研究開発が継続し得ることを念ずるものである。

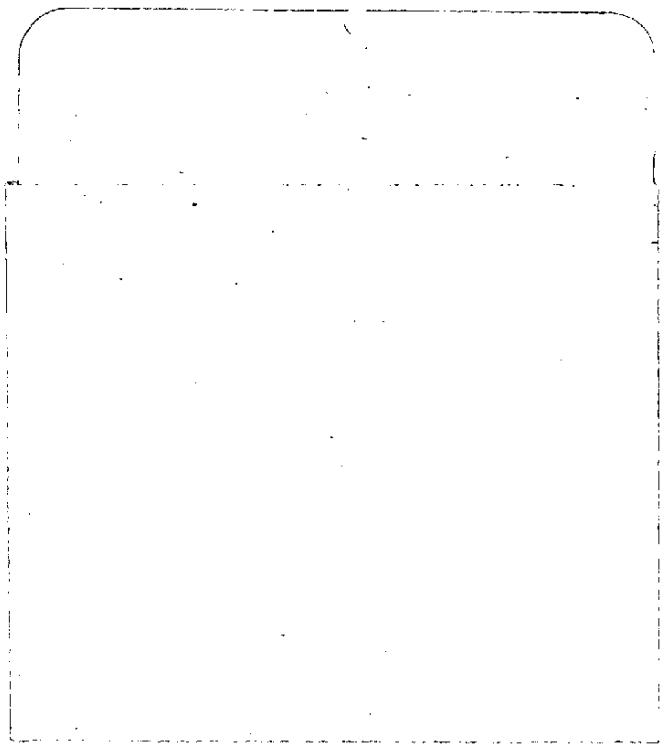
最後に、本プロジェクトの遂行にあたり、誠意ある協力を惜しまれなかっ

た、日立製作所、日本電気、富士通各社の関係各位に心から感謝の意を表する。

禁無断転載

昭和52年3月発行

発行所　財団法人　日本情報処理開発協会
東京都港区芝公園3－5－8
機械振興会館内
TEL（434）8211（代表）

印刷所　株式会社　三州社
東京都港区芝大門1－1－21
TEL（433）1481

51－S003