修士論文

題目

共有メディアにおける利己的なエージェントの振舞いの観察

指導教授

所 眞理雄 教授

慶應義塾大学大学院

理工学研究科 計算機科学専攻

1996 年度

89524385

成田 多良

修 士 論 文 要 旨

専　攻	計算機科学	学　籍 番　号	89524385	フリガナ 氏　名	ナリタ トミオ 成田 多良

(論 文 題 名)

共有メディアにおける利己的なエージェントの振舞いの観察

(内 容 の 要 旨)

（内容の要旨は約25行程度で記入のこと）

近年, 計算システムにおける市場性, 経済性を重要視する気運が高まっており, 経済学的なアプローチから, ネットワークやマルチエージェント・システムにおける資源分配を分析する研究がなされている. これらの研究は, 不確実性を持つシステムにおける, 必ずしも協調的とは仮定できないユーザ間の資源共有を論じたものであり, 将来の開放型分散環境における資源分配を考える上で必要不可欠な問題を取扱っている.

本論文では, コストを考慮した利己的なエージェントによる資源競合という観点から, 共有メディア型ネットワークをテストベッドとして, 複数のエージェントによって構成される人工生態系としてモデル化を行なう. 本モデルにおいて, エージェントは遺伝的に獲得される戦略に従ってサービスを提供する. また, 性能とコストの関数で表される独自の効用関数を持ち, 効用の時間的変化に対して独自の判断を下すことによって他のエージェントを選択淘汰し, 試行錯誤を通じて自己の効用を最大化すべく振舞う.

本稿では, 上記のモデルを実行するシミュレータを設計・実装し, CSMA 型共有メディアにおいて複数のエージェントが利己的に振舞う環境での, ネットワーク・システム全体の進化的な挙動を観察する. そして, シミュレーションの結果から, 必ずしも全てのエージェントが協調的だとは限らない場合であっても, コストを考慮することによって, 比較的短い時間に低いコストで, エージェントが協調的な振舞いを見せる集団へと自己組織化し, 結果的にシステムが安定化することを示す.

Summary of M. S. Thesis

Major	Student No.	SURNAME, Firstname
Computer Science	89524385	NARITA, Tomio

Title

Observation of Selfish Agents' Behavior on Media Resource Sharing

In recent years, there has been growing momentum for studies on a sense of economy and marketability in computer systems. Taking account of users' satisfaction with resource allocation is addressed as a key component in economic approach such as game-theory, and also considered agents' incentives in theoretical models of artificial ecosystems. Discussion about resource contention on such environments can be applied to the notion of *open system* where users are potentially selfish and systems inherently involve uncertainty stems from their embodiment.

We employ a single shared medium that is associated with *pricing* but technically performs like the Ethernet as a test-bed for modeling an artificial ecosystem in terms of resource contention among selfish agents. In our model, each agent has a *gene* which characterizes its service strategy, and an *utility function* determines its attitude toward resource contention. Agents make efforts eventually to maximize own utility by selecting one agent as a contractor after another through trial-and-error.

We design and implement a network simulator based on our model, and conduct some simulations to observe selfish agents' behavior and look into dynamism of evolutional artificial ecosystems. Simulation results say that although there is no premise that agents always act in a cooperative manner, cost-sensitive agents can collectively self-organize into a mass which exhibits cooperative appearance at relatively low cost in the short term, and consequently, a network system can reach stable equilibrium as a whole.

Graduate School of Science and Technology, KEIO UNIVERSITY

(The summary of thesis should be written within 35 lines)

目次

1 序論 **1**
1.1 イントロダクション . . . 2
1.2 目的 . . . 4
1.3 本論文の構成 . . . 5

2 関連研究 **6**
2.1 Internet Economics . . . 7
2.1.1 ユーザの満足度 . . . 7
2.1.2 効率とは何か? . . . 7
2.1.3 輻輳制御と Responsive Pricing . . . 8
2.1.4 課金方法のスキーム . . . 9
2.2 利己的なユーザによる資源共有 . . . 11
2.2.1 非協調的アプローチによる輻輳制御 . . . 11
2.2.2 輻輳制御のゲーム理論的分析 . . . 12
2.2.3 平等とは何か? . . . 14
2.2.4 CSMA とゲーム理論 . . . 15
2.3 人工生態系と遺伝的アルゴリズム . . . 17
2.3.1 Computational Ecosystem . . . 17
2.3.2 戦略の遺伝的適応 . . . 18
2.3.3 進化するネットワーク . . . 19
2.3.4 人間世界と人工生態系の融合 . . . 20

3 シミュレーション・モデルの構築 **22**
3.1 アーキテクチャ . . . 23
3.2 共有メディア . . . 24
3.3 CSMA エージェント . . . 24
3.3.1 CSMA エージェントの突然変異 . . . 25
3.4 ノード . . . 26
3.4.1 CSMA エージェントの観測 . . . 26
3.4.2 ノードの効用関数 . . . 26
3.4.3 ノードの判断のタイミング . . . 27
3.4.4 ノードの突然変異 . . . 27
3.5 ユーザ . . . 27
3.5.1 ノードの観測 . . . 27
3.5.2 ユーザの効用関数 . . . 28
3.5.3 ユーザの判断のタイミング . . . 28
3.6 情報ブローカ . . . 28

4 シミュレーションと評価 **31**
4.1 予備シミュレーション . . . 32
4.2 利己的なエージェントによる資源共有 . . . 35

4.2.1 設定 . . . 35
4.2.2 コストを考慮することの意義 . . . 36
4.2.3 義務としての協調 . . . 40
4.2.4 利己的なエージェント . . . 41
4.2.5 システムの進化 . . . 45
4.2.6 エージェントの変更に伴うコスト . . . 48
4.2.7 平等性についての評価 . . . 49
4.2.8 まとめ . . . 50

5 議論 **54**
5.1 複数の異なる QoS の実現 . . . 55
5.1.1 異なる効用関数を持つエージェントによる資源共有 . . . 55
5.1.2 効用関数の進化 . . . 57
5.1.3 異種の QoS の実現 . . . 60
5.2 Accounting に関する考察 . . . 60
5.3 進化のモデルの妥当性 . . . 61

6 結論 **64**

謝辞 **66**

参考文献 **67**

A シミュレータの設計と実装 **71**
A.1 SimFrame 概論 . . . 72
A.2 フレームワーク . . . 72
A.2.1 イベント . . . 73
A.2.2 イベントによる同期 . . . 74
A.2.3 イベントのスケジューリング . . . 75
A.2.4 スケジューラ . . . 77
A.3 プリミティブ . . . 78
A.3.1 イベントの配置 . . . 78
A.3.2 イベントの送信 . . . 82
A.3.3 イベントのブロードキャスト . . . 82
A.3.4 状態遷移マシン . . . 83
A.4 関連研究 . . . 83

図 目 次

2.1 スマート・マーケット 10
2.2 スイッチを中央に置いたネットワーク 12
2.3 人工生態系内の雇用関係 20
2.4 メディア・アクセス方式の進化木 21

3.1 シミュレーション・モデルの構成要素 23
3.2 CSMA メディア 24
3.3 情報ブローカ 29

4.1 pp-CSMA/CD 特性 (a = 0.0232) 32
4.2 pp-CSMA 特性 (a = 0.0232) 33
4.3 pp-CSMA/CD 特性 (a = 0.232) 34
4.4 pp-CSMA 特性 (a = 0.232) 34
4.5 コストを考慮しない場合のスループット 36
4.6 コストを考慮する場合のスループット 37
4.7 コストを考慮しないエージェントによる p の選択 38
4.8 コストを考慮するエージェントによる p の選択 38
4.9 コストを考慮しないエージェントによるαの選択 39
4.10 コストを考慮するエージェントによるαの選択 39
4.11 コストを考慮しない場合の義務としてのバックオフの効果 40
4.12 コストを考慮した場合の義務としてのバックオフの効果 41
4.13 コストを考慮しない利己的なエージェントとバックオフ 43
4.14 コストを考慮する利己的なエージェントとバックオフ 43
4.15 コストを考慮しない場合のバックオッフを使用するエージェントの比率 . . 44
4.16 コストを考慮する場合のバックオッフを使用するエージェントの比率 . . 44
4.17 義務的協調による p の選択 46
4.18 義務的協調によるαの選択 46
4.19 利己的なエージェントによる p の選択 47
4.20 利己的なエージェントによるαの選択 47
4.21 ノード・エージェントの交換回数 (累計) 48
4.22 各ユーザが送信を完了したビット数 49
4.23 スループットの平均 (1 分毎の統計) 51
4.24 スループットの偏差 (1 分毎の統計) 52
4.25 スループットのゆらぎ (1 分毎の統計) 53

5.1 ユーザの異なる効用関数による選択 (短いインターバル) 56
5.2 効用関数の選択の効果 (短いインターバル) 57
5.3 η の選択 (高レートのグループ) 58
5.4 η の選択 (低レートのグループ) 59
5.5 効用関数の選択の効果 (長いインターバル) 59
5.6 価格レートを変動させた場合の資源競合 62

5.7 価格レート 62

A.1 SimFrame の構成要素 73
A.2 イベントとメソッド呼出し 73
A.3 イベントによる同期 75
A.4 イベントのスケジューリング 76
A.5 イベント・オブジェクト 77
A.6 ハッシュ・キュー 78
A.7 ブロードキャスト・キューと逆引きキュー 79
A.8 CSMA エージェントの状態遷移図 83

第 1 章

序論

会社	収益 (単位:百万ドル)	シェア (米国内)
UUNET	46.78	9.52 %
NETCOM	31.22	6.35 %
Sprint	29.94	5.74 %
PSI	22.92	4.67 %
Supernet	10.37	2.11 %
ANS	10.30	2.10 %
CERFnet	8.0	1.63 %
BBN	6.82	1.39 %
World	4.75	0.97 %
SURAnet	4.35	0.89 %
その他	320.59	64.63 %
合計	496.04	100 %

表 1.1: インターネット・プロバイダの市場規模 (1994 年)

1.1 イントロダクション

1970 年代中頃にパケット・ネットワークと, そのネットワーク間相互接続の研究のために設置された DARPA Internet は, 1980 年代に入って TCP/IP[Pos81] プロトコル・スイートを採用し, TCP/IP の優れた相互接続能力によって次々と接続するサイト数を増やし続け, その開始から 10 年足らずの間に, インターネットは, 米国内に留まらない, 主要な科学研究機関, 政府機関, 大学のネットワークを相互に接続する巨大なネットワークの集合体, connected Internet に成長した.

それまで研究者のためのネットワークとして捉えられていたインターネットは, 1990 年代に入り, 政府出資のプロジェクトの領域を越えて数多くの企業間のネットワーク接続に利用されるようになり, インターネットの商業利用という側面が急速に強まった. とりわけ, 電話回線を通じたインターネット接続をサービスするインターネット・プロバイダの出現と, Mosaic や Netscape といったユーザ・フレンドリなナビゲーション・ソフトウェアに支えられた World Wide Web の登場によって, ユーザは極めて容易にインターネット上の資源にアクセスできるようになり, インターネットの商業利用が本格化している. インターネットへの接続を提供するインターネット・プロバイダの市場は 1994 年の段階で 5 億ドルの規模まで成長し, CompuServe などのよう独自に情報を提供するプロバイダまで含めれば, 米国内だけで 10 億ドル以上の規模の市場がインターネットを中心に形成されている (表 1.1)[Bai95].

このように, インターネットが全てのコストを利用者から回収するという商業利用の側面を強めるにつれて, 価格設定やコスト回収をいかにして行なうべきかというモデル

の研究に対する重要性が増してきている [Cla95].

複数の価格設定による課金はコストを回収するためだけではなく, ユーザが異なるサービスを自由に選択するためにも効果的な手段である. 例えば, 値段が安く, ネットワーク容量も少ないサービスを選ぶ利用者もいれば, ネットワーク容量が多ければ値段が高くても良いという利用者もいるだろう. しかし, 現在のインターネットは, ただ一つのサービスクラス "best-effort" しか提供しておらず, ネットワーク上に生じる輻輳への対応に関しても, 課金などの経済的な手段によってではなく, TCP/IP を用いた技術的な方法で資源の分配を行なっている. そのため現状では, より多くの代価を支払ってもよいから, より良いサービスを得たい, といった要望を実現することは不可能であり, このまま商業インターネットが成熟したならば, 異なる代価に応じて異なるサービスを得ることができない現状を不満に感じるユーザが増えることが予想できる.

この問題を解決するためには, 必然的に, 現在のインターネットの資源を分配する機構の修正や, 機能の追加が必要となる. この問題に対し, 経済学の分野でも, 統計的な資源共有による低い処理コストや高い相互接続性, そして利用者が多いこと自体によるインターネットのメリットを維持しつつも, インターネットにおける, より洗練された課金体系のモデルを展開させることを目的とする Internet Economics と呼ばれる研究が行なわれている [McK95].

Internet Economics では, ネットワークの性能は, 技術的な尺度だけでなく, ユーザの満足度も考慮に入れた基準で評価されるべきだと考えられている. 具体的に言えば, ネットワークを利用するユーザに対して, 課金のバックプレッシャをかけることにより, ユーザの経済的判断によってユーザ自らの行動を決定させるべきだという立場である. そのエッセンスとして, ユーザの満足度, すなわち効用関数を考慮することは, 経済学において, 19 世紀の中期から既に標準的な事柄であったが, ネットワークの分野において, そのような視点からネットワーク資源の分配を分析する研究がなされるようになったのは, ごく最近のことである [Est90][Coc91].

効用関数を持つユーザが自己の効用を最大化しようと振舞った時, システムは一般にナッシュ均衡の状態に落ちつく.[1] ナッシュ均衡とは, 複数のプレーヤが参加するゲームを行なった時に, 各人が互いに相手方の選択を所与として自分の最適選択を行なった場合にシステムが収束する均衡点のことである. しかし, その均衡点におけるプレーヤの満足度は必ずしも最適な状態 (パレート最適) とは限らず, プレーヤが協調することによって, プレーヤ全員が, ナッシュ均衡における効用より高い効用を得られる場合がある. このように, 複数のプレーヤが参加するゲームにおいて, その局面を数理的に捉えるゲームの理論の発展に伴い, ネットワークにおける資源競合が見せる均衡状態の性質も, ゲーム理論的な見地から分析する研究が行なわれている.

Shenker らの研究 [She94] は, 複数の利己的に振舞うユーザを仮定し, これらのユーザ

[1] 戦略を択一的に選ぶ pure strategy の範囲では収束しない場合もあるが, 確率的に戦略を選択する mixed strategy の範囲では必ずナッシュ均衡が存在する.

がパケットのストリームを, 集中制御が可能なスイッチへ送信し続けるというモデルを設定している. そこで彼等は, ユーザに転送レートと遅延の関数による効用関数を与え, それらのユーザが自己の効用を最大化しようとした時の, 資源競合の均衡状態をゲーム理論的に分析している. 彼等の結論では, 利己的に振舞うユーザを仮定した場合でも, 集中制御が可能なスイッチにおけるサービスの方針を適切に選択することで, システムの状態を安定にすることが可能であることが示されている.

しかし, 現在広く利用されている Ethernet のような共有メディアにおいては, Shenker らが設定した集中制御を行なう管理実体は存在しないため, 彼等の結論が共有メディアを想定したときにも, そのまま成り立つとは限らない. そこで本研究では彼等が行なった研究を基礎として, 共有メディアにおける利己的なユーザの振舞いを観察することにした.[2]

本稿に影響を与えた別の研究として, マルチエージェントシステムにおける資源競合の理論的解析が挙げられる. 例えば, Huberman らは, 情報の遅延などに起因するシステムの不確実性, および, 自己の効用を最適化しようとする利己的なエージェントをマルチエージェントシステムの本質的な要素として捉えた上で, 本来ならばすみやかにナッシュ均衡に収束するはずのシステムが, 情報の遅延により, 非線型な振動やカオス的な振舞いを呈することを結論として示している [Hub88b]. Huberman らは, これらのエージェントの集団によって形成される人工的な生態系を Computational Ecosystem と呼んでいる.

これらの研究は, 共通して, ユーザが必ずしも協調的とは仮定できない環境における資源競合を論じたものであり, そのような議論は, 将来のオープンシステム [Tok91][所95] における資源分配を考える上で必要不可欠な問題であると思われる.

1.2 目的

本稿では, 利己的なエージェントによる資源競合という観点から, 共有メディア型ネットワークをテストベッドとして, 複数のエージェントによって構成される人工生態系のモデル化を行なう. 本モデルにおいて, エージェントは遺伝的に獲得した戦略に従ってサービスを提供する. エージェントは特定のサービスのプロバイダであり, かつ, 別のサービスのユーザでもありうる. 各エージェントは, 利用するサービスの性能とコストを観測し, 性能とコストの関数で表される独自の効用関数により効用を計算する. エージェントは効用の時間的変化に対して独自の判断を下すことによってプロバイダを選択淘汰し, 試行錯誤を通じて自己の効用を最大化すべく振舞う.

本稿では, 上記のモデルを実行するシミュレータを設計・実装し, Ethernet に類似した共有メディアにおいて複数のエージェントが利己的に振舞う環境での, ネットワーク・システム全体の進化的な挙動を観察する.

本稿では以下の点に焦点を当ててシミュレーション結果の評価を行なう:

[2] Shenker らの研究は, 本研究の出発点となっており, 多くの点で本稿の基礎となる研究と言える.

- コストを考慮することの意義
  コストを考慮するエージェントの振舞いとコストを考慮しないエージェントの振舞いの違いは, 一体如何なるものであろうか?
- 利己的なエージェントの振舞い
  利己的に行動することが許されるエージェントは, 果して, 全く非協調的に振舞うのだろうか? また, 遅延を含む情報に対しては, どのように反応するだろうか?
- 進化するネットワーク
  ネットワークを通過するパケットの数は膨大である. そのため, システムの要素に遺伝的操作を加え, 選択淘汰させる対象として極めてふさわしいことが予想される. 実際に観測される進化的な挙動は如何なるものか?

人工生態系は計算システムの身体性をモデル化した抽象化概念であり, 本稿のシミュレーション・モデルも, 人工生態系の観点から設計される. 人工生態系内部で活動するエージェントは時間の概念を持ち, それらのエージェントが共有する情報は必然的な遅延を伴ってエージェント間を伝播する. そのようなモデルに従って, 厳密な設定の下でシミュレーションを行なうことは, これまで言及されることの少なかった, エージェントがサービスを実行するためのコストを定量的に見積るためにも重要であると考えられる.

Huberman らのモデルにおいて, エージェントの戦略は既知のものであった. 一方, 本稿では, エージェントの戦略は遺伝的に獲得されてゆくものと仮定している. エージェントが人工環境によって選択淘汰されることにより, システム自体が進化を遂げる進化的人工生態系の概念は人工生命の分野でも研究が行なわれているが, より具体的な例として現実的な共有メディア型ネットワークをモデル化し, シミュレーションを行なうことに本稿の意義がある.

1.3 本論文の構成

本論文の構成は以下の通りである. まず, 2 章において, 利己的なエージェントによる資源共有という観点から関連研究のサーベイを行ない, 3 章では, それに基づいて本研究に必要なシミュレーション・モデルを構築する. 4 章では, まず, シミュレータの動作を検証するために予備シミュレーションを行ない, 次に実際のシミュレーションの結果を示しながら評価を行なってゆく. 5 章では, 進行中のシミュレーションの中間結果を元に今後の課題について議論し, 最後に, 6 章において結論を述べる.

第 2 章

関連研究

本章では, 利己的なエージェントによる資源共有という観点から関連研究のサーベイを行ない, 本研究のシミュレーション・モデルに必要な性質を明らかにする.

2.1 Internet Economics

2.1.1 ユーザの満足度

Internet Economics が提案する基本的なコンセプトに, ネットワークの性能に対する新しい考え方がある. ネットワークが良いか悪いかを判断するのは人間であり, ネットワークの性能は, 従来のような, パケットの平均遅延時間や平均ロス率といった技術的な尺度だけではなく, 総合的なユーザの満足度も考慮して計られなければならない. これは決して従来の尺度を否定するものではなく, むしろ従来の QoS (Quality of Service) の考えを包含している.

従来の研究者たちは, 多様な QoS 要求をネットワークの操作に反映するために, アプリケーションやトラフィックの性質に従って利用法をサービスのクラスに分割してきた [Coc91]. 例えば, リアルタイム・ビデオ, リアルタイム・オーディオ, オンライン端末, オフライン・データ転送などに分類し, それぞれのサービスクラスをユーザが選択, 利用するという形態である. しかし, 同じ QoS に対するユーザの満足度は, ユーザごとに異なることもあれば, 同じユーザですら時間によって異なることもある. そのため, QoS に対して予めプライオリティを設定するなどの固定的なアプローチでは, ユーザの満足度の変化には対応できない.

また, 現状では, 個々のユーザの QoS 要求は, ある時間間隔, あるいは他のユーザの要求との間で平均化され, その平均化された値によってネットワークの制御が行なわれる. 実際のデータ転送は TCP/IP によって行なわれる一方で, ユーザは TCP のフィードバック・ループから切り離されているため, QoS をより適切に実現するための判断を直接下す方法がない. これに対し, Internet Economics は, ユーザがネットワークのフィードバック・ループの一部としてユーザ指向の性能評価を行なうことを提案している [Mac95].

2.1.2 効率とは何か?

従来の枠組みでは, ユーザがネットワークの性能を評価する基準となるものは, それを利用する際の効率である. 例えば RSVP[Zha93] や ST-II[Top90] などのインターネットにおける資源予約プロトコルの多くは, ネットワーク資源を "効率的" に利用することを目的として設計されてきた. ここで言う効率とは, 技術的な効率のことであり, 経済的な効率は, それに付随して自然と達成されるものと考えられているか, あるいは, 全く考慮されていない. しかし, Internet Economics では両者を明確に区別している.

- ネットワークの効率:
  バンド幅やバッファ・スペースなどの, ネットワーク資源の利用に関する尺度.
- 経済的な効率:
  ユーザがネットワーク・サービスに対して下す, 相対的な価値評価に関する尺度.

もし, 目的のサービスのレベルを保ちつつ, そのサービスのために消費される資源の量が最小化できれば, ネットワークの効率は良いと言える. 一方, 限られた資源を分配する時に, その資源に低い価値評価 (支払い) をするユーザが, 高い価値評価をするユーザよりも多くの資源を得ることがない場合, それは経済的に効率的と言える.

経済的な効率を考慮することは, ユーザの満足度に対して負のフィードバックをかけることに等しい. ユーザは, ある QoS 要求の結果として得られるネットワークの性能に対して価値評価を行ない, 支払うコストとの間でバランスを考えなくてはならなくなる. 本稿では性能とコストの関数を性能/コスト比として抽象化し, これをユーザの満足度の尺度と考えることにする.

2.1.3 輻輳制御と Responsive Pricing

フィードバックはネットワークの効率を上げるための一般的な方法である. 最も長い時間スケールにおいては, ユーザは, ネットワーク・プロバイダの課金体系や実際の利用状況に基づいて特定のプロバイダと契約するか否かを決定する. セッションを張ろうとする段階では, ネットワークの負荷がフィードバックとなって, 特定の時間に負荷が高くなることを知っているユーザが, 低負荷の時間帯を選んでネットワークを利用するという行動が考えられる. これらの比較的長い時間スケールにおいては, ユーザの反応はしばしば経済的な理由に基づいて決定される.

一方, 短い時間スケールとしてコネクション・レベルでは, ユーザはネットワークからのフィードバックに反応して, 入力するトラフィックや QoS 要求をアダプティブに調節することができる. 例えば, TCP は slow-start や fast-retransmit[Jac88] によって, 入力レートをその時点で利用可能なバンド幅に合わせることができ, ATM でも ABR (Available Bit Rate) Service を利用するユーザは, ATM ネットワークが発するフィードバックに従うことで一定以下のセル棄却率を保証される.

しかし, 必ずしも全てのユーザが協調的だとは限らない公衆ネットワークにおいては, TCP のような伝統的なフィードバックはロバストにユーザの行動を制御できるとは言いきれない. フィードバック・シグナルを無視するプログラムは比較的簡単に書くことができるし, そもそもフィードバック・ループを持たない UDP[Pos80] を使用してリアルタイムデータを転送するアプリケーションは現実に存在している. オープンシステムにおいては, そのような『抜け駆け的行為』に走る利己的なユーザがいることを無視するこ

とは危険であるため, 仮に利己的な行動をとるユーザがいる場合でも被害を最小限に抑えることができるようなロバストなシステムを設計することは重要である.

ネットワークの負荷に応じた価格設定は, 利己的に振舞うユーザに対しても有効なフィードバックになると考えられる [Est90]. 価格に敏感なユーザは経済的効率を考慮してその時点の価格に応じてトラフィックを投入するか否かを決定する. ネットワーク・サービスに魅力を感じるユーザは高い代価を払ってでもトラフィックを流しつづけ, そうでないユーザは価格が下るのを待つことになる. ネットワークの負荷が低下した時には, 価格はほとんどゼロに近くなり, 全てのユーザがデータを入力できるようになる.

このように, ネットワークの状態に応じて価格が変化する課金体系は, ネットワークの効率と経済的な効率を共に増大させる潜在的能力を持っている. Internet Economics では, ネットワークが QoS を守れなくなる程の輻輳状態に陥った時にのみ, 輻輳の程度に応じて価格を変化させる価格設定を **Responsive Pricing** と呼び, Responsive Pricing に基づいてインターネット資源の分配を行なうことを提案している [Mac95].

Internet Economics の議論で使われる課金の概念は, 必ずしもドルやセントといった現実的な金銭に適用することを意図したものではない. 適切なコストと価値評価の関数が与えられれば, 実際の金銭に捕われずに Responsive Pricing の機構をネットワークに適用することができる.

2.1.4 課金方法のスキーム

課金を実行するためには, それに伴うオーバヘッドを最小限に抑える努力をする必要がある. 電話会社の料金回収に費やされるコストは, 概算で料金の 50%に相当することが Gillett らによって報告されており [Bai95], これをインターネットに適用することは到底受け入れられない. しかし, Responsive Pricing の目的は輻輳が引き起こすコストへの課金であるため, 方法によっては低いオーバヘッドで Responsive Pricing を実装することができる. その一つとして, セリ (bid) を基本とする Smart market pricing が提案されている (図 2.1)[Mac95].

Smart market のアプローチでは, パケットを送信しようとするユーザは, ある期間に自分が払ってもよいと思う代価を各データグラムのヘッダ部分に書き込むことにする. そして, そのインターバルの間, ネットワーク上のルータは, 到着するパケットに記入されたセリの『付け値』をソートして, その順序に特定の性能を維持できる限りの数のパケットを決定し, 輻輳のコストとして, それらの付け値における最低額をユーザに請求する.

マルチエージェント・システムの分野の研究でも, このようなオークションの機構を利用した資源共有が有効であることが示されている. Huberman らは, ビルの空調制御のために, オークションを使用して冷気や暖気を売り買いするエージェントのモデルを設計し, そのモデルを現実のビルの空調管理に応用することにより, 実際に無駄の少ない公平なエネルギーの分配が行なわれることを実証した [Hub95]. この例において, オークションが

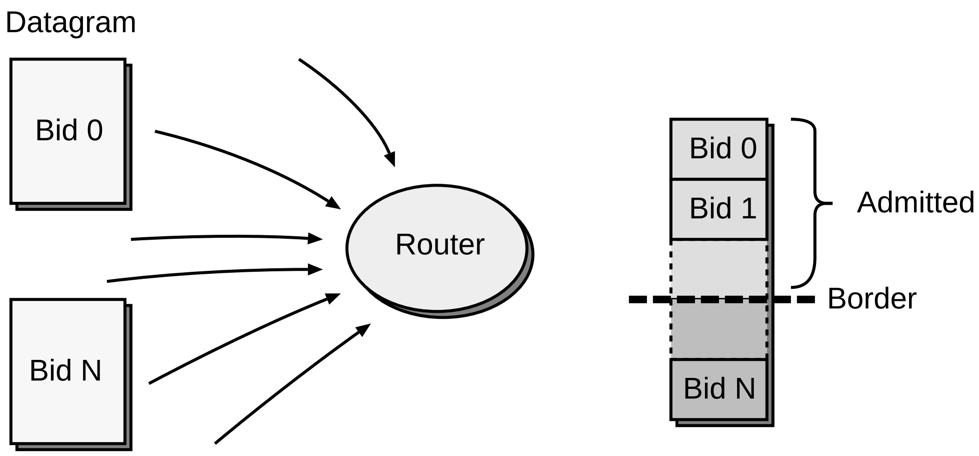

図 2.1: スマート・マーケット

実際の貨幣を用いて行なわれる金銭的取り引きではなく, 資源を有効に分配するための市場メカニズムの一つとして捉えられていることは, ネットワークにおける Responsive Pricing にも共通する考え方である. ビルと同様に, 一つの組織の中で閉じたネットワークであっても, ゲームとしての市場原理を利用することによって, 技術的効率と経済的効率を共に向上させる意義があると思われる.

2.2 利己的なユーザによる資源共有

ユーザの満足度を考慮した資源共有に関する興味深い研究に, 利己的なユーザによるネットワーク資源の分配のモデルに関する Shenker らによる研究がある [She94]. 本節では, 利己的なユーザによる資源共有という観点から, 具体的な資源共有のモデルを調べてゆくことにする.

2.2.1 非協調的アプローチによる輻輳制御

国家が巨額の費用を投入しても一向に巷の交通渋滞が解消されないのに似て, コンピュータ・ネットワークにおける輻輳制御は長年に渡って研究者の興味の対象であった. 早期に提案された輻輳制御メカニズムの大部分は, 特定のフロー制御アルゴリズムを実装したエンド・ホストにおけるユーザが協調的に振舞うことを仮定している. このアプローチでは, システムの設計者は, システムが高い利用率や低い遅延を達成できるようにアルゴリズムを設計し, そして, ユーザは合意されたアルゴリズムに従わなければいけない.

それに対し, そのような合意されたアルゴリズムを設計することは不可能であり, ユーザは個人的な興味に基づいて『利己的』に振舞うと仮定するアプローチもある. Shenker らは, ネットワーク資源の共有を考えるにあたって, まず, 次のような仮定を設けた:

- ユーザは利己的であり, 独立して行動する.
- 集中制御はネットワークのスイッチにおいてのみ行なわれる.

一つのスイッチに対して複数の利己的なユーザがデータを送信し続けるというネットワーク (図 2.2) において, ユーザは自分の利益のみを考えて振舞い, それに対してスイッチはサービス方針を決定し, その方針に基づき集中制御を行なう. ここで言う集中制御とは特定のスイッチにおけるサービスの実行方法に関する取り決めに言及するものであり, TCP が仮定するような全世界的に合意されるはずの (そして合意される見込みのない) 取り決めを意味するものではない. このアプローチにおけるシステム設計者の役割は, ユーザが利己的に振舞う環境においても, ネットワークが良いサービスを提供するために, スイッチにおいて実行されるサービス方針を設計することである.

なぜ我々は利己的に振舞うユーザを仮定しなければいけないのか? Shenker らの研究の結論では, 非協調的アルゴリズムを用いた場合, ゲーム理論的に言えばパレート最適なナッシュ均衡は得られない. 一方, 協調的アルゴリズムを用いた場合, 極端な話, 完全スケジューリングをしてしまえば, 最高の効率を達成することができるのである. それならば, 現状のまま, ユーザの協調性を前提として, 協調的アルゴリズムを用いた方が良いのではないだろうか?

しかし, 協調的アプローチを用いるためには, ユーザ全員が協調したいと望んでいる必要があるだけでなく, どのように振舞うことが協調的なのかという点に対する合意も必

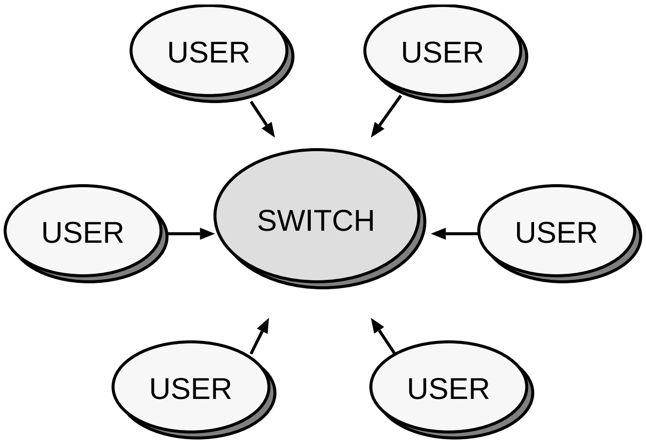

図 2.2: スイッチを中央に置いたネットワーク

要になる．それを定義するはずの普遍的に合意されるフロー制御アルゴリズムの実現にはいくつかの問題点があり，それらは非協調的アプローチの欠点，すなわちパレート最適を達成できない点と十分に比較検討されるべき問題である [She94].

- ア・プリオリな効用関数
  自己最適化によらない最適化は，未体験の事象に対する既知の効用関数を必要とする．
- 柔軟性の低下
  ネットワークと計算機，そしてアプリケーションは常に変化している．変化しない普遍的なアルゴリズムを採用することはネットワークの柔軟性を低下させる．
- 利己的なユーザに対する非力さ
  ユーザの協調性を前提とするアルゴリズムは，単発的に発生するユーザの抜け駆け的行為に対して無力である．

『非協調的』という言葉は，必ずしも『悪意を持つ』という意味のみに限定されるわけではない．ユーザは単に自分自身の満足を向上させるために利己的に振舞えばよい．始めは，せいぜい『しかたない現実』としか思えない非協調的アプローチも，頻繁に変化する巨大で多様な将来のネットワークにおいては，良い性能を発揮するための最善の方法であると Shenker らは考えている．

2.2.2 輻輳制御のゲーム理論的分析

ここで，非協調的アプローチに基づく輻輳制御をゲーム理論的な見地から分析する．図 2.2で示したモデルの構成要素は，集中制御のスイッチと利己的に振る舞うユーザである．

ユーザは一斉にパケットのストリームをスイッチへ送り始め, パケットの到着率がスイッチの処理率を越えた時, スイッチは輻輳状態に陥る. スイッチはある決められた方針に従ってネットワーク資源をユーザに分配することができる. ユーザは自分が投入するレートに対する遅延を観測し, それぞれのユーザが持つ効用関数を最大化する方向でレートを変動させる. スイッチにおける輻輳制御は次のような数学的モデルで記述することができる.

$user_i$は Poisson レート r_iでスイッチにストリームを送り, スイッチはユーザ毎にキューを配置する. このキューで待つパケットの平均長は輻輳状態の指標となり, $c_i = C(r_i)$ で表される. ここで言及する “利己的なユーザ” とは, ユーザが独自の効用関数 $U_i(r_i, c_i)$ を持ち, 独自のフロー制御により r_iを変化させて $U_i(r_i, c_i)$ を最適化することを指す. また, $U_i(r_i, c_i)$ は r_iと c_iに関して共に単調であると仮定する.

以上のモデルにおいて, 集中管理のスイッチが行なう輻輳の分配方法, すなわち $C(\vec{r})$ をどのように選ぶかが問題になるが, ここでは *Fair Share* アルゴリズム:

$$\begin{aligned} C_k^{FS}(\vec{r}) &= C_{k-1}^{FS}(\vec{r}) + (g((n-k+1)r_k + r_{k-1} + \ldots + r_1) \\ &\quad - g((n-k+2)r_{k-1} + r_{k-2} + \ldots + r_1))/(n-k+1) \end{aligned}$$

(ここで, n はユーザ数. k は r_iを昇順にソートした時の i の順位. また, $g(x) = x/(1-x)$.) を選択することで, 次のようないくつかの特性が得られることが示されている.

- 唯一の Nash 均衡を持つ.
  $\vec{r}^{Nash}$が Nash 均衡とは:
  $U_i(r_i^{Nash}, C_i(\vec{r}^{Nash})) \geq U_i(\hat{r}_i, C_i(\vec{r}^{Nash}|^i\hat{r}_i))\ for\ all\ \hat{r}_i\ and\ i.$
  ($\vec{r}|^i\hat{r}_i$は$\vec{r}$の r_iを$\hat{r}_i$で置き換えた値.)

- Nash 均衡におけるアロケーションは平等である.
  Shenker らは平等を Unilaterally envy-free[1] (表 2.1) によって定義している.

- この均衡への収束はロバストで, しかも収束は速い.

- 均衡外でも, ユーザは他のユーザから不当な害を受けない.

- しかし, Nash 均衡が Pareto 最適であることを保証することはできない.
  $(\vec{r}, \vec{c})$ が Pareto 最適とは:
  $U_i(r_i, c_i) \leq U_i(\bar{r}_i, \bar{c}_i)\ for\ all\ i$
  $U_i(r_i, c_i) < U_i(\bar{r}_i, \bar{c}_i)\ for\ at\ least\ one\ i$
  という可能なアロケーション $(\vec{\bar{r}}, \vec{\bar{c}})$ が存在しない.

[1]直観的な意味は “どのユーザも他のユーザが得たレートと遅延のアロケーションを妬まない” である.

もし次式が成り立つならば, そのアロケーションは Unilaterally envy-free:
$\{U_i(r_i, C_i(\vec{r})) \geq U_i(\hat{r}_i, C_i(\vec{r}\|^i\hat{r}_i)) \ for\ all\ \hat{r}_i\} \Rightarrow \{U_i(r_i, C_i(\vec{r})) \geq U_i(r_j, C_j(\vec{r})) \ for\ all\ j\}$

表 2.1: Unilaterally envy-free による平等の定義

以上の結果について重要な点は, 利己的に振る舞うユーザを仮定した時でも, スイッチの制御方法を巧妙に選択することにより, 均衡点におけるサービスが効率的で公平になりうるということである. しかし, 前述の通り, その均衡点がパレート最適であることは結論できない.

2.2.3 平等とは何か?

ここで再び Internet Economics の立場で,『平等』の概念について検討することにする. Responsive Pricing に対する異論の一つとして, 次のような意見がある.『なぜ課金を含まない手法では不十分なのか? 適切な管理方法を用いれば, いくつかの平等の概念をユーザに押し付けることができる. 例えば, トラフィックにプライオリティを付けて, 必要に応じてプライオリティを選択させれば良いのではないか?』

しかし, 一体, 誰が平等の概念を定義するのだろうか? むろん, ネットワーク管理者はできるだろうが, ユーザ指向の満足度を考慮すれば, 結局, 平等とはユーザの総体が決定すべきものなのである. 恐らく, ほとんど大部分の人は, e-mail よりは, 緊急に行なう必要のあるテレメディシンによる心臓手術のセッションに高いプライオリティを設定することに同意することができるだろうが, DOOM や DESCENT といった通信ゲームと e-mail を比較した時は, どちらに高いプライオリティを与えるべきなのだろうか? アプリケーションに与えるプライオリティを変えられるとして, 自分のアプリケーションに対して最高のプライオリティを与えようとしない人がいると考えることができるだろうか? それが『不適切』なプライオリティだと判断されたとして, では誰が『適切』なプライオリティを定義し直すことができるだろうか? そこで結局, 競合を解決するためにプライオリティに値段を付けるという考え方 [Coc91] が, Responsive Pricing のエッセンスである.

プライオリティに限らず, 多くの管理ポリシーに対して, 同様の議論が展開できる. 例えば, 10Mbps のリンクに n 本のコネクションを張る場合, それぞれのコネクションに対して 10Mbps/n だけの資源を割り振れば平等なのだろうか? 一見 "平等" に見えるスキームにおいても, 狡猾なユーザは, この場合, 複数のコネクションを張ることによって, 他のユーザよりも多くの資源を得ることができるだろう.

Shenker らのモデルにおける平等の概念が, 上記のような資源の分配量だけに注目した

議論ではなく, ゲーム理論的な見地から Unilaterally envy-free によって定義されていることは興味深い. この定義はユーザの満足度, すなわち効用関数を主眼としており, その効用関数がコストによるバックプレッシャを受けていたとすれば, 平等であるか否かがユーザの総体によってリーズナブルに決定されることを示唆している. この解析結果のインパクトは, 仮に非現実的な効用関数を持つ (輻輳が起っても送信レートを上げ続けるような) ユーザがいたとしても, スイッチのサービス方針さえ適切に設計すれば, そのようなユーザから理性的なユーザを守ることができ, 結果としてシステムがロバストである点にある. ここで述べた方法以外にも, 平等の概念には多くの異なった定義をすることも可能であるが, 本稿では Shenker らが選択した Unilaterally envy-free の考えをを平等の定義とする.

2.2.4 CSMA とゲーム理論

Shenker らのモデルの構成要素は利己的なユーザと *Fair Share* アルゴリズムを実装した一つの集中制御可能なスイッチだった. これらの仮定は現実のネットワークに直接的に対応するものであり, 少なくとも一つの現実のネットワークを観測した結果は, おおよそ, そのゲーム理論的解析に一致すると思われる. しかし, 実際のネットワークは複数のスイッチか, あるいはルータによって構成されているため, 複数のエンド–エンドのユーザの組のそれぞれが, 異なる経路を用いて通信を行なうことが考えられる. そのため, Shenker らの解析をそのままネットワーク間接続されたネットワークに拡大適用することは難しく, 特にユーザ毎に異なるルータを利用することになるため, 性能と輻輳の関数を効用関数としている限り, それに基づいて平等を議論することは難しい. ただし, それ以外の大部分の論点に関しては, 一つのスイッチの理論をインターネットワークに適用できることが分かっている.

それでもなお, 上記のモデルをインターネットに適用するにあたって一つの問題点が残っている. それは, Ethernet[Dig80] のような, 集中制御が不可能なメディアの存在である. Ethernet の実体は CSMA/CD[Kle75] による共有バスであり, そこにはサービスの方針を決定, 実行するような管理実体はない. そのため Ethernet の使用方針は, 共有メディアに接続されたプロセッサに実装された IEEE 802.3 標準に従うというユーザ間の合意によって決定されていると考えることができる. [2] つまり, 共有メディア型のネットワークには, サービスの提供者と利用者という明確な区別がなく, メディアを共有するユーザ間の相互作用によって各ユーザの得るサービスの質が変化し得るという点で, 集中制御のモデルとは大きく異なった側面を持っている.

CSMA (Carrier Sense Multiple Access) は OSI 参照モデル [Tan89] におけるメディア・アクセス副層に相当する, 複数の送信者が一つのメディアを共有するための基本的なメ

[2]近年, Ethernet の拡張として開発された Switched Ethernet[Bre95] や Fast Ethernet[Joh96] でも, それは同様である.

ディア・アクセス方式である. Ethernet は, その変種の CSMA/CD (CSMA with Collision Detection) をアクセス方式として採用している. CSMA の基本は, 送信者が送信前にチャネルを観測し, チャネルが使用中であればキャリアが止まるまで待機することである. さらに, Ethernet のように, 送信中にパケットが衝突したことを検出できる場合には, 衝突の発生に伴い送信者は, ただちに送信を停止し, キャリアの停止を待って再びパケットの送信を試みるという動作を繰り返す.

パケットが衝突する本質的な原因は, キャリアの伝播に物理的な遅延があることである. 全てのユーザが確実にキャリアの停止を待って送信を行なった場合でも, パケットの衝突が発生する確率はゼロにはならない. なぜなら, あるユーザがパケットの送信を始めたとしても, それが別のユーザに検知されるまでは, そのユーザにとっては, チャネルは未使用状態に見えるからである. Ethernet のキャリアは光の速度の約 80%のスピードで伝播するが, ネットワークの物理的な大きさを考慮して, 規格上, 最大 23.2μ秒もの伝播遅延を許しており, この値は衝突を発生させるのに十分な大きさである. それが光の速度で伝播するメディアであっても, そこに伝播遅延がある限り, CSMA 共有メディアにおけるパケットの衝突は避けられないと言える. 伝播遅延の存在によって各ユーザは過去の情報に基づいて, 送信するか否かの判断を行なわなければならない. 遅延の大きさは $a =$ (伝播遅延時間)/(パケットの時間長) で規格化されるが, a が大きくなると CSMA の特性は急激に劣化する.

CSMA は, 一般化すると, non-persistent CSMA と p-persistent CSMA (pp-CSMA) と呼ばれるメディア・アクセス方式に分類できる. 前者は, チャネルが空いていれば送信し, 使用中であれば一定時間待ってチャネルを観測し, 空いていれば送信し, 空いていなければ同じことを繰り返す. この方式は送信に固執しないため, non-persistent CSMA と呼ばれる. 一方, 後者は, CSMA における待機時間を確率変数 p のばらつきで以って決定する CSMA 方式で, チャネルが使用中の場合にはチャネルが空くまで観測を継続し, 空いた時点で確率 p で送信動作に移る. この方式は送信に固執するため, p-persistent CSMA と呼ばれる. もし確率 $(1 - p)$ で送信しない場合は, non-persistent CSMA のように一定時間だけ待って, 再び同じ動作を繰り返す. この一定時間はスロットと呼ばれ, 待機時間をスロットで決定する CSMA はスロット化 CSMA と呼ばれる. これに対し, 待機時間を連続関数で決定することも可能であり, その場合, スロットは用いられない. pp-CSMA は, 変数 p が大きいと送信しやすくなる一方でパケットの衝突確率は高くなり, 逆に p が小さい場合, 衝突確率が下がるかわりに過度に送信を控えるようになり, 性能が落ちるという性質を持つ [Kle75].

Ethernet の CSMA 方式は, 正確に言うと, スロット化しない 1-persistent CSMA/CD で, さらに, 衝突が発生した時は待機時間をランダムな要素を加えて 2 倍してゆくバックオフ (binary exponential backoff) の機構を持つ. 実質的にキャリアが止った直後に送信動作を開始する 1-persistent という方式を採用していても, このバックオフの機構により Ethernet は高いスループットを達成することができる [Bog88].

Ethernet がパケット衝突後のユーザ間の調停に, 一方的な送信レートの低下を強要するバックオフの機構を用いることは, 丁度, インターネットにおいて TCP が slow-start を用いて一方的に送信レートを下げることによって, バンド幅のアダプテーションを行なうことに類似している. 前述の通り, TCP はユーザの協調性を前提に設計されたフロー制御プロトコルであり, Ethernet も同様に協調的なアプローチによって成功したプロトコルの代表例である. 非協調的アプローチによる輻輳制御の観点から言えば, TCP のレベルと同様に Ethernet のレベルでも, バックオフを無視して利己的に振舞うユーザの存在の可能性を無視するわけにはいかない. しかも, この場合, スイッチのように集中管理が可能な管理実体が存在しないため, 利己的なユーザの振舞いを制御することは, それほど容易ではないことが想像できる. そこで本研究では, 手始めとして, Ethernet と同等の共有メディアにおける利己的なユーザの振舞いをシミュレーションを通じて観察することにする.

2.3 人工生態系と遺伝的アルゴリズム

前節では, 輻輳制御をゲーム理論的に分析する際に, 利己的なユーザは自分の効用を最大化する方向に振舞う, という仮定を行なった. しかし, ユーザは一体どのようにして, それを実行したらよいのだろうか? 実のところ, ゲーム理論では, 実際に個別のユーザが効用を最大化する方法, それ自体については議論の対象外としている. ゲーム理論に必要なのは,『ユーザは自分の効用を最大化するために頑張るものである』, という仮定であり, その仮定に基づいてナッシュ均衡の周辺の状態や, その均衡への収束, ロバスト性などが分析できれば, 実際に効用を最大化する方法を議論する必要はないのである.

しかし, 工学的な立場から言えば, 解析的に解くことが困難な場合においても, ユーザが何らかの方法で効用を最大化することができる手法を準備することも, 極めて重要である. 本節では, ユーザが自分の振舞いを決定する手法を, Computational Ecosystem[Hub88a], 遺伝的アルゴリズム (GA)[北野93] を通じて考察してゆく.

2.3.1 Computational Ecosystem

集中制御が存在しないネットワークにおける資源分配の問題は, マルチエージェントシステムにおける資源競合という視点からも分析することができる. Huberman らは, 非同期かつ局所的に制御されたエージェントのモデルを用いて, 効用関数を持つエージェントが自分の効用を最大化するために利己的に振舞う環境における資源競合のダイナミクスを理論的に分析している [Hub88b].

マルチエージェントシステムを特徴づける本質的な要素は, 次の二点:

- 不確実性: エージェントはシステム全体に関する完全な情報は持ち得ない.

- 戦略: エージェントは不確実な情報を元にして, 自分の振舞いを決定する.

であり, これらの要素は本稿で具体的に検討するネットワークシステムにも共通する事柄である. Huberman らのモデルにおいて, 複数のエージェントは 2 種類の資源をめぐって資源競合を行なう. 資源 rには, それを利用するエージェントの比率 fに対する効用が定められている:

$$G_r(f) = A_r + B_r f - C_r f^2$$

この効用関数は Shenker らのモデルで言う効用とは意味が異なり,『資源が持つ効用 A_r』,『エージェントが使用することによって生れる効用 $B_r f$』,『エージェントが使用することによって失なわれる効用 $C_r f^2$』の, 3 つの項を用いて設計者がア・プリオリに決定した (この世界では)"正しい", "普遍的な" 効用である. 各エージェントは報酬として$\hat{G_r(f)}$を知覚する. システムには不確実性があるという前提から, 各エージェントが知覚する$\hat{G_r(f)}$は標準偏差σの正規分布の誤差をともなう $G_r(f)$ として定義する. 各エージェントは, あるポワソン率に従う指数分布の時間間隔で 2 つの資源の報酬$\hat{G}_1, \hat{G}_2$を評価し, 大きい報酬を与える資源を選択する. その情報には遅延τを設定し, エージェントは過去の情報に基づいて選択を行なうことにする. ただし, システムの不確実性という視点からだけではなく, 時定数τはエージェントの戦略を支配するパラメータとなる.

エージェントが知覚する報酬に不確実性がない時 ($\sigma = 0$), エージェントはシステムの状態を完全に把握しているため, システムは すみやかにナッシュ均衡へ収束する. しかし, 不確実性が増すにつれて均衡はパレート最適な状態から遠ざかり, 各資源を利用するエージェントの割合は等しくなってゆく. 情報の伝播に遅延がある場合, このような均衡状態は非線型振動やカオスへと陥り, 結果としてシステムの性能は低下する.

そのような振動は Hogg と Huberman らによって提案された機構を用いることで平衡状態へと引き戻すことができる [Hog91]. それは, エージェントの戦略に適応性を与えるために, エージェントが異なる戦略を選択することができる機構を設定することである. システムのダイナミクスの進展と共に, エージェントは自分の戦略を最適化しようと試みる. これらのエージェントの集団は人工的な生態系を形成し, Huberman らは, それを Computational Ecology, あるいは, Computational Ecosystem と呼んでいる.

2.3.2 戦略の遺伝的適応

Huberman らは, 各エージェントが最適な戦略に関する前提知識を持たないことを強調しているが, 実際のシミュレーションにおいては, 設計者がいくつかの戦略をア・プリオリに用意している点に注目したい. その戦略自体がア・プリオリに決定できない場合は, エージェントは, 一体, いかなる方法で戦略を最適化させればよいのだろうか? その一つの方法として, ここでは遺伝的アルゴリズムを利用した遺伝的適応について考察する.

遺伝的アルゴリズム (GA) は, 自然界で行なわれている生命進化の原理のエッセンスを抽出し, 確率的探索, 学習, 最適化の一手法として工学的に応用したアルゴリズムである.

現実世界には, 解法の確立されていない問題や, 解法が分かっていても, その計算量が膨大で現実的には解くことのできない問題が多数存在する. しかし,『最適解は分からないが解の評価は可能である』という場合には, それらの問題に対して GA を適用することにより, リーズナブルな計算量で精度のよい近似解を求めることができる. この点が GA を工学的に応用する上での最大の利点である.

GA では, 暫定的な解を表す『遺伝子』に対して, その解に対する評価の度合を示す適応度を定義する. そして, 遺伝子に交叉や突然変異などの遺伝的操作を加え, それを適応度によって選択淘汰することによって, 暫定解を徐々に求めるべき近似解に近づけてゆくという操作を繰り返す.

人工生態系を形成するエージェントの戦略に対しても, 適応度を定義することで GA, あるいは遺伝的プログラミング (GP)[Koz92][Koz94] を適用することができる. 実際,『囚人のジレンマ』のゲームや追跡ゲーム [Ben85] における戦略を GA, GP で進化させることによって, それらのエージェントが取るべき最適戦略を求めようという研究がなされている [Dav87].

2.3.3 進化するネットワーク

GA の研究者たちが考える『戦略の進化』は, Huberman らが提唱した Computational Ecosystem において必要とされる『戦略の進化』とは, 必ずしも一致していないように思える. なぜなら, GA の立場では, 戦略の適応度は『設計者』が決める事項であるのに対し, Huberman のモデルでは, それはむしろ『エージェント』の視点で決定されるべきものだからである. 例えば, 囚人のジレンマの最適解を求めたいと思うのは人間 (設計者) であり, 設計者は,『最も利得の高くなる戦略』というような特定の戦略を求めるために, ゲーム世界の人工生態系を利用して, その解を得ようとする. このような,『設計者主導の進化』に対し, Huberman らの人工生態系で必要とされるのは, エージェントの自律的な判断によって選択淘汰が行なわれる『エージェント指向の進化』ではないかと考えられる.

この点に関しては, 従来の GA よりも, さらに生物学的な立場から人工生態系を考えた人工生命 [Lan89][Lan95] からのアプローチが興味深い. 生物界ではア・プリオリに定義された評価関数が与えらえることはあり得ないため, 人工生命の世界のエージェントたちは, 遺伝的に獲得した評価関数に基づいて行動の選択を行ない, それらが相互作用しあうことによってシステムが進化してゆくことになる.

人工生命の立場に立てば, ネットワークシステムにおいても, 同様にシステムの進化を考えることができる. 簡単に言えば (そして夢のある言い方をすれば),『次の日, 目が覚めたら, 昨日使っていた TCP が TCP' に進化していた』, といったネットワークである.

しかし, 人工生命の手法を延長しただけでは進化するネットワークは生まれないだろう. なぜなら, 人工生命の世界のエージェントたちは, 言うなれば, 野性化したエージェ

ントであって, 人間世界の事象とは無関係の場所で繁栄を築いたり, 衰退したりするものだからである. エージェントたちには, それなりの存在理由があるのだろうが, 人間の制御できないところで勝手に進化するエージェントは, 人間の立場から見れば, (科学的な興味は引くとしても) それほど工学的な価値はないと思われる. むろん, その進化の挙動やその最終的な結果に驚くべき工学的な価値がある可能性までは否定しない. そのような気の長いタイムスケールではなく, もっと直接的に, 我々が必要としているのは, むしろ, 従来の GA が生みだす家畜化したエージェントと人工生命の生みだす野性化したエージェントの中間に位置する進化の形態を持つエージェントであると考えられる.

2.3.4 人間世界と人工生態系の融合

人工生態系においてエージェントが自律的に戦略を進化させてゆく世界は, 必ずしも生来, 人間世界と完全に分離しているものではない. そもそもエージェントは人間の依頼を実行するために働くものであり, エージェントの生態系の最上部に位置して人間と接っするエージェントは, 直接的に人間からの命令を受けとるため, その効用関数がア・プリオリに定義されたとしても何の矛盾もない.

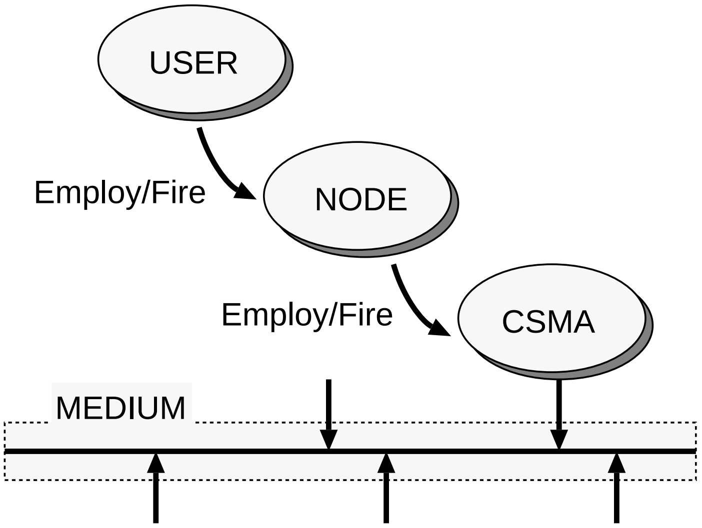

図 2.3: 人工生態系内の雇用関係

人間の要求に対応する効用関数を持ったエージェントは, 人工生態系の内部では自らの判断で活動し, 要求を実行するために仕事の一部を別のエージェントに依頼する. ネットワークシステムにおいて, 人間の QoS 要求を受けとったエージェントは, 実際に共有メディアにデータを送信する段階で, 特定のメディア・アクセス方式を実装する別のエージェントにデータの送信を依頼するかも知れない. 依頼されたエージェントも生態系の一部であり, 自らの戦略をもって自律的に行動するだろう. それらのエージェントの戦略は

ゲーム理論的な “駆け引き” を反映し, 必ずしも協調的とは限らない環境で資源の共有が行なわれるはずである. もし仕事を依頼した側が, 得られた結果に満足できなければ, そのエージェントは次から依頼する相手を変更することができる. この選択はエージェントの戦略の一部を変更することに相当する. 遺伝子の交叉や突然変異によって生まれるエージェントの変種の中には, 依頼する側にとって好ましい戦略を持つものもいれば, そうでないものもいるだろう. それらの戦略は依頼する側に選択され, 生き残ることによって, その環境に適応してゆくはずである. これらのエージェント間の関係は, 一種の “雇用関係” と見なすことができ, 生物学的というよりもむしろ, 経済的な選択淘汰の原理が人工生態系内部の進化を促すと考えることができる (図 2.3).

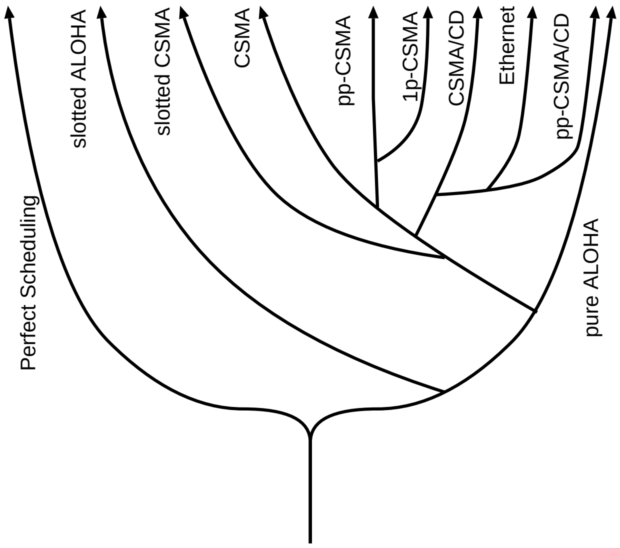

図 2.4: メディア・アクセス方式の進化木

エージェントの変種は, 必ずしも遺伝的に発生する必要性はない. 最も単純な ALOHA 方式から Ethernet に至るまでの, 現実の共有メディア・アクセス方式の進歩 (図 2.4) は, アクセス方式が勝手に進化したものではなく, 人間が考えて実装してきたものである. しかし, 生物界の生態系においても, 近年の遺伝子組替え技術の進歩によって, 例えば『トマト』の遺伝子を組替えて『日持ちの良いトマト』を作り出すことは可能であり, それらが農家で生産され, 流通し, 消費者に受けいれられ, それによって再び農家は生産量を増やし, 最終的に日持ちの良いトマトが『選択』され, 生態系の一部となりうることを想像すれば,『人間という生物を含めた自然環境』によって選択されたトマトは, 立派に進化したトマトと呼ぶことができよう. 同様に, 人間によって手の加えられたエージェントであっても, それが人工生態系の中でエージェントによって選択されるとすれば, システム全体がエージェント指向の進化を遂げると考えることができる.

第 3 章

シミュレーション・モデルの構築

本章では, 前章で述べたサーベイと考察に従って, 利己的なエージェントが共有メディアという土台の上で資源競合し, それらが資源を利用するための戦略を選択することによって, システム全体が進化してゆくシミュレーション・モデルを構築する. このモデルは, あくまで, Ethernet に類似した共有メディアをテストベッドとして, 共有メディアにおける利己的なエージェントの振舞いを観察するためのモデルであり, Ethernet の性能を解析したり, 新しい共有メディア・アクセス方式を開発することを直接の目的としているわけではない.

3.1　アーキテクチャ

本章の冒頭でも述べた通り, ネットワークを利用するための QoS を設定し, それらをユーザが自由に選択して使用できることが重要である. 本モデルではエージェントの戦略を QoS に対応させ, エージェントを選択することは QoS を選択することであるとする. エージェントは, あるサービスを提供するために, 別のサービスの利用者となり, 一つのサービスに関して『ユーザ』と『プロバイダ』の二つの立場としての側面を持つ. 一つの QoS 要求を実現するために, エージェントは, いくつかの階層を形成する. 人間の要求する "High-Performance" や "Low-Cost" といった抽象的な QoS は, 多段のエージェントの階層を下るにつれて, 具体的なパラメータ (転送レート, 遅延など) に解釈されてゆき, QoS を実現するための抽象的なアーキテクチャ[Cam93] を構成すると考えることができる.

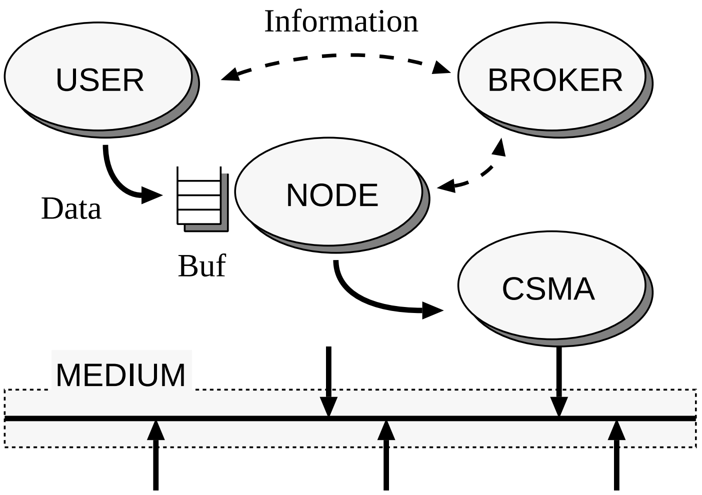

図 3.1: シミュレーション・モデルの構成要素

本モデルでは, 共有メディアを中心に, ユーザ, ノード, CSMA エージェント, 情報ブ

ローカ, の四種類のエージェントを設定し[1], それらが構成要素となる人工生態系 (図 3.1) を考える. 以下, それぞれの要素の説明を行なってゆく.

3.2　共有メディア

本モデルで使用する共有メディアは, Ethernet と同等の性能と見なせるように, 物理的パラメータを, バンド幅 10Mbps, 最大伝播遅延 23.2μ秒と設定した. ただし, パケットの衝突以外によって (例えばノイズによって) パケットが失なわれることのない理想媒体であるとし, また, 全ての CSMA エージェント間の伝播遅延が均一となるよう, balanced-star 型のトポロジを持っていると仮定している. (図 3.2) このような設定は現実的ではあり得ないが [Bog88], 今回のシミュレーションの目的が Ethernet 自体の性能解析ではないため, 無闇に複雑な設定はしないことにする. もちろん, このことは本論文での議論の本質を覆すものではない.

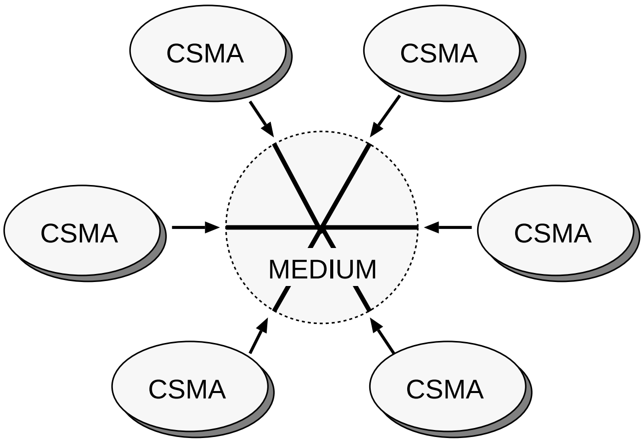

図 3.2: CSMA メディア

3.3　CSMA エージェント

CSMA エージェントはメディア・アクセス方式として, バックオフ可能な p-persistent CSMA/CD を実装し, 共有メディアへパケットを送信するサービスを利用者に提供する. 本モデルでは pp-CSMA の待機時間を 2 つのパラメータα, p を用いて連続関数で表現し:

$$(\text{待機時間}) = \alpha \times \log(1 - R)/\log(1 - p)$$

[1]ここでは共有メディアをエージェントとは考えないことにする.

(ここで R は (0.0,1.0) の一様乱数) とした. p はキャリア・センス後に送信動作に入る確率を表し, α は待機時間のスケーリング・ファクタとなる.

CSMA エージェントは pp-CSMA に加えて, 共有メディアで発生するパケットの衝突を検出することができる. エージェントは, パケットを送信する前にチャネルを観測し, メディアが使用中でないことを確認した上でパケットの送信を開始するが, メディアに伝播遅延があるため, 送信開始直後に別のエージェントが送信したパケットのキャリアが届く可能性がある. そのキャリアが届いた瞬間に, エージェントはパケットが衝突したことを認識できるものとする. パケットの衝突を検出したエージェントは, その事実を, メディアの利用者全員へ確実に知らせるために, Ethernet の規格で定義された最大妨害時間 (maximum jam time), 4.8μ秒の間, そのままデータを送信しつづける. 以上の動作が正常に行なわれるためには:

$$(\text{パケットの時間長}) \geq 2 \times (\text{最大伝播時間}) + (\text{最大妨害時間})$$

(64 バイトに相当) でなければならない. また, 平均アクセス時間を制限するために Ethernet と同様にパケットの最大長を 1518 バイトと設定する[2].

CSMA エージェントはパケットの衝突に伴う再送において, 再送の回数 $retries$ に応じて待機時間をバックオフ (binary exponential backoff) することができる. この時の待機時間は:

$$(\text{待機時間}_{backoff}) = 2^{retries} \times \alpha \times \log(1-R)/\log(1-p)$$

と定義することにする. Ethernet は再送回数に 16 回という制限を設けているが, 本モデルでは再送回数に制限はないことにする[3].

3.3.1 CSMA エージェントの突然変異

本モデルにおいて, CSMA エージェントは他のエージェントの選択を一切行なわない. 一方, CSMA エージェントの持つパラメータ, p とα は遺伝的に獲得していく形質であると見なし, 特定の確率でそのパラメータを変異させることにする. 現在は, CSMA エージェントがメディアにパケットを送信しようとする直前に, 100 分の 1 の確率でパラメータが変異する設定にしている. その際, p は 0.0 から 1.0 までの値を 32 ビットのビット列に変換した時にハミング距離 1 で値を変更する. この場合, ビットの位置によってビット変化の重みが異なるため, 変異するビット位置は, 一様乱数を 2 を底とする指数分布に変換して決定される. ただし, それだけでは変化の度合が小さすぎるため, 分布のグラフに 19 ビットのバイアスを加える操作を行なっている.

α は 0.0 から 1.0 の値に限定されないため, ハミング距離 1 の変異を設定することは難しい. そのため現在は, 一様分布の乱数を用いて最大 3%の増減を変異として設定して

[2]ただし, 今回のシミュレーションでは, 125 バイトと 1250 バイトの固定長パケットしか扱わない.

[3]制限はないが, 著しく性能が低下するような状況に陥ったエージェントは, 上層のエージェントによって選択淘汰されるだろう.

いる. ただし, ある値からランダムウォークした時に元の値に戻ってくるように, 減少する場合は元の値の最大 3.0%, 増加する場合は元の値の最大 3.09%を増減させせることにする[4]. α の変異の方法は, それを利用するエージェントが局所解に陥いる可能性を否定しきれないため, 今後も検討が必要である.

上記のα と p 以外にも, バックオフを行なうか行なわないかを制御するパラメータも遺伝子として持たせることにする. バックオフの性質が突然変異で変化するような設定が妥当であるか否かは疑問だが, 本モデルでは 10000 分の 1 の確率で変異することができるように設定している.

なお, 遺伝子に対する遺伝的操作は, 本モデルでは突然変異のみであり, 交叉は全く行なわれない.

3.4 ノード

ノードは利用者から入力されたパケットをバッファに蓄え, CSMA エージェントを利用して, それをメディアへ送信する. ここでノードと CSMA エージェントを別の実行実体として区別したのは, ノードが使用するメディアが必ずしも CSMA 共有メディアに限られず, FDDI や ATM であったりすることを想定して, 汎用性を持たせる意味で分離した.

3.4.1 CSMA エージェントの観測

ノードは使用する CSMA エージェントの性能を観測することができる. CSMA エージェントに対する一回の送信依頼に対し, CSMA が送信に成功した場合に次の値を観測し, その平均値を計算する:

- 性能: (パケット長 $[bit]$)/(送信時間 $[sec]$)
  メディアの瞬間的な送信レート. 最大レートはバンド幅と同一で, 10Mbps.

- コスト: (送信ビット数 $[bit]$) $\times$ (メディア単価 $[yen/bit]$)/(送信時間 $[sec]$)
  本稿のモデルでは送信ビット数に対する単価は 1.0 に設定しているため, そのまま送信ビット数がコストに反映される.

これらの値は, 時間に関して加重平均される. 現在の設定では, 1.0 秒毎に 50%の割合いで過去の平均値は忘れられてゆく.

3.4.2 ノードの効用関数

各ノードは CSMA エージェントの観測結果に対して独自の効用関数を定義し, 効用を CSMA エージェントの評価に用いる. ノードの効用関数は, CSMA の性能に対する評価

[4] $x = (x \times (1-y)) \times (1/(1-y))$, $y = 0.03$ の時, $1/(1-y) \simeq 1.0309$ である.

をコストのメトリックに変換し, その比を取ることで計算されるが, ここでは簡単に:

$$(\text{ノードの効用}) = \frac{(\text{加重平均の性能})}{(\text{加重平均のコスト})^{\eta}}$$

としている. η はコストの重みづけの意味を持つ. ノードの効用は CSMA の観測と同様に加重平均される. ノードは効用の時間変化を調べ, その傾きを計算している. 傾きは, 効用の加重平均値と新しい観測値の差を加重平均して得られる.

3.4.3 ノードの判断のタイミング

ノードは効用の時間変化の傾きを調べることで, 使用中の CSMA エージェントの良し悪しを判断することができる. この判断は平均 1.0 秒の時間間隔のポワソンレートで非同期に行なわれる. 判断の際にノードは, 効用の傾きが負の場合に限り, 30%の確率で CSMA エージェントを解雇し, 新しいエージェントを雇うことにする. その方法の詳細については後述する情報ブローカの項で説明する.

3.4.4 ノードの突然変異

ノードに対しても突然変異の遺伝的操作が施される. ノードが CSMA エージェントにパケットを送信する時に, 1000 分の 1 の確率で η の値を変異する. その変異方法は CSMA エージェントの α の変異と同様である.

3.5 ユーザ

ユーザは, あるポワソンレートでパケットを生成し, パケットのストリームをノードに入力する.

3.5.1 ノードの観測

ユーザは使用するノードの性能を観測することができる. ノードに対する一回の送信依頼に対し, ノードが送信に成功した場合に次の値を観測値し, 平均値を計算する:

- 性能: (ノードが送信した積算ビット数 [bit])/(積算時間 [sec])
- コスト: (ノードが費した総コスト [yen])/(積算時間 [sec])

これらの値は, 時間に関して加重平均される. 現在の設定では, 1.0 秒あるいは 10.0 秒毎に 50%の割合いで過去の値は忘却されてゆく.

3.5.2 ユーザの効用関数

各ユーザはノードの観測結果に対して独自の効用関数を定義し, 効用をノードの評価に用いる. ユーザの効用関数は, ノードの性能に対する評価をコストのメトリックに変換し, その比を取ることで計算される. 単純なものとしては, ノードが自分の効用を算出するのと同様に:

$$(\text{ユーザの効用}) = \frac{(\text{加重平均の性能})}{(\text{加重平均のコスト})^{\kappa}}$$

としている. κ はコストの重みづけの意味を持つ. この他の効用関数については 5 章で説明する. ユーザの効用はノードの観測と同様に加重平均される. ユーザは効用の時間変化を調べ, その傾きを計算している. 傾きは, 効用の加重平均値と新しい観測値の差を加重平均して得られる.

3.5.3 ユーザの判断のタイミング

ユーザは効用の時間変化の傾きを調べることで, 使用中のノードの良し悪しを判断することができる. この判断は平均 1.0 秒あるいは平均 10.0 秒の時間間隔のポワソンレートで非同期に行なわれる. 判断の際にユーザは, 効用の傾きが負の場合に限り, 30%あるいは 100%の確率でノードを解雇し, 新しいエージェントを雇うことにする. その方法は次の情報ブローカの節で説明する.

3.6 情報ブローカ

ユーザ側のエージェントは, 効用の変化を見て, 使用中のエージェントの評価が下ってきている場合, それを変更することができる. しかし, 次に『どの』エージェントを選ぶべきかを判断することは, それほど簡単な問題ではなく, 判断の基準となる『情報』が必要である. そこで本モデルでは, エージェント間の情報伝達に情報ブローカを介在させ, エージェントを変更するために必要な情報を蓄積させる.

エージェントの変更の判断と同じタイミング (平均して 1 秒に一回, あるいは平均して 10 秒に一回のポワソンレート) で, 各エージェントは使用中のエージェントの成績を情報ブローカに報告する. エージェントの成績とは, 効用関数とは無関係な, エージェントが出した客観的な結果であり, ユーザ側の判断は加えられない. もしプロバイダを変更しようという場合は, ブローカへの報告と同時に, 次に選択すべきエージェントの情報をブローカから得ることができる.

ブローカは FIFO の名前空間[5]を持っており, 使用中のエージェントの成績も含めて, 過去に報告されたエージェントの成績が蓄積されている. 本稿のモデルでは, FIFO のス

[5]全てのエージェントは異なる整数の名前を割り当てられている. エージェントは名前を指定してコピーすることが可能であるとする.

ロットを 512 個用意している. 変更の際には, エージェントはブローカに自分の効用関数を渡し, ブローカの名前空間に含まれるエージェントの中で, 過去に最も高い効用を与えたエージェントを探索させる.

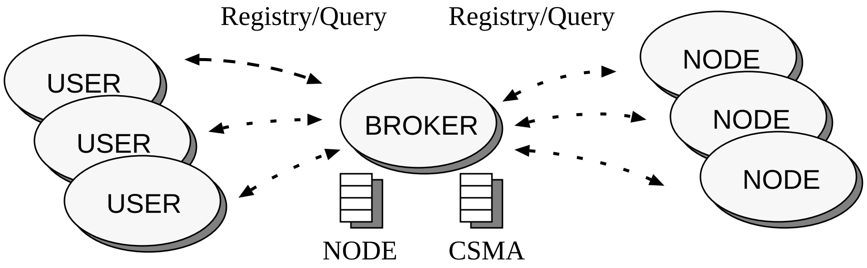

図 3.3: 情報ブローカ

遺伝的アルゴリズムの枠組みでは, 世代交代に際して, いくつかの選択スキームが考えられる. 適応度に比例してプレーヤを入れ換える方法や, 最高の適応度のものを増植させる方法, その他, いくらでも方法が考えられるが, それは解を求めようとしている人間にとって適切なのであって, 本モデルのような自律的なエージェントが自分の判断で選択を行なう場合にまで, 直接的に当てはまる方法論ではないように思える. なぜなら, 仮に確実にランキングされたエージェントを選択するとして, なぜ 2 位以下のエージェントを選択することに魅力を感じるかの説明がつかない. むろん, 確実にランキングされていない場合は別である. ランキングが不確実であることを知っている利口なエージェントは, 自分の判断で 1 位でない, 別のエージェントを選択するかもしれない. しかし, そのスキームを実装するのは困難であるため, 本モデルでは単純にランキングのトップに当るエージェントを選択させることにする.

一方, 解雇されたエージェントがどのように保存されるかというと, それはブローカの FIFO が一周するまでは, ブローカが保存することにする. もちろん, ブローカの FIFO は単なる名前空間であり, その中には捨てられたエージェント以外に, 現在, 使用されているエージェントの名前も記録されている. FIFO が一周して, 既に登録されていたエージェントの名前を消す必要がある場合, 使用中でなければ名前から実体を調べてブローカが deallocate することになる. 逆に使用中であれば, その名前は単純に消してよい. 使用中のエージェントは生き続けているため, すぐ後に登録され直すだろう.

ブローカはエージェントの中身が何であるかは理解しない. その価値を判断するのは, それらのエージェントを雇おうとしている別のエージェントである. 新しくエージェントを雇う側のエージェントは, ブローカが返したエージェントの名前を調べ, そのエージェントのコピーを作ることができる. そのため, 現役で働いているエージェントの遺伝子を, それが生きている間に複製することになる. もし, 選んだエージェントが捨てられたものであれば, コピーせずに, そのまま再利用することになる.

本モデルにおいて, エージェントの遺伝子の突然変異がコピーミスで起るものではないことは, 覚えておく必要があるだろう. 本モデルにおいて, 突然変異は定められた確率で定期的に発生するものであり, そのエージェントが生きている最中にも遺伝子が変化するという設定を行なっている.

新しいエージェントを雇うためにはコストが必要である. エージェントの交代にはコストや時間のペナルティを課すことにしている. ノードが CSMA エージェントを変更するためには, 10000 ビット分のメディア消費と, 100m 秒のペナルティ(変更中は送信できないはずである), ユーザが ノードを変更するためには 500m 秒のペナルティ(それと一定量のコスト) を設定している. ただし, エージェントとブローカの間で実際には行なわれるはずの通信は, 本モデルでは実際にはシミュレートしていない. 設定したコストと時間のペナルティ以外には, シミュレーション上のイベントは発生しない. 共有メディアの上を一般のパケットと同様に伝送することも可能であるが, あくまで共有メディアは通常のパケットの伝送に使用するという方針で設定した. これは本モデルの中で唯一, リアリティに欠けた設定となっているが, このことは本論文における議論の本質を覆すほどのものではない.

第 4 章

シミュレーションと評価

4.1 予備シミュレーション

まずは準備段階として, p-persistent CSMA/CD の特性を調べる予備シミュレーションを行なった. 3章で述べたモデルにおいて, ユーザは平均 10m 秒間隔でポワソン到着する固定長パケットのストリームを生成し, ノードへ送信する. ノードは無制限のバッファを用意して, それらのパケットをバッファリングし, CSMA エージェントに送信する. そして, CSMA エージェントはバックオフ無しの設定で, pp-CSMA/CD 方式でメディアにアクセスする. pp-CSMA の待機時間は α を最大伝播時間の 10 倍に設定し, p をパラメータとして, トラフィックに対するスループットの特性を測定した. ここで, トラフィックは, ストリームを発生させるノードの数によって見積っている. つまり, 一つのノードの生成するトラフィックの整数倍のトラフィックのみを観測している. 一方, スループットは, シミュレータをシミュレーション時間で 5.0 秒間回した時の, 衝突せずに有効に情報を運んだパケットの総計レートと定義した.

遅延の大きさを $a =$ (伝播遅延時間)/(パケットの時間長) で規格化した時, $a = 0.0232$ のシミュレーション結果を図 4.1に示す. グラフはトラフィック, スループット共にメディア・バンド幅 10Mbps で正規化してある.

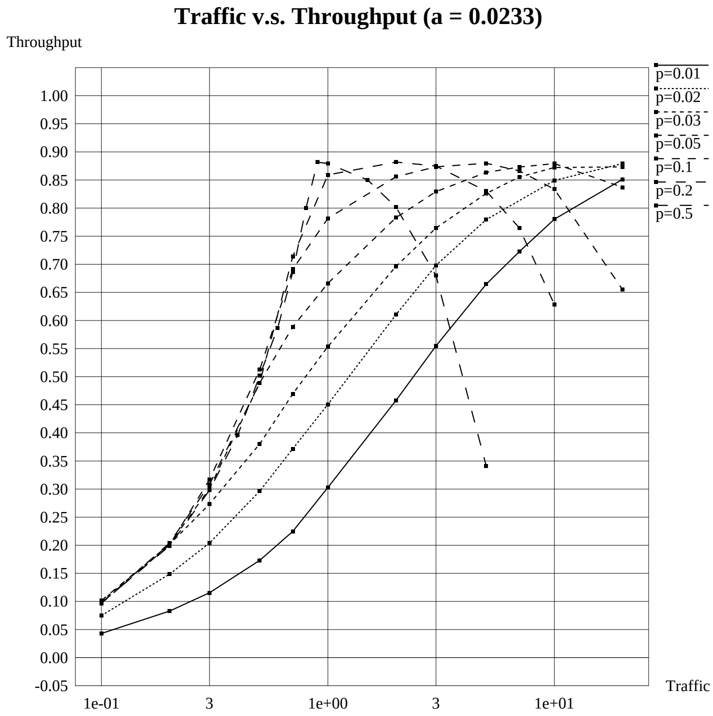

図 4.1: pp-CSMA/CD 特性 (a = 0.0232)

図 4.1に対応する理論値 [Kle75] として, slotted p-persistent CSMA の解析的結果を図 4.2に示す. 本来ならば pp-CSMA ではなく pp-CSMA/CD の理論値 [Apo86] [Coy83],

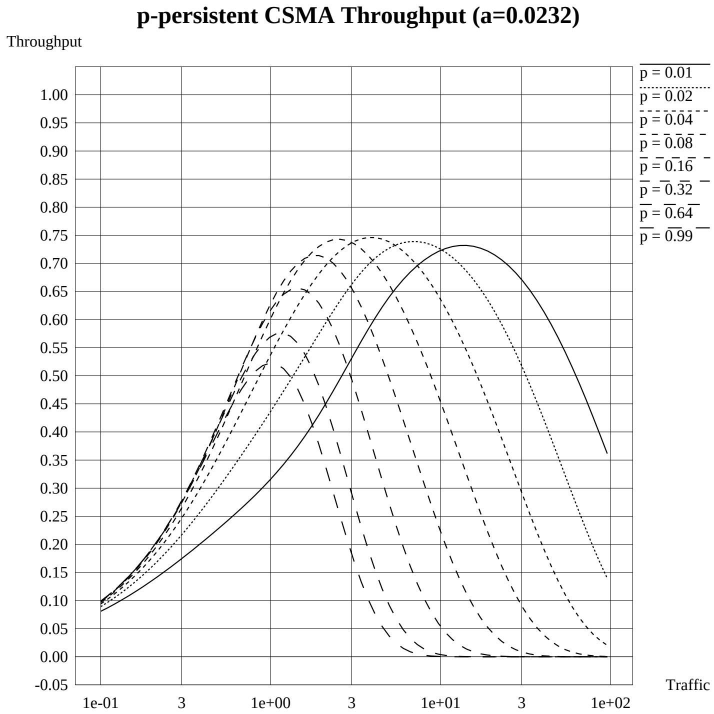

図 4.2: pp-CSMA 特性 (a = 0.0232)

[Tak85] [Tas86] を示すべきであるが, それらが再送中にバッファに溜ったパケットを無視しているなど, 多くの点で本稿の設定と異なる箇所があり, 結果として解析結果が大幅に異なるため, その代りとして純粋な pp-CSMA の理論値を示すことにした. ただしこの場合, 理論が『無限のユーザ』を仮定しているなどの点で比較の際には慎重に構える必要があるが, それに対応してシミュレーションでは無限のバッファを仮定しているため, 比較対象としては妥当であると思われる (ただし, 図 4.2さえも, p が小さい場合に当てはまる近似値である.).

図 4.1と図 4.2を比較すると, 理論値とシミュレーション結果は, おおよそ一致していることが分かる. 理論値に対してシミュレーションの結果が僅かながら改善されているのは, 衝突検出のアルゴリズムの影響だと思われる.

次に遅延の影響を調べるために, $a = 0.0232$ に対して, 10 倍の規格化した遅延を設定した $a = 0.232$ の場合のシミュレーション結果を図 4.3に, 理論値を図 4.4に示す.

図 4.1と図 4.3を比較すると, a の値が大きくなると, CSMA の性能が急激に劣化することが分かる. これはメディアに接続された CSMA を処理するプロセッサが, 遅延の大きさに応じた過去の情報に基づいてパケットの送信タイミングを決定しなければならないことに原因がある [Kle75].

以上のように, CSMA の理論値とシミュレーションのグラフの概形がおおよそ一致し,

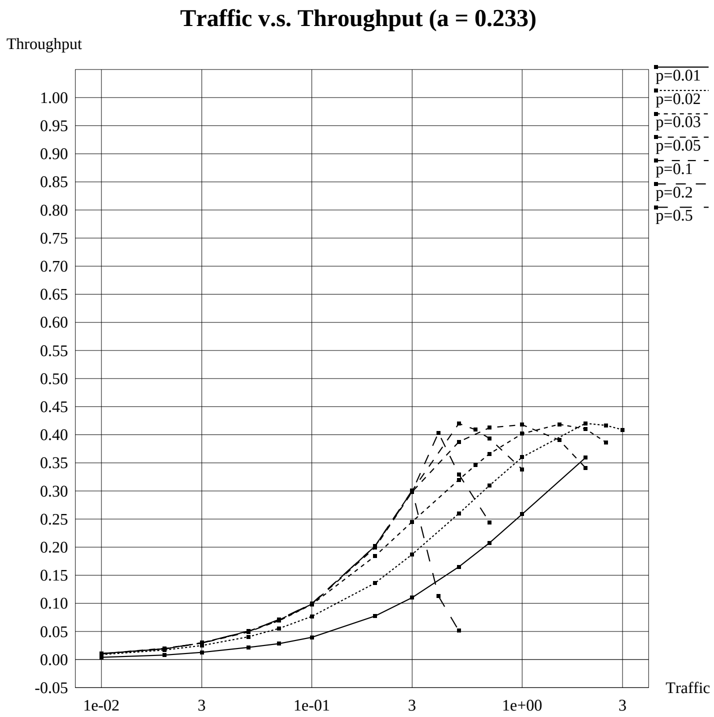

図 4.3: pp-CSMA/CD 特性 (a = 0.232)

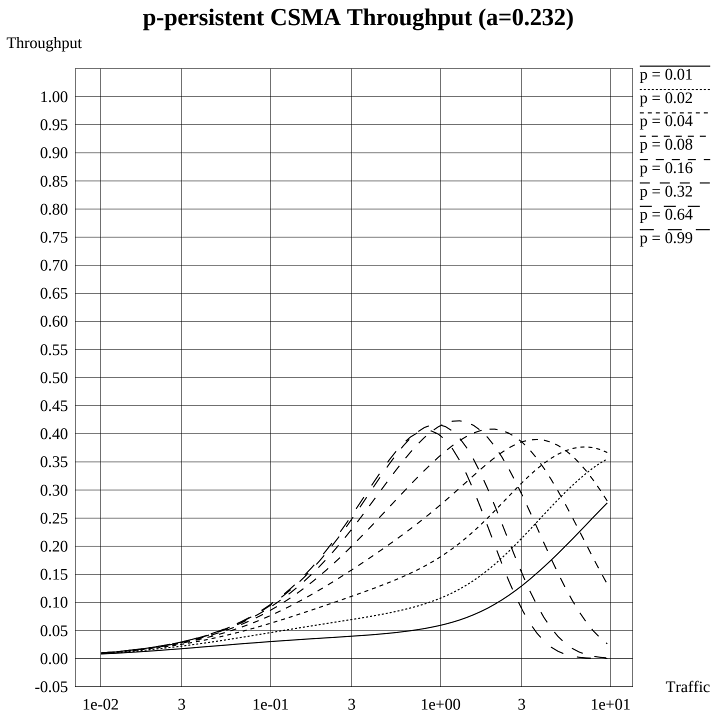

図 4.4: pp-CSMA 特性 (a = 0.232)

伝播遅延に対する反応も, 傾向としておおよそ一致することから, この予備シミュレーションの結果が, 実際に共有メディアにおける CSMA の特徴を表現していると言うことができるだろう.

4.2 利己的なエージェントによる資源共有

本節では, 非協調的アプローチによる輻輳制御, そして利己的なエージェントによる資源共有という観点からシミュレーションを行なう. これから述べるシミュレーションでは, 人工生態系内部でエージェントの自律的な判断に基づく選択淘汰が行なわれる環境で, 12 人のユーザが平均 1Mbps のストリーム (その遅延は前節の $a = 0.0232$ に相当する) を生成する. 生成されるストリームの総計レート (12Mbps) がメディア・バンド幅 (10Mbps) を越えるため, 潜在的に資源競合が行なわれる設定となっている[1].

4.2.1 設定

前節までの予備実験とは異なり, ノードのバッファには制限を設け, 32 個までのパケットをバッファリングできるようにする. バッファが埋め尽されている時は, 後から生成されたパケットは捨てられる. エージェントは性能/コスト比による独自の効用関数を持ち, 観測された性能に満足できない時は, 使用中のエージェントを変更することができる. また, エージェントは依頼した仕事にタイムアウトを設けて, その時間内に仕事を終えないエージェントを強制的に解雇することができるようにする.

ここでは, ユーザの効用関数は全て$\kappa = 0.9$ と設定し, ユーザは平均 1.0 秒間隔のポワソンレートで判断を行ない, 効用の傾きが負の場合に限って 30%の確率でノードを選択する. 一方, ノードによる CSMA エージェントの選択は行なわれないものとする. つまり, ここでは, ユーザによるノード (そして, 連帯的に, ノードが雇っている CSMA エージェント) の選択のみが行なわれる. また, タイムアウトによるエージェントの変更はユーザのみが実行し, そのタイムアウト値は 3.0 秒とした. ノードを変更している間, ユーザはパケットを送信することができない. その時でも, ノードのバッファが空いていればバッファに蓄積することができる.

なお, これから示すシミュレーション結果の全てのグラフにおいて, シミュレーション時間における 2.0 秒毎の値のプロットである. また, 全てのデータは 12 人のプレーヤを用いてシミュレーションされたものだが, 全てのプレーヤが同様な特性を持っているような場合には, グラフの可読性を考慮して, 必要に応じてプロットする線の数を 12 本から 8 本に減らして表示している.

[1]生成レートが 10Mbps であっても, $a = 0.0232$ の場合の最大スループットが 10Mbps より低いため, 競合状態は発生する.

4.2.2 コストを考慮することの意義

ここではまず, コストを考慮することの意義を調べるため, 全くコストを考慮せずに振舞うエージェントによる資源競合とコストを考慮した場合の資源競合の様子をシミュレーションで観察する. コストを考慮しないとは, ユーザの効用関数が「性能」と一対一対応していることを意味する.

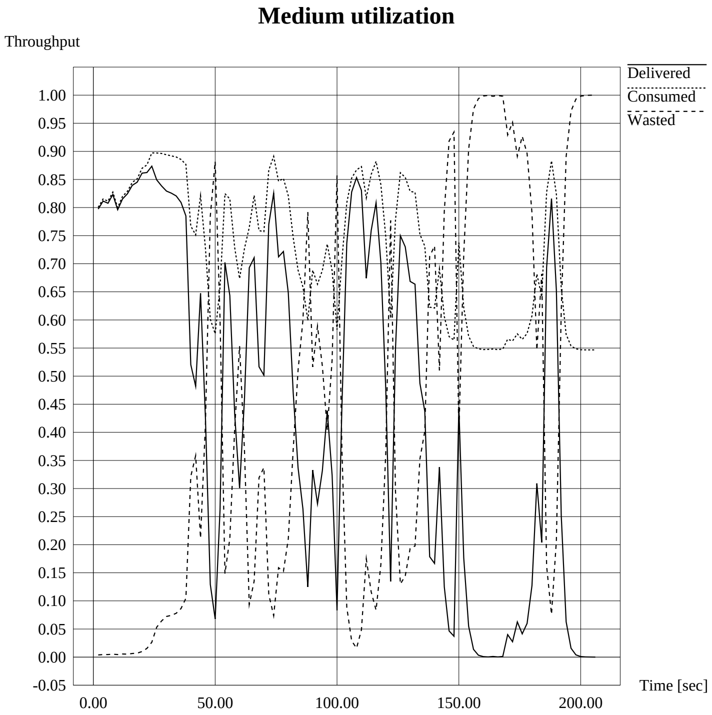

図 4.5: コストを考慮しない場合のスループット

図 4.5は, エージェント間の資源競合によって, 有効に利用されたバンド幅 (delivered), 再送も含めて消費されたバンド幅 (consumed), 消費されたバンド幅に対する失なわれたバンド幅の比率 (wasted) を時間に対してプロットしたグラフである. ユーザの効用関数は $\kappa = 0.9$ に支配されており, 定常的なメディア価格レートを設定した場合は, より多くの資源を獲得する方向に振舞う[2].

図 4.5を見ると, システムは全く制御のきかない不安定な状態に陥り, 最終的にスループットは 0.0 に落ちこんだ. これはユーザの効用関数 (すなわちレート) が激しく振動を続け, それらのエージェントが闇雲にパケットを送信して, 使用するバンド幅のほとんどをパケットの衝突で浪費している状態である. この時に選択された CSMA エージェントの p の分布は図 4.7, α の分布は図 4.9のようになった. これを見ると, p に関しては脈絡のない選択が行なわれ, レートを上げるために α を下げる傾向にあることが分かる. ただ

[2] κは秩序変数になっており, 1.0 より大きいか小さいかでエージェントの振舞いが異なる.

し, α の値が 10μ 秒付近まで落ちると, 既にメディアの伝播遅延よりも小さい値になっており, それより小さな値になってもレートの改善がないため, そのようなエージェントは淘汰されてしまっている.

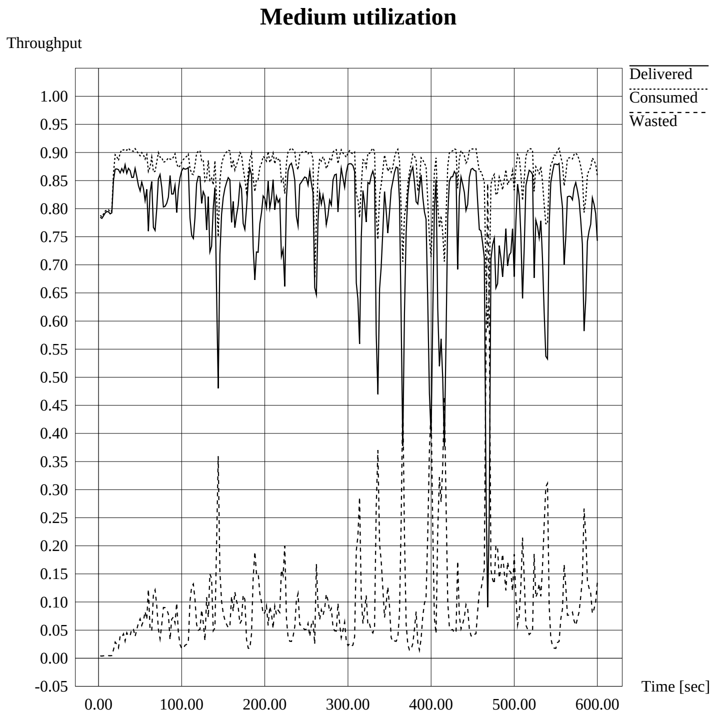

図 4.6: コストを考慮する場合のスループット

これに対して, コストを考慮するエージェントは対照的な振舞いを見せた (図 4.6). おおよそ 80 ~ 90%のバンド幅を消費しつづけ, パケットの衝突によって失なわれるバンド幅は, 総消費量の 10 ~ 20%に留まった. p と α の選択の様子 (図 4.8, 4.10) を見ても, これらはコストを考慮しないエージェントによる選択とは傾向が異なることが分かる. 図 4.7(コストを考慮しない場合) では無作為に選ばれているようにも見えた p の選択が, 図 4.8(コストを考慮する場合) では 0.5 よりも大きな値で集団を形成しているように見える. それとは反対に, 図 4.9(コストを考慮しない場合) では 0 に向って一方的に選択された α の値が, 図 4.9(コストを考慮する場合) では, 10μ 秒から 300μ 秒付近の大きなレンジで周期的に推移している様子が観測された.

ただし, コストを考慮したからといって劇的というほどまではシステムの利用率は改善されていないように思える. 特に, 数十秒間隔で利用率が極端に下がるバーストの状態が周期的に発生しており, システムの安定性に関しては問題がある.

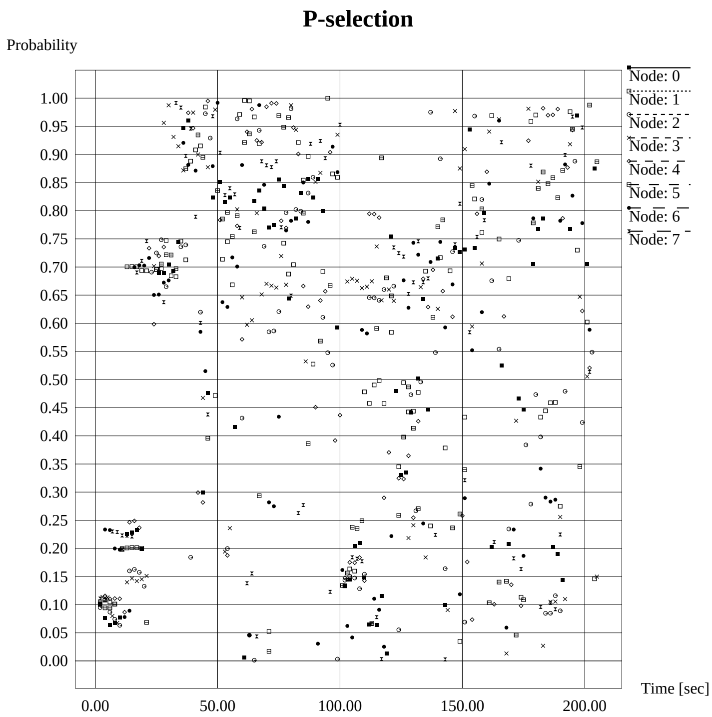

図 4.7: コストを考慮しないエージェントによる p の選択

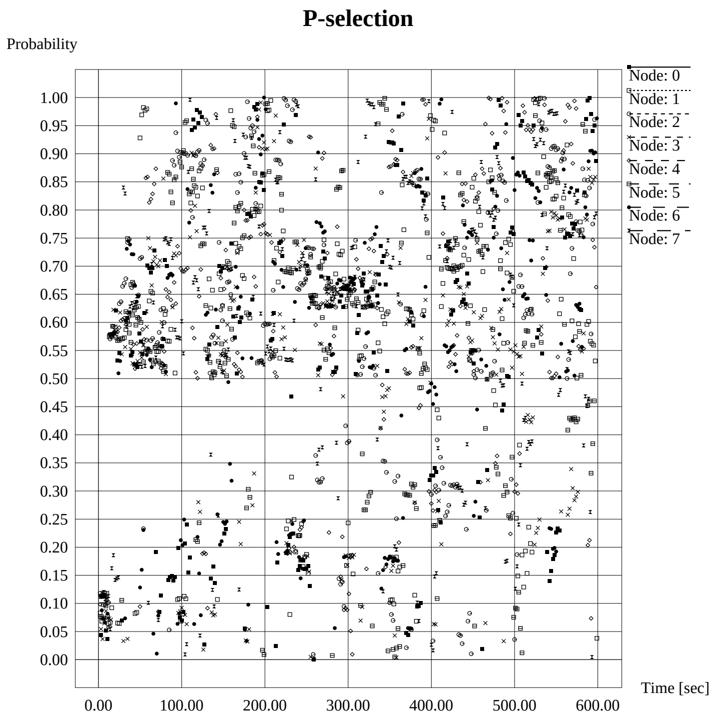

図 4.8: コストを考慮するエージェントによる p の選択

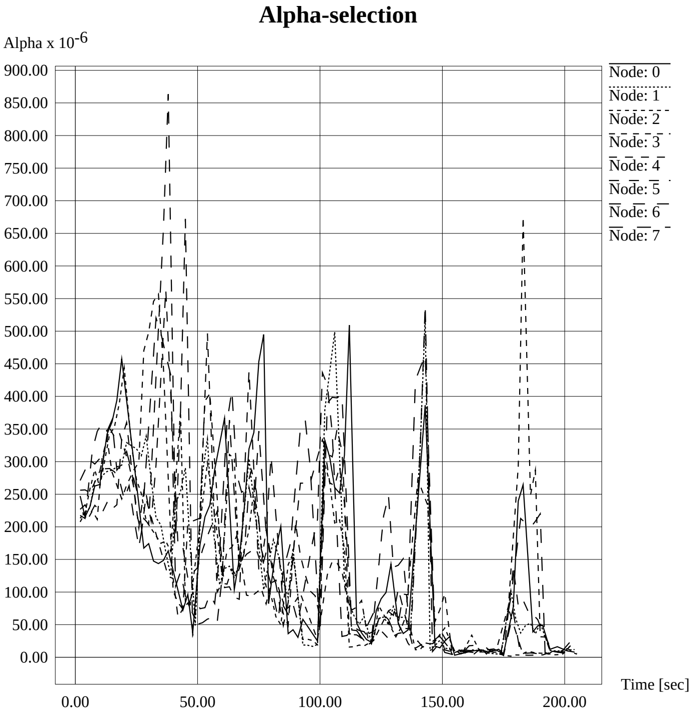

図 4.9: コストを考慮しないエージェントによるαの選択

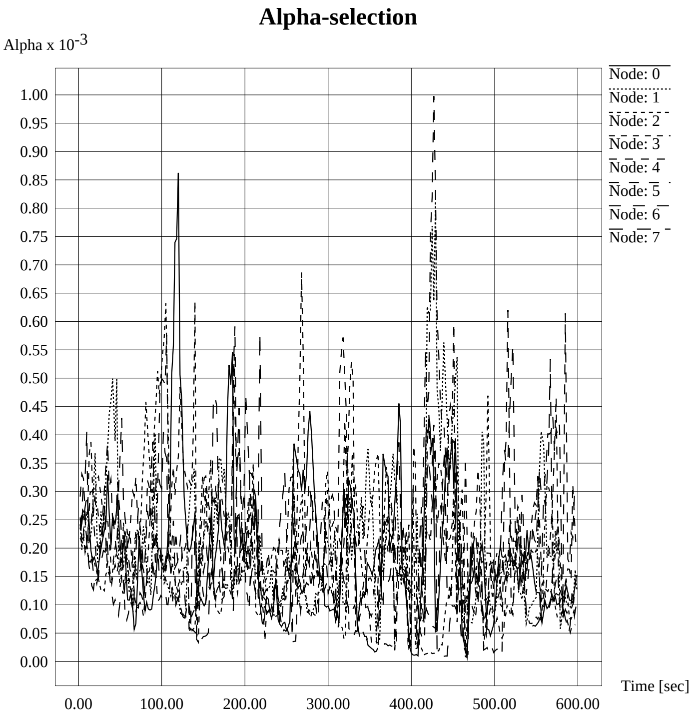

図 4.10: コストを考慮するエージェントによるαの選択

4.2.3 義務としての協調

Ethernet においては, 当然のことながらコストを考慮するエージェントの存在は考えられない. しかし, Ethernet は過負荷の状態でも目立ったパフォーマンスの低下もなく動作することが可能である [Bog88]. 最もインパクトを与える Ethernet の戦略は, バックオフの機構を用いることにある. 一般に, persistent CSMA 共有メディアの性能は, その再送レートによって大きく左右されることが分かっており[3], Ethernet では binary expornential back-off の戦略を用いることで, 再送レートを変動させ, コストを考慮しない場合でも, 常にシステムが安定な状態を作り出している.

本モデルにおいても, pp-CSMA/CD にバックオフの機構を加えることで Ethernet と同等の状態を作り出すことができる (図 4.11). メディア利用率は, ほぼ 80 ~ 90%で安定し, パケットの衝突によって失なわれるバンド幅の割合は 5%にも満たない.

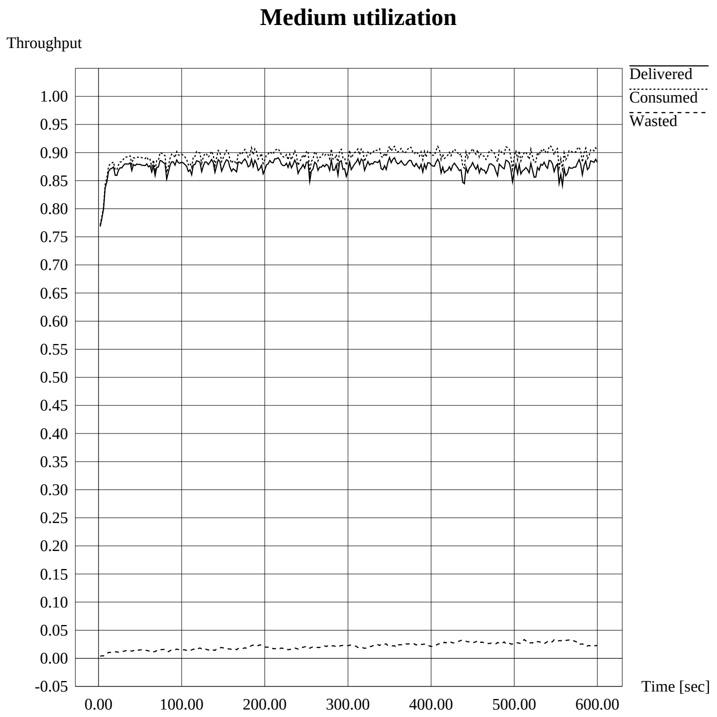

図 4.11: コストを考慮しない場合の義務としてのバックオフの効果

このように, 仮にエージェントが性能だけを求めて利己的に振舞ったとしても, バックオフという協調的メカニズムを義務として負わせることにより, システムを安定化させることができる. このシミュレーションから, Ethernet の戦略は極めて周到に設計されていると言える. 一方, 義務的協調がある場合, コストを考慮したとしても, 失なわれるバンド幅に注目すれば多少の改善はあるが, 目立った差はないようである. (図 4.12)

[3] 固定の, しかも高い再送レートは persistent CSMA の性能を著しく劣化させる.

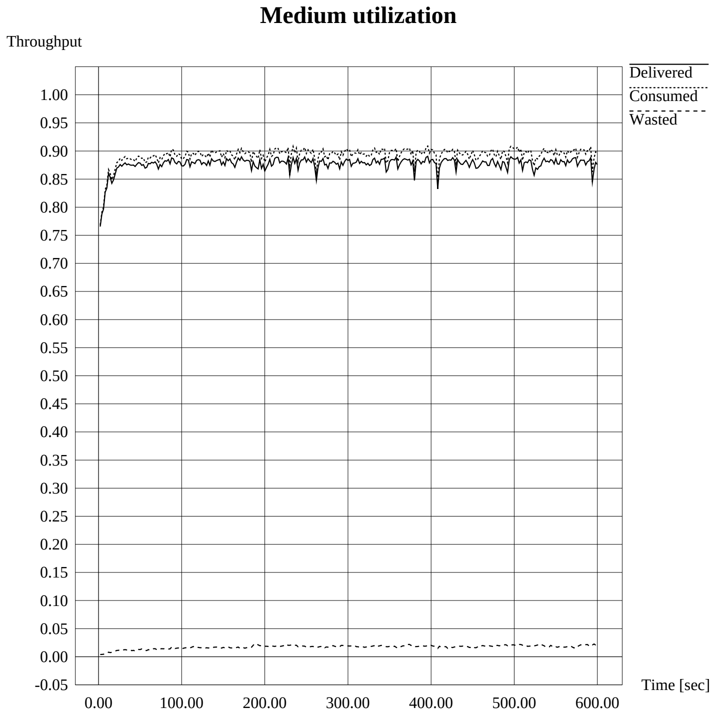

図 4.12: コストを考慮した場合の義務としてのバックオフの効果

4.2.4　利己的なエージェント

共有メディアにおいてバックオフという協調的アルゴリズムを用いることは, 丁度, Internet において TCP という協調的輻輳制御アルゴリズムを用いることに対応していると考えることができる[4]. TCP は slow-start や fast-retransmit の機構を用いて, その時, 利用可能なバンド幅に送信レートをアダプトさせる. これは送信者が皆, 送信したいという願望を抑えて, 一方的にパケットの送信レートを下げることにより, 協調して輻輳制御を行なうことを意味している. しかし, 輻輳の徴候を無視してパケットを送信し続ける利己的なユーザが加わった場合, TCP を用いるユーザはその利己的なユーザに一方的に搾取されてしまう. TCP は一方的にレートを下げ始め, 資源を全く使えない状況に陥ったとしても不思議ではないのである.

pp-CSMA/CD の場合でも, 『輻輳状態に陥った時はバックオフをする』という合意を無視し, 他のユーザがバックオフをした隙を狙ってパケットを送信してしまうという利己的なユーザの存在の可能性を度外視するわけにはいかない. この仮定は, 『実際の Ethernet で, そのような利己的なユーザを考える』という狭い範囲にとどまらず, 共有メディアのような集中制御を行なう管理実体がいない状況において一般的に考慮すべきことである. 仮に集中制御を行なう管理実体がいる場合ならば, そのサービス方針を巧妙に

[4]そもそも TCP も再送の判断のためのタイマ管理にバックオフの機構を用いている.

設計することで対処できるはずだが, そうでない場合に, システムは一体いかなる挙動を呈するのであろうか?

一人を除いて全員がバックオフする場合, バックオフを無視するプレーヤが圧倒的に有利な結果を得ることが想像できる. もしエージェントがバックオフするという戦略と, バックオフを無視するという戦略を (意図せずにでも) 選択することができる場合, エージェントはいかなる行動をとると予想できるだろうか. まず, 全員がバックオフすることを当然だと思っている状況があるとする. しかし, ある時, たまたまバックオフを無視して送信したエージェントが, 想像以上の利益を得ることを知ってしまう. そして,『バックオフすると得である』という噂がエージェントの間に広がるにつれて, それを真似してバックオフを無視するエージェントが増えてくるはずである. しかし, 既に図 4.5と図 4.11とで示した通り, バックオフを無視するプレーヤが増えるにつれてメディアの利用率は下り, 同時にコストは増え続け, 結果的に全てのプレーヤにとって悲劇的な状態に陥る. エージェントたちは現在の自分の選択に疑問を感じるだろう. ブローカに問い合わせてみると, ある程度の過去において, 自分にとってリーズナブルな結果を出すエージェントがいたことが分かる. そこでエージェントは現在の戦略を捨て, その戦略に乗り換えることになるはずである.

そのような日和見的な行動をとるエージェントたちによる資源競合の様子を, 図 4.13(コストを考慮しない場合), 図 4.14(コストを考慮する場合) に示す. まず, コストを考慮しないエージェントが『抜け駆け』的にバックオフを無視する場合, 図 4.13に示すとおり, システムは非常に不安定な状態に陥る. それに対し, コストを考慮するエージェントの場合,『抜け駆け』するエージェントがいるにも関わらず, 抜け駆けするエージェントの全くいない場合 (図 4.6) よりも, むしろシステムが安定化することは興味深い.

これらのエージェントたちが選択した CSMA エージェントの形質として, バックオフをするか否かを調べると, 図 4.15(コストを考慮しない場合), 図 4.16(コストを考慮する場合) のようになる. バックオフを実行するエージェントの割合は, コストを考慮する場合の方が, コストを考慮しない場合よりも, むしろ低いにも関わらず, より安定な資源の共有を行ない, それによってネットワークの利用率が高いことは注目に値する.

このように, 全てのユーザに義務的協調を負わせなくとも, コストによるフィードバックをかけることにより, 協調的戦略と非協調的戦略を共存させつつ, 全体として長期的にはある程度, 協調的な振舞いを呈することは興味深い. また, システム全体の性能という視点からも, 利己的なエージェントが日和見的に振舞う環境でも, コストによるフィードバックをかけることにより, システムが安定に動作することが, シミュレーションによって確認できる.

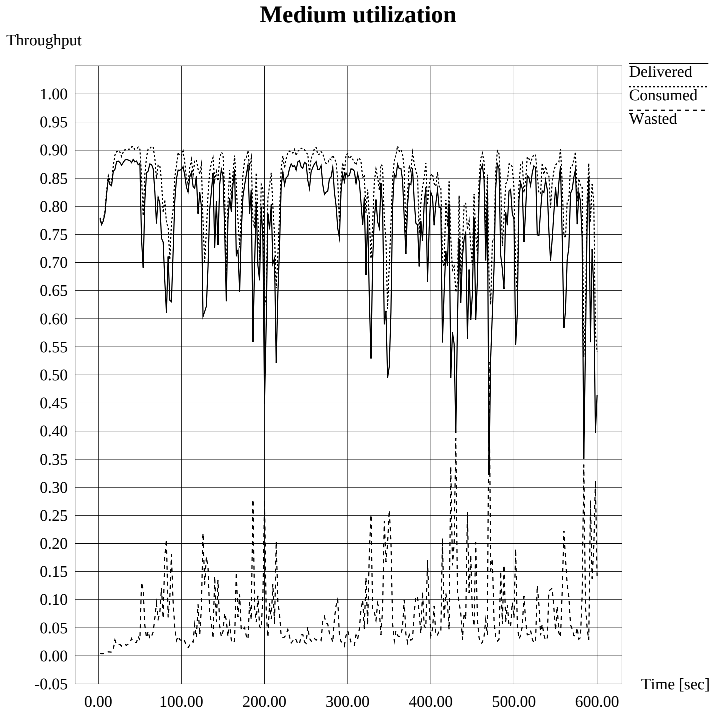

図 4.13: コストを考慮しない利己的なエージェントとバックオフ

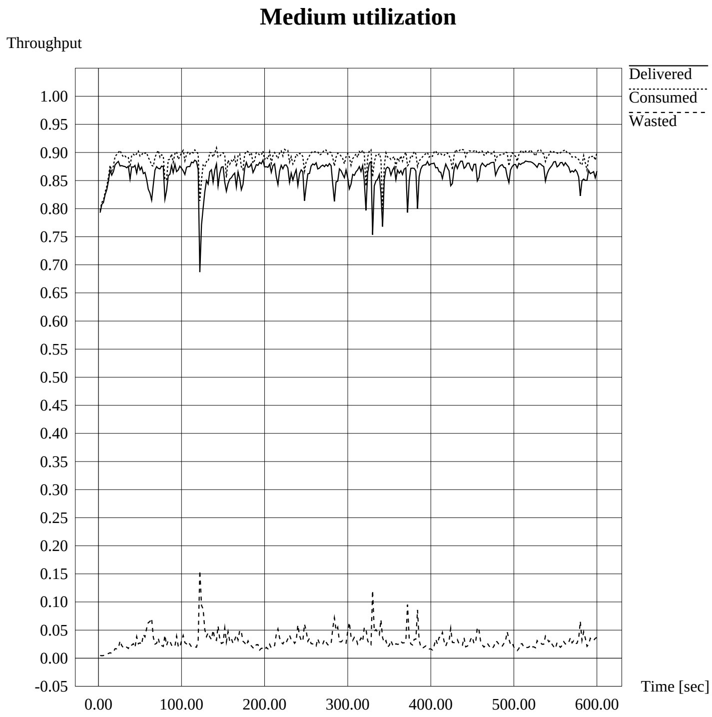

図 4.14: コストを考慮する利己的なエージェントとバックオフ

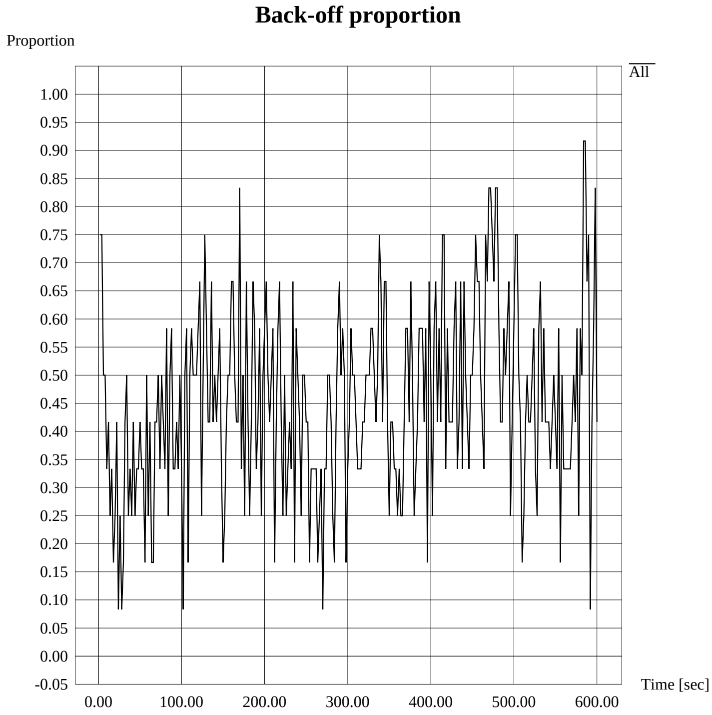

図 4.15: コストを考慮しない場合のバックオッフを使用するエージェントの比率

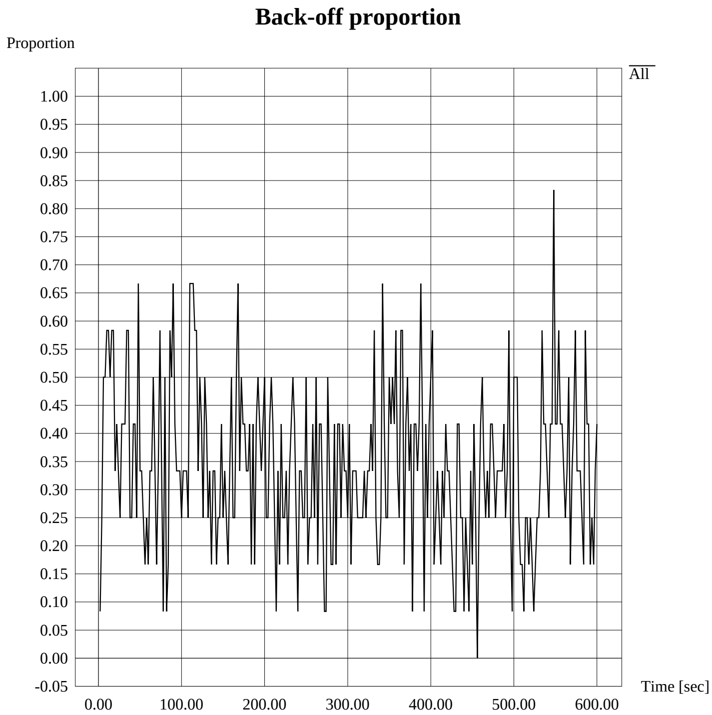

図 4.16: コストを考慮する場合のバックオッフを使用するエージェントの比率

4.2.5 システムの進化

ここで視点を変え, システムの進化という視点から, コストを考慮しない義務的協調のケースと, コストを考慮して抜け駆けを含んだ協調のケースを比較してみる.

コストを考慮しない義務的バックオフとは, p-persistent CSMA/CD with binary exponential back-off を意味する. ただし, CSMA エージェントが遺伝子として持つ p とαが突然変異によって遺伝的に変化するため, 同じ pp-CSMA/CD を実装したエージェントの間でも, 性能や性質が異なるものが存在する. 本モデルでは, 上位のエージェントに, それらの異なるエージェントの形質を選択させることによって, システムそのものを進化させることを提案している.

義務的にバックオフした場合の (既に述べた図 4.11と同じシミュレーションにおける) p とαの選択結果は, それぞれ図 4.17, 図 4.18のようになった. グラフを見ると, p は 0.1 から進化を始めて数十秒後に約 0.75 以上の値で安定な集団を作り (p が集団を形成していることは, 図 4.7や図 4.8と比較することで言えるだろう), α はメディアの伝播遅延に漸近するように値を下げ続けた. この時, ネットワークシステムが獲得した形質は, Ethernet が採用している戦略, 1-persistent CSMA/CD with binary exponential back-off に極めて近い. しかも, この形質が数十秒のオーダで獲得されることは, 今後の本モデルを拡張することを考えるに当たって心強い観測結果である. 逆に, 義務的に協調を行なう共有メディア型ネットワークにおいては, Ethernet の戦略はエージェントにとっても魅力的で, 結果として必然的なものであると考えることもできる.

一方, エージェントがコストを考慮した時の抜け駆けを含んだ協調のケースは, 図 4.19, 図 4.20のようになった. この場合, p は図 4.17と同様に集団を作りながら推移したが, 1.0 に向って進化するだけではなく, 400 秒を越えた付近で別の集団に全体が遷移するという興味深い挙動を見せた. α は図 4.18のような一方向の進化ではなく, 100μ ~ 300μ 秒付近で周期的な振動を見せた. これらの選択に対応するネットワークのスループットのグラフ (図 4.14) を見ると, 400 秒付近を境に突如としてシステムが安定な動作を始めており, 上記の値 (遺伝子) が選択された結果, ある平衡状態に到達したことが予想できる. しかも, この時のシステムの利用率は図 4.1から, ほぼ最適値 (約 88%) であることが分かる. このような複雑な振舞いを見せながらも, 進化の過程で資源共有がスムーズに行なわれる戦略を利己的なエージェント自らが発見し, その戦略によって資源競合が解決されるという結果は, エージェントの自己組織化という観点からも興味深い.

もちろん, ここで得られた結果を直接的に全ての人工生態系に当てはめることは難しいかもしれない. それは, 本稿が対象に考えているモデルが簡単すぎるものであるかもしれないし, 本シミュレーションにおける効用関数が, 偶然, 設定しやすいものだったかもしれないからである. ただ, そのような場合においても, 人間指向の選択淘汰ではなく, エージェント指向の選択淘汰を行なうことによって, システム自体を進化させる可能性は十分に示せたのではないだろうか.

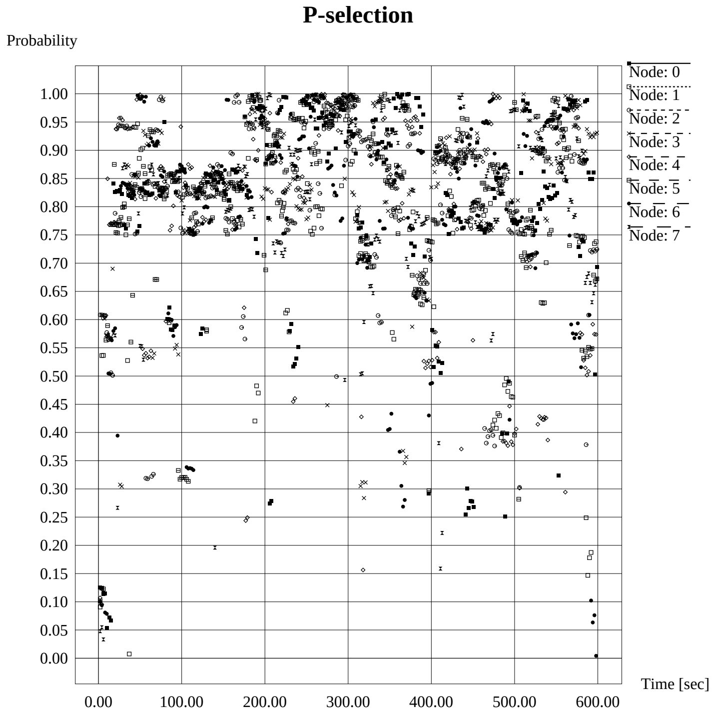

図 4.17: 義務的協調による p の選択

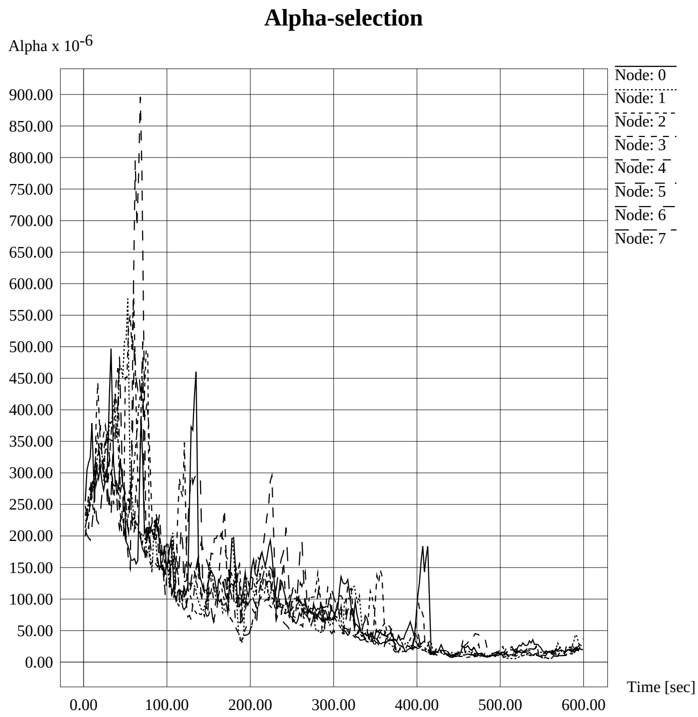

図 4.18: 義務的協調によるαの選択

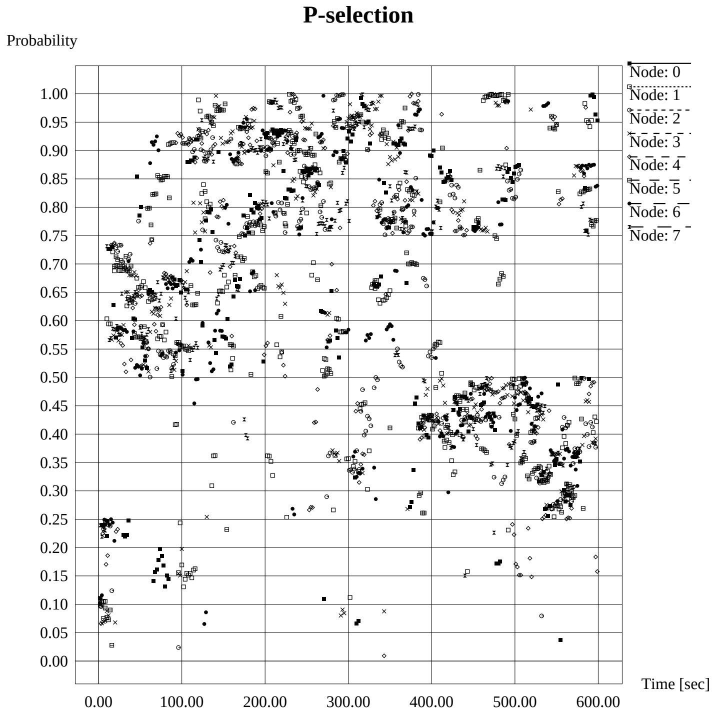

図 4.19: 利己的なエージェントによる p の選択

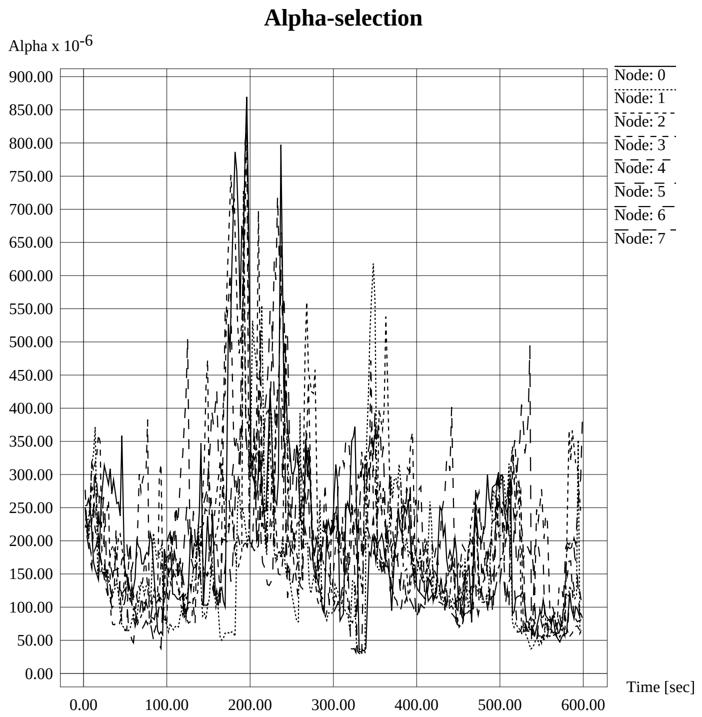

図 4.20: 利己的なエージェントによるαの選択

4.2.6　エージェントの変更に伴うコスト

エージェント指向の進化を考えるに当たり, たとえその結果が興味深く有望なものであったとしても, エージェントを選択淘汰する際に膨大なコストがかかってしまっては, その考えを一般に受け入れてもらうことは難しいだろう. そこで, コストを考慮して抜け駆けを含んだ協調のケースを例にして, エージェントを変更する際に必要なコストを調べることにする.

この場合, ノードの性能を観測したユーザは, 平均 1.0 秒の間隔でノードの性能をブローカへ報告しなければならない. さらに, 効用の傾きが負であった場合に限り, 30%の確率でノードを変更するのであるが, 図 4.21に示す通り, 12 人のプレーヤが 600 秒の間にノードを変更した回数の累計は 909 回だった (最大 85 回, 最低 64 回).

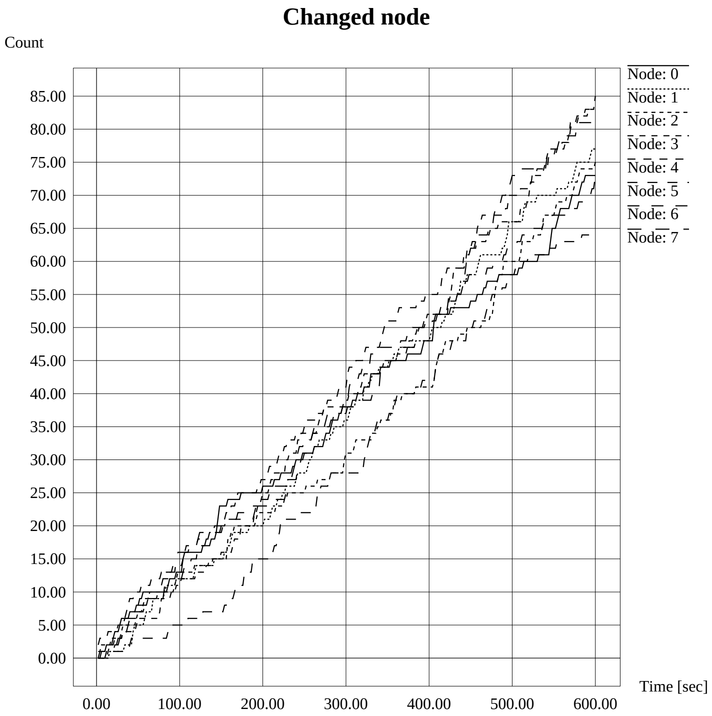

図 4.21: ノード・エージェントの交換回数 (累計)

ノードの変更のタイミングは, ほぼブローカへの報告と同期しているため[5], ノードの変更に関するブローカへの要求は, 性能報告と同じパケットを使用することができる. しかし, ブローカからの返答は別のパケットに含める必要があるため, 総計 $7200 + 909 = 8109$ 個のパケットが送受信されることになる. 仮に一つのパケットが 1000 バイトだったとしても, これらのパケットを伝送するには最低限, 108Kbps のバンド幅が必要であり, これは 10Mbps の 1%に相当する. もし 10Mbps の資源を制御するために 100Mbps のバンド

[5]タイムアウトした場合以外は完全に同期している.

幅が必要ならば, それは問題外であるが, 1%で済むのであれば, それは道理にかなった範囲と思われる.

もちろん, このバンド幅の 88%はノードの性能評価をブローカへ報告するために使用されるのであって, その頻度を下げることによって必要なバンド幅を少なくすることはできる. しかし, 報告が遅くなるほど, より大きな遅延をノードの情報に与えることになるため, その頻度はバランスを十分に考慮した上で決定しなければならないだろう.

4.2.7 平等性についての評価

これまでの設定で, プレーヤの効用関数は同一であり, 仮に各プレーヤが同量の資源を分配されたとすると, 彼らの満足度は等しく, Unilaterally envy-free の観点からも平等である. しかし, 実際には全てのプレーヤが等しい資源を得るわけではないため, その時の平等性について考察する必要がある.

図 4.22は, 図 4.14に対応する, 最終的にユーザが送信を完了したビット数を示している. これを見ると, 各ユーザが得た資源の量には差があることが分かる.

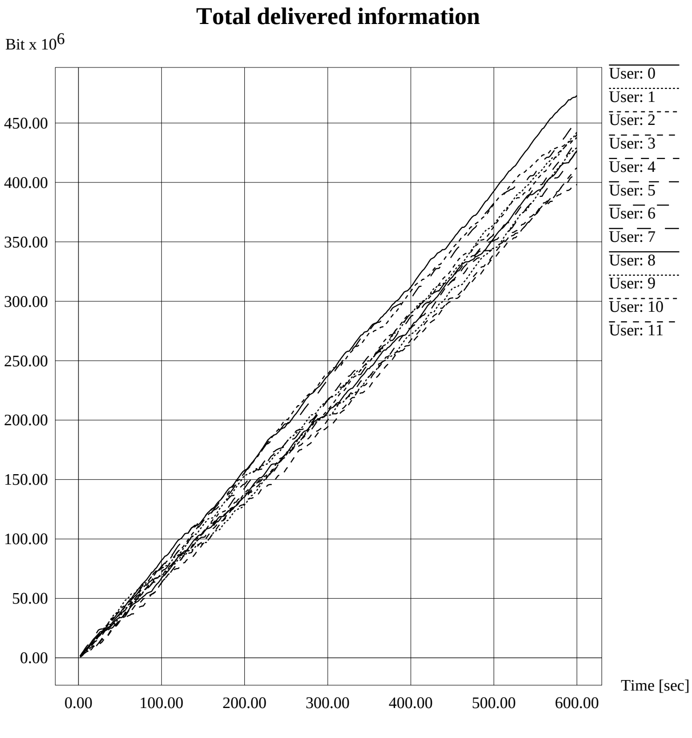

図 4.22: 各ユーザが送信を完了したビット数

では, この時のユーザの効用はどのような値になっているのだろうか. ここで, 平等性を論じるために, 各ユーザが得た転送レートと費したコストのレート, そして効用の値を

	転送レート	コストのレート	効用
平均	719K [bit/sec]	773K [yen/sec]	3.6085
偏差 σ	31504.6 [bit/sec]	30377.2 [yen/sec]	0.0340596
ゆらぎ δ	0.0437826	0.0392696	0.00943853

表 4.1: 資源とコスト, 効用の統計

プレーヤごとに平均し, その平均値を 12 人のプレーヤで比較した時の平均値と偏差 (σ), ゆらぎ (δ) をまとめ, 表 4.1に示す.

まず, 全てのプレーヤの効用関数が同一であるため, 効用にばらつきがあることは, 直接的に, これが平等な状態ではないことを表している. その不平等の具合を調べるために:

$$(\text{ゆらぎ}\ \delta) = \frac{(\text{偏差}\ \sigma)}{(\text{平均})}$$

(δ は無次元数である) の値を見てみると, 資源の分配 (転送レート, コスト) に関する見掛け上の δ に対して, それらを換算して得られる効用の δ が, ずっと小さいことが分かった. つまり, 完全に平等であるとは言えないにしても, 各エージェントの満足度のゆらぎは分配された資源とコストの見掛け上のゆらぎよりも小さく, Unilaterally envy-free の観点から平等に近い状態であると考えることができる.

4.2.8　まとめ

ここで, エージェントが使用する共有メディアのスループットに焦点を当てて, 本章で述べてきたシミュレーション結果のまとめを述べる. まず, 本章で述べた全ての場合に関するスループットの一分毎の平均値を図 4.23に示す. データセットは, それぞれ以下の場合に対応する:

- UP: コストを考慮しない利己的なエージェントの資源共有
- UPC: コストを考慮する利己的なエージェントの資源共有
- UP-ab: CSMA エージェントが必ずバックオフを行なう時の, コストを考慮しない利己的なエージェントの資源共有
- UPC-ab: CSMA エージェントが必ずバックオフを行なう時の, コストを考慮する利己的なエージェントの資源共有
- UP-mb: バックオフをしない CSMA エージェントが混ざった時の, コストを考慮しない利己的なエージェントの資源共有

- UPC-mb: バックオフしない CSMA エージェントが混ざった時の, コストを考慮する利己的なエージェントの資源共有

以上に共通する事柄は, 全てのケースにおいて, エージェントは集中制御によらず自己の判断で自己の振舞いを決定することである. CSMA エージェントがバックオフするか否かの違いは, 観察者が設定した『CSMA エージェントの機能の差』に過ぎず, いずれにせよ各エージェントは自分の判断で利己的にプロバイダを選択することに注意されたい.

さて, 本稿で注目した点は, まずエージェントにコストを考慮させることの意義であった. 一個体もバックオフを行なう CSMA エージェントがいない場合 (UP, UPC) を比較すると, 全くコストを考慮せずに性能のみでプロバイダを判断したケースは, 利己的ながらもコストを考慮してエージェントが振舞うケースと比べて, スループットが格段に低いということが分かる. つまり, コストというバックプレッシャが存在しない環境においてエージェントが利己的に振舞った場合, システムは著しい機能低下にみまわれる. この結果は『エージェントが利己的に振舞う』と仮定した時に想像できる最も典型的な (そして悲劇的な) 結末と言えよう.

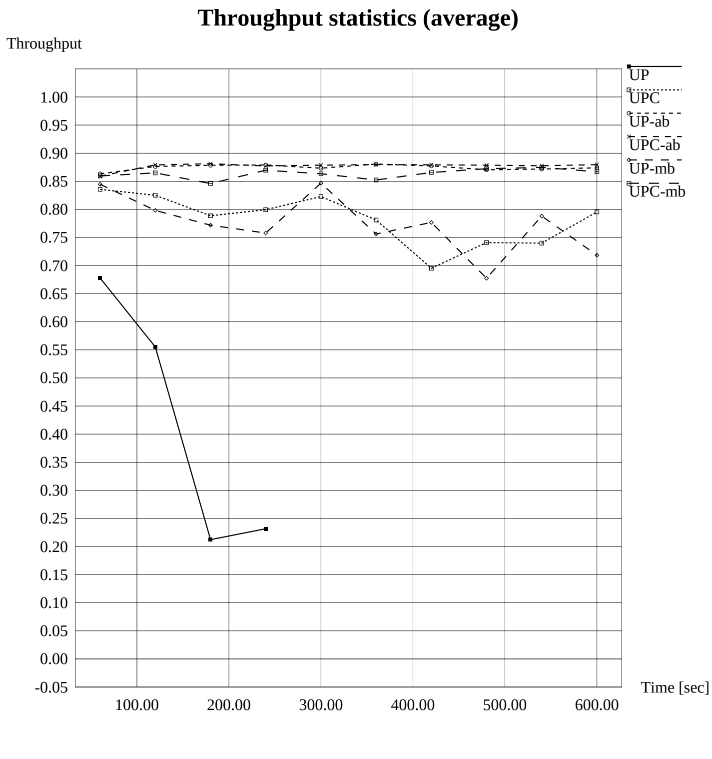

図 4.23: スループットの平均 (1 分毎の統計)

しかし, 本稿では, コストを考慮しない場合でも, システムが高い性能を安定に持続できるケースがあることを指摘した. それは CSMA エージェントが Ethernet と同様に, 必

ずバックオフを行なう場合である (UP-ab, UPC-ab). ここでは, システムの安定性を示す図として, スループットの偏差とゆらぎのグラフを図 4.24, 4.25に示す.

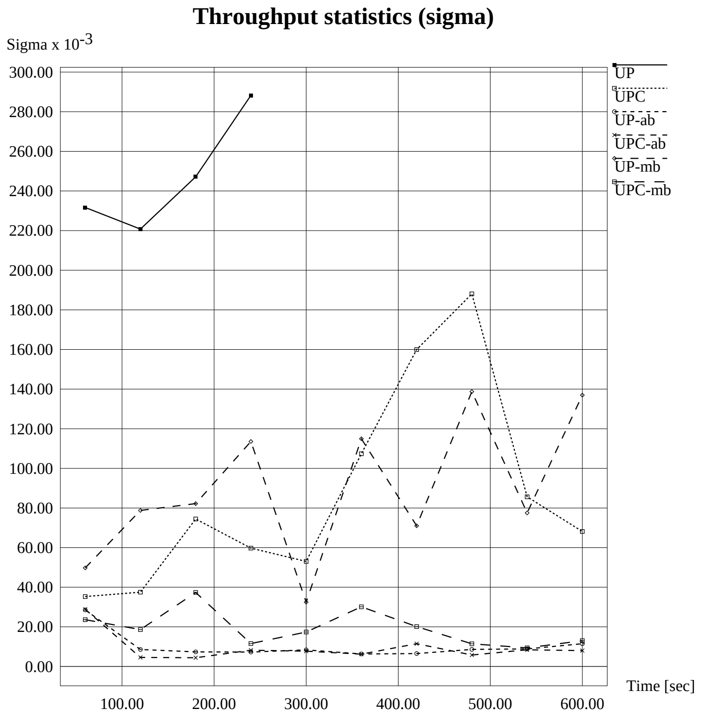

図 4.24: スループットの偏差 (1 分毎の統計)

もし, そのような約束事としての協調的動作を守れるのであれば, それはシステムにとっても, エージェントにとっても好ましいことである. しかし, そのような普遍的な約束事が仮定できない分散環境においては, 他の CSMA エージェントがバックオフをした隙を狙ってパケットを送信してしまうような利己的なエージェントを考慮せずにシステムを設計することは危険である.

そのような『抜け駆け』を行なうエージェントがいる場合 (UP-mb, UPC-mb) においても, エージェントにコストを考慮させることによってシステムが安定化した事実は, 本研究の示す結果の中でも最も意義のある結果であると思われる. 特に, 抜け駆けを含む協調のケース (UPC-mb) に関しては, 本来ならば CSMA エージェントのパラメータ p, α が, それぞれ 1 と 0 へ近づいた方が各エージェントにとっての目先の利益が高いはずであるのに, そのような直観に反して, p とα の値がそれぞれ図 4.19, 4.20で示した通りの進化を遂げたことは, システムの進化という観点からも, 極めて興味深い結果である. この時, 図 4.19において p の集団が 400 秒付近を境に 0.75 ~ 1.0 の集団から 0.25 ~ 0.5 の集団へと遷移し, その結果, システムが安定化[6]したことは, 図 4.25において, 抜け駆けを含

[6]個々のエージェントの嗜好に逆行する形で集団の遷移が発生し, それがシステムを安定化させたことは注目に値するであろう.

む協調のケース (UPC-mb) のスループットのゆらぎが, 400 秒付近を境に義務的協調のケース (UP-ab) と同等まで小さくなったことで理解できる.

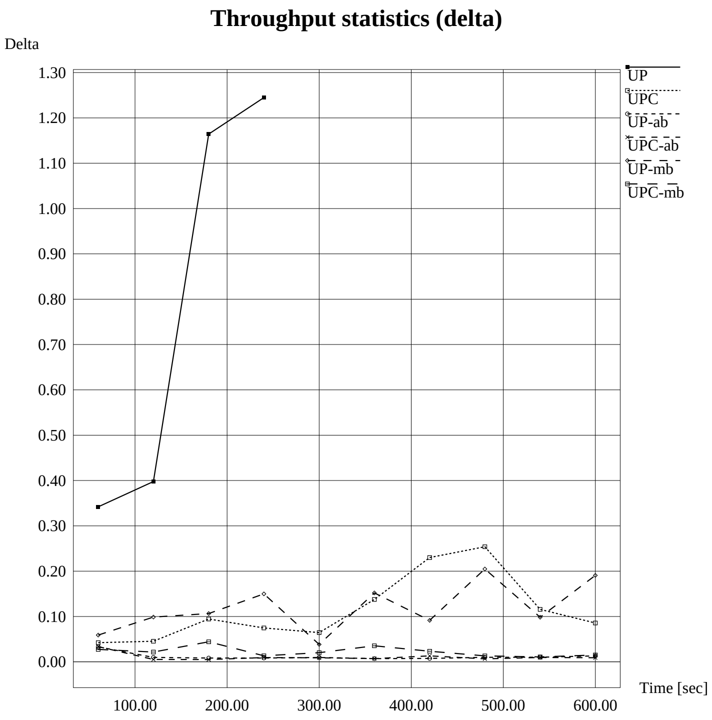

図 4.25: スループットのゆらぎ (1 分毎の統計)

ここで重要なのは, あくまでエージェントは自己の利益のみを考えて振舞っているにも関わらず, そのような協調的振舞いが一切の集中制御なくして観測されたことである. この事実は, 今後の開放型分散環境における効率的な資源分配の研究にとっても重要な意味を持つと考えられる.

第 5 章

議論

本章では, これまで特に触れてこなかった, 三つの点に関して議論を行なう. 一つ目は, エージェントの階層を利用して QoS の解釈と実行をするための抽象的アーキテクチャに関する議論であり, 二点目は, 課金スキームと価格変動についての議論である. そして最後の三点目は, 進化のモデルの妥当性に関する議論である.

5.1 複数の異なる QoS の実現

3.1節で述べたように, 本モデルは抽象的な QoS アーキテクチャを構成している. しかし, 前章において, 全てのプレーヤが持つ効用関数は同一であり, 人工生態系が異なる QoS によって選択淘汰される様子までは調べていない. 本節では, 複数の異なる QoS を設定するための準備と, 実際に複数の QoS によって人工生態系が進化する可能性について議論する.

5.1.1 異なる効用関数を持つエージェントによる資源共有

前章ではユーザの効用関数は κ のみをパラメータとした簡単なものだった. κ は秩序変数となっており, 1.0 を境にシステムの相が変化する:

- $\kappa < 1.0$
  得られる資源の量と価格の比が同じであれば, 資源は可能な限り多くの量を取った方が良い.

- $\kappa > 1.0$
  得られる資源の量と価格の比が同じであれば, 資源は可能な限り少ない量を取った方が良い.

実際, κ を 1.0 より大きな値に設定すると, プレーヤの選択は『資源を取らない』方向へ一方的に進み, 各プレーヤの転送レートは右下りに落ちてゆくことが観測できる. しかし, 転送レートがゼロまで落ちこんだとして, そのレートに価値があるかというと, 恐らくそのレートに価値を感じるプレーヤはいないはずである[1]. そこで, プレーヤへレートに関するオフセットを持つ効用関数を与えることを考えてみる.

オフセットを設定するにはシグモイド関数:

$$sigmoid(x) = \frac{1}{2} \times (1 + \tanh(\frac{x - offset}{factor}))$$

を利用すればよい. $factor$ を極めて小さな値にすることによって, 性能の価値判断に関して, オフセット以下のレートには価値がなく, オフセット以上のレートの価値は同じである, という設定ができる. もちろん, 階段的に価値を変化させることは合理的でないため,

[1] 2Kbyte のページを転送するのに 100 万年かかるような資源は価値がない.

シグモイドの傾きを適切に選ぶことで, その変化を緩やかなものにすることができる. そこでまず:

$$Utility = \frac{performance \times sigmoid(performance/10Mbps)}{cost^{\kappa}}$$

という効用関数を設定し, この効用関数によって人工生態系の振舞いを観察する. ここで, シグモイド関数内において性能を最大バンド幅 10Mbps で割っているのは, シグモイドを正規化するためである. ユーザは 2 つのグループに分割し, $offset = 0.1$ のグループ (Node:0~5) と $offset = 0.01$ のグループ (Node:6~11) を設定した. κ および $factor$ の値については, それぞれ$\kappa = 1.1$ と $factor = 0.025$ の双方のグループで共通な値とした. $offset$ の違いは, エージェントが送信レートに対して取る態度に反映される.

まず最初に, 効用関数を除いて, 前章と同じ設定でシミュレーションを行なったところ, 図 5.1に示す結果を得た. このグラフは時間に対して, ユーザが送信を完了した積算ビット数をプロットしたものである.

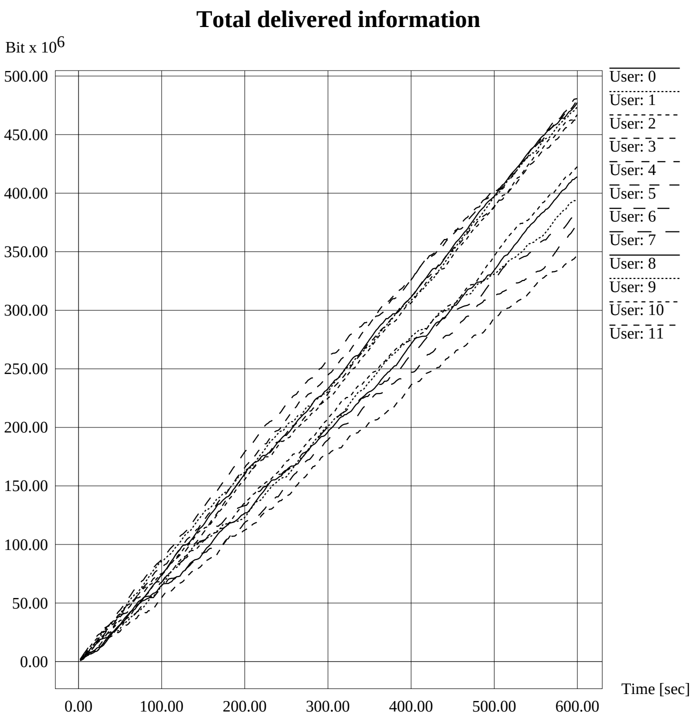

図 5.1: ユーザの異なる効用関数による選択 (短いインターバル)

プレーヤは 2 つのグループに分割されてはいるものの, 全てのプレーヤが同一のポワソンレートで送信しているにも関わらず, この結果は, 異なる効用関数を持っているプレーヤによって 2 種類の異なる性質を持ったノード (そして, 連帯的に CSMA エージェント) が選択されたことを示している. ここで注目したいのは, 2 つのグループは, 同じ生態系内

部に棲息する CSMA エージェントを選択していることである．つまり, CSMA エージェントは, 元々, 同じ遺伝子から進化しており, しかも, ブローカは 2 つのグループが存在していることは認識しておらず, 前章と全く同じメカニズムでサービスしているにも関わらず, このように 2 種類の QoS が, 結果として出現しているということになる.

5.1.2 効用関数の進化

ここで再度確認しておきたいのは, ノードは CSMA エージェントを選択しないことである．つまり, ここでユーザが選択しているのは実質的にノードではなく, CSMA エージェントであるということになる．次のステップとして, これまで全く触れてこなかったノードによる CSMA エージェントの選択を考えてみる．ノードに選択をさせるためには効用関数を設定しなければいけない．ここでは, モデルで述べた通りの効用関数を用いて, $\eta = 1.0$ の状態から突然変異を用いて η を変化させ, それをユーザに選択させるということを試みた．つまり, ノードは『遺伝的に獲得した効用関数』によって CSMA エージェントを選択することになる.

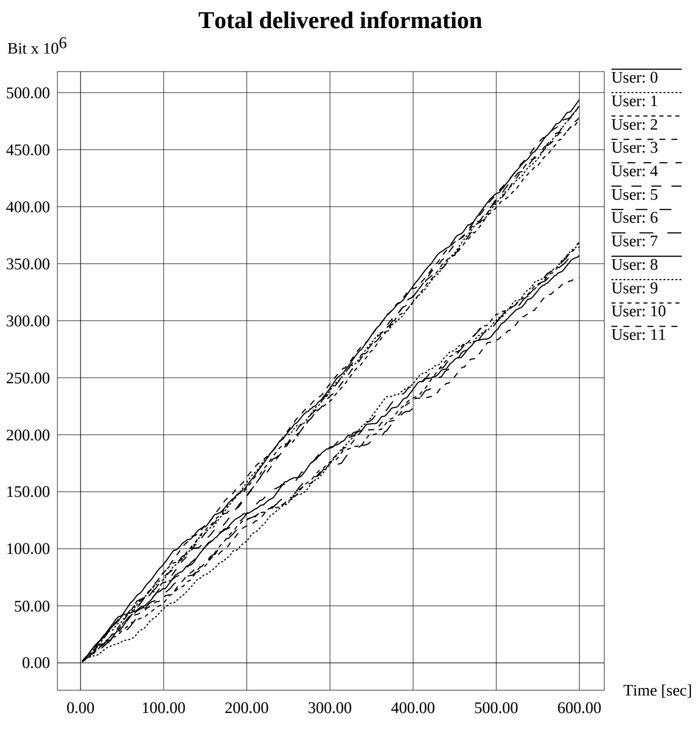

図 5.2: 効用関数の選択の効果 (短いインターバル)

最初に行なったシミュレーションの結果を図 5.2に示す．この図と図 5.1を比較すると, ユーザの選択に加えて, 効用関数を持ったノードが CSMA を選択する場合の方が, 二つ

のグループの性能の分離の度合が鋭いことが分かる. これはノードの効用関数を選択した効果であると考えることができる.

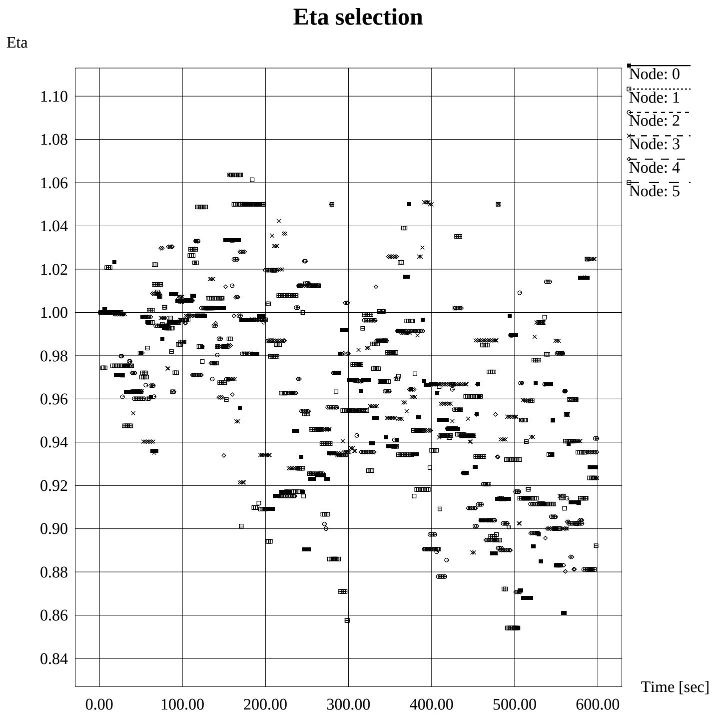

図 5.3: η の選択 (高レートのグループ)

この時, 実際に選択されたノードの η を, 二つのグループに分けて, 図 5.3と図 5.4に示す. はっきりした傾向とは言い難いが, 高レートのグループは主に$\eta < 1.0$ の値の集団を選択し, 低レートのグループは主に$\eta = 1.0$ 付近の集団を選択していることが分かる.

ノードが効用関数を持つ意味は,『ユーザに代わって CSMA エージェントを選択する』ことであると考えたい. つまり, 人間としてのユーザは, ミリ秒単位に効用をチェックすることは不可能であるから, ずっと長いインターバル (時間単位, 日単位, 月単位) で判断を下すことにし, 実際のシステムのミリ秒, マイクロ秒単位の制御は, エージェントの階層の下層のエージェントが行なうことが望ましい.

そこで, 次の段階として, ユーザが長いインターバル (10.0 秒) をおいて選択を行なう場合のシミュレーションを行なってみた (図 5.5).

この結果では, ユーザの選択の力がノードの効用関数を正しく選択できていないため, ノードが何もしない場合 (図 5.1) よりも, むしろ不安定[2] な状態が発生してしまっていることが分かる. 図 5.1は正しい (ユーザの) 効用関数によって, 短いインターバルで選択

[2]低レートのグループが, 適度なレートを与える CSMA エージェントを選択できず, 送信レートの高い状態と低い状態を繰り返している.

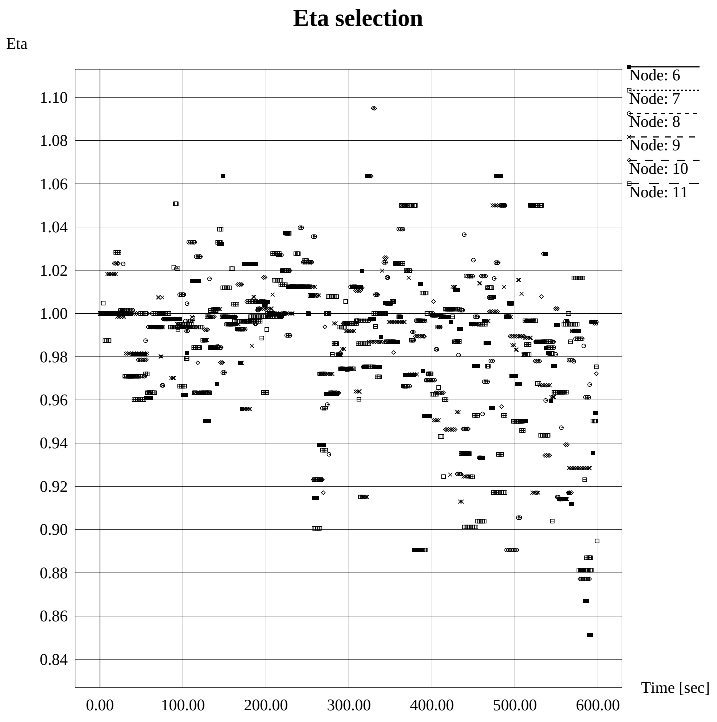

図 5.4: η の選択 (低レートのグループ)

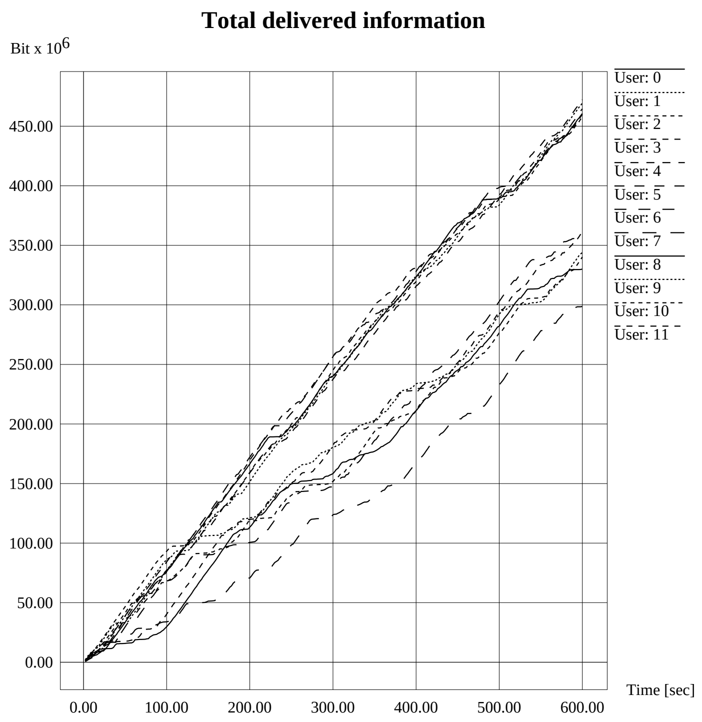

図 5.5: 効用関数の選択の効果 (長いインターバル)

を行なっているため, 一概に比較はできないが, いくつかのシミュレーションを通じて言えることは, 効用関数を選択するというメタな選択は, 現段階では非常に難しいことが分かっている. 今後の課題として, そのようなメタな選択を含めた人工生態系の振舞いを調べることが必要であると言えるだろう.

5.1.3 異種の QoS の実現

本稿で設定した QoS は, 性能の尺度として転送レートのみを扱っていた. しかし, 実際のネットワークにおいては, telnet のような端末ソフトを使用するユーザは, 転送レートより, 遅延に重みを置いてネットワークを評価する. このようなユーザは, 本稿の設定とは異なる効用関数を設定することになるので, 様々な種類の QoS が混在する人工生態系で選択淘汰が行なわれることが考えられる.

例えば, エージェントの階層に対して『遅延は大きくても良いから, できる限りバンド幅を得られるサービスが欲しい. ただし, コストは 3 円±0.5 円の範囲で実行しなさい.』という QoS 要求を送ったとき, エージェントが, 途中経路に当る回線交換リンクのバンド幅を売り買いするエージェントと交渉して, 最終的に衛星回線を使用してルーティングを行なったり, 逆に,『遅延とジッタをできる限り小さくして欲しい』という要求に対しては, それを実行するのに必要なコストを調べてユーザの所持金と比較し, 過去のユーザの要求と結果, その満足度の履歴によって, エージェントが制御の判断を行なうというような, 人間指向のネットワーク制御ではなく, エージェント指向のネットワーク制御が行なわれることが想像できる.

それほど知的なシステムではなくても, 今後の課題として, 転送レートと遅延への嗜好の違いを含んだシミュレーションを行なうことは, 異種の QoS の実現という点からも, 本研究に不可欠なものと考えられる.

5.2 Accounting に関する考察

本稿ではネットワークの利用者に対して, 利用した資源に対するコストをフィードバックするモデルに基づきシミュレーションを行なった. インターネットにおいて, このような課金を考えることは, 1章の冒頭で述べたように, 事実上, インターネットが商業利用のネットワークとして機能する状況では, 決して実現不可能なことではないと考える. なぜなら, インターネットは決して『無料』ではなく, ユーザは現在でもプロバイダに料金を支払っているからである. インターネットの統計的な資源共有による低い処理コストに見合う, ユーザが納得できる料金を設定するためにも, そのような課金体系を開拓することは必要であると考えられる.

しかし, これまでのシミュレーションは『使用した資源に応じて (消費したビット数に応じて) 課金される』という固定的な設定で行なっており, また, 効用関数に関しても『比

較問題としての』コストの項は含まれていたにしても, 現実の課金に対応できるまでのコストに対する配慮がなされていない[3]. 例えば, メディアの使用に対するコストのレートを倍に設定したとしても, これまでの効用関数を使用する限り, 実は, これまでのシミュレーション結果と, ほとんど変りないグラフが取れる. それが顕著なのは前節で設定した効用関数であり, それらが『最適なレート』を効用関数のピークとして持つため, 価格が半額になったとしても, プレーヤが好むレートに, あまり大きな変化は生じない.

価格が半額になったときに倍のレートを好むという事実は, プレーヤの可処分な財が一定であり, しかも, 価格とレートが完全に反比例しているという仮定が必要であり, 通貨供給量やプレーヤの財のような経済的な設定を厳密に行なわない限り, シミュレーションの設定としてコストを反映させることは難しい. 仮に効用関数を:

$$Utility = \frac{priceRate \times performance \times sigmoid(performance/10Mbps)}{cost^{\kappa}}$$

(ただし, シグモイドの $offset$ を $offset/priceRate$ に置き換える) という設定にしたとすれば, 価格が下った場合 (図 5.7) には, 図 5.2のグラフは, 図 5.6のように変化し, 価格に反応してプレーヤがレートを上げるという振舞いを見ることはできる. しかし, その効用関数の設定は作為的であり, 結果を評価することは難しい.

同様の理由で, 今回のシミュレーションでは『セリ (bid)』の要素を加えることができなかった. また, 異なる効用関数を持つプレーヤの満足度が公平であるか否か, という点についても, 十分な議論を行なうための材料を示すことができなかった. しかし, 本研究の現在の成果は, それらの設定を行なう以前の問題として, ユーザの効用関数が与えられた場合に人工生態系がどのように振舞うのかを調べたことにあり, 後からでも設定さえ正しく行なえば, 経済的な側面からもシステムが安定に, そして公平に振舞う可能性を示したことにある. 今後の課題として, これまで経済学者たちが研究してきた成果を本モデルに取り入れることが望まれる.

5.3 進化のモデルの妥当性

本モデルでは, エージェントの遺伝子の変異はエージェントが生存している最中に起ると設定した. 従来, 生存中の生物の形質が変化するような遺伝的変化は起らず, 生殖細胞に発生した突然変異 (と交叉) による変化のみが遺伝的形質として次世代に受け継がれる, という遺伝的機構だけが生物界を支配する法則であると信じられてきた. しかし, 最近の生物学の成果によれば, 免疫細胞に関する遺伝子は個体の生涯の間でも激しい突然変異を繰り返しており, 個体内で頻繁に発生する突然変異は, 個体の生涯内での遺伝的適応を可能にすると考えられる点で興味深い. ただ, 子孫に伝わるものという意味では, 生殖細胞で発生した変異しか遺伝しないと考えてよいだろう.

[3]しかも, 結局, 送信レートの異なる best-effort のサービスしか提供できなかった.

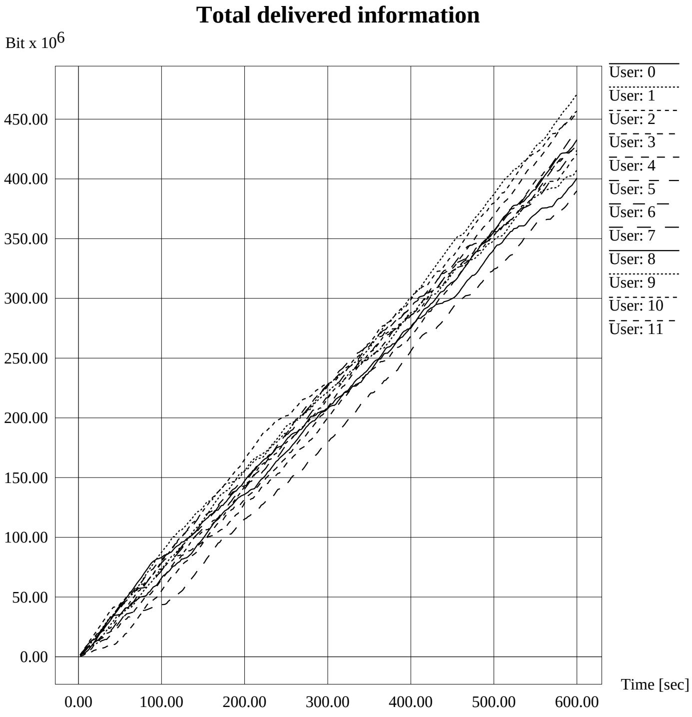

図 5.6: 価格レートを変動させた場合の資源競合

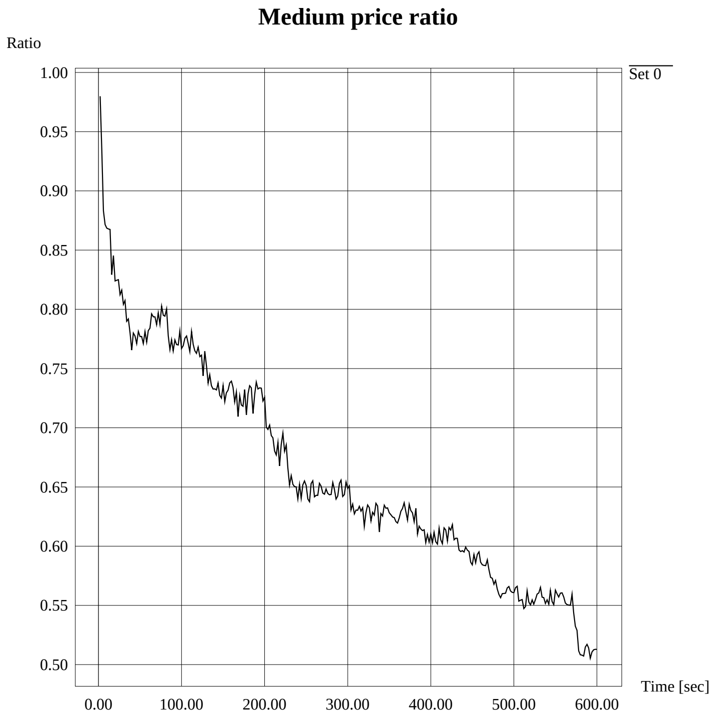

図 5.7: 価格レート

もちろん, 本モデルにおいても『自分が変えたいように遺伝子を変える』という設定は, 一切, 行なっていない. 遺伝子の選択は常に上層のエージェントが行なうのであって, どの p が良いかを決定するのは CSMA エージェントではなく, ノードや他の CSMA エージェントも含めた人工環境である. 特徴的なことを挙げれば, ノードは CSMA エージェントが具体的に何をやっているのか, どんな遺伝子を持っているのかは全く理解していない. つまり, p の選択は p を意識して行なわれている訳では全くない.

しかし, 生物界で起りえないからという理由ではなく, 工学的な進化のモデルとして, エージェントの生存中に形質が変化することは, 客観的に許されることなのだろうか? この問題は, エージェントの学習を考慮した場合に顕著に現われる. あるエージェントが学習した内容が, 次の世代に受け継がれるべきものかどうかという問題 [佐々木96] が, 本モデルでは隠された問題として存在している. 本モデルでは扱わなかったエージェントの学習は, 人工生態系におけるエージェント指向の進化というテーマとは密接な関係があり, 今後の課題として考慮しなければならない問題である.

第 6 章

結論

本稿では, コストを考慮した利己的なエージェントによる資源競合という観点から, 共有メディア型ネットワークをテストベッドとして, 複数のエージェントによって構成される人工生態系のモデル化を行なった. 本モデルにおいて, エージェントは遺伝的に獲得される戦略に従ってサービスを提供する. エージェントは特定のサービスのプロバイダであり, かつ, 別のサービスのユーザでもありうる. 各エージェントは, 利用するサービスの性能とコストを観測し, 性能とコストの関数で表される独自の効用関数により効用を計算する. エージェントは効用の時間的変化に対して独自の判断を下すことによってプロバイダを選択淘汰し, 試行錯誤を通じて自己の効用を最大化しようと振舞う. 観測されたサービスの性能とコストの情報は, 情報ブローカを仲介としてエージェント間で共有される.

本稿では, 上記のモデルを実行するシミュレータを設計・実装し, CSMA 型共有メディアにおいて複数のエージェントが利己的に振舞う環境での, ネットワーク・システム全体の進化の様子を観察した. シミュレーションの結果, 得られる結論は以下の通りである:

- コストを全く考慮しない場合でも, エージェントに協調性を強いることで, システムを安定化させることができる.
- 一方, 必ずしも全てのエージェントが協調的でない場合であっても, コストの概念を導入することによって, エージェントは協調的な振舞いを見せる集団へと自己組織化し, 結果的にシステムは安定化した.
- エージェントは, 比較的, 低いコストで自己組織化できることが分かった.
- システムが安定化した時のエージェントの効用は, Unilaterally envy-free の観点から平等に近い状態だった.
- 各エージェントが戦略を選択淘汰することによって, ネットワーク・システム自体の進化的な挙動が観察できた. その結果は, エージェントが義務として協調的に振舞う場合と, 利己的に振舞う場合とでは大きく異なっていた.
- システム自体が進化することによって, 比較的, 短い時間でシステムの性能が, 最適に近い状態へ収束することが分かった.

謝辞

本研究の機会を与えられ, 日頃から懇切な御指導を頂きました慶應義塾大学計算機科学専攻教授 所 眞理雄 博士に心から感謝いたします.

お忙しいところ本稿の査読を引き受けて下さり, 同時に, 貴重な助言を頂きました本研究室博士課程 福田 健介氏, ならびに, 本研究室博士課程 佐々木 貴宏氏, そして深夜遅くまで研究を共にした本研究室修士 2 年生をはじめ, 本研究室の諸氏に心から感謝します.

学業を支援して頂いた両親に, そして論文執筆期間中, 精神的な支えとなって頂いた多くの友人に感謝します. また, 論文執筆期間中のストレス解消に効果を発揮してくれたゲーム "DESCENT" の作者に感謝し, ストレスの捌け口となって破壊されていった無数のロボット達に[1], この場を借りまして, 謹んで哀悼の念を表します.

最後に, いつも心の片隅で私を支え続けてくれた種ともこさんの美しい歌声と, その詩を通して私に多くのインスピレーションを与えてくれた種ともこさんに, 心から深く感謝し, 永遠の愛を誓いたいと思います.

1997 年 2 月
成田多良

[1]赤ハルクにロックされた瞬間がたまらない.

参 考 文 献

[佐々木96] 佐々木 貴宏．エージェントの進化における学習の効果と遺伝機構に関する研究．1996．

[所95] 所 真理雄．マルチエージェント研究の目指すもの．コンピュータソフトウェア，12(1)，January 1995．

[北野93] 北野 宏明．遺伝的アルゴリズム．産業図書，1993．

[And83] Andres G. R. and Schneider F. B. Concepts and Notations for Concurrent Programming. *ACM Computing Surveys*, 15(1), 1983.

[Apo86] Apostolopoulos Theodore K. and Protonotarios Emmanuel N. Queueing Analysis of Buffered CSMA/CD Protocols. In *IEEE Transactions on Communications, Vol. COM-34, No.9 September 1986, 898-905*, 1986.

[Bai95] Bailey Joseph P. Economics and Internet Interconnection Agreements. *MIT Workshop on Interent Economics*, 1995.

[Ben85] Benda M., Jagannathan V. and Dodhiawalla R. *On Optimal Cooperation of Knowledge Sources.* Technical Report, Boeing AI Center, 1985.

[Bog88] Boggs David R., Mogul Jeffrey C. and Kent Christopher A. Measured Capacity of an Ethernet: Myths and Reality. In *SIGCOMM '88*, 1988.

[Bre95] Breyer Roobert and Riley Sean. *Switched and FAST Ethernet: How It Works and How to Use It.* Ziff-Davis Press, 1995.

[Cam93] Campbell Andrew, Coulson Geoff and Hutchison David. A Multimedia Enhanced Transport Service in a Quality of Service Architecture. In *Fourth International Workshop on Network and operating System Support for Digital and Audio and Video*, 1993.

[Cla95] Clark David D. A Model for Cost Allocation and Pricing in the Internet. *MIT Workshop on Interent Economics*, 1995.

[Coc91] Cocchi Ron, Estrin Deborah, and Shenker Scott, Zhang Lixia. A Study of Priority Pricing in Multiple Service Class Network. In *SIGCOMM '91*, 1991.

[Coy83] Coyle Edward J. and Liu Bede. Finite Population CSMA/CD Networks. In *IEEE Transactions on Communications, Vol. COM-31, No.11 November 1983, 1247-1251*, 1983.

[Cub95] Cubert Robert M. and Fishwick Paul. *Sim++ Version 1.0.* 1995.

[Dav87] Davis, L. *Genetic Algorithms and Simulated Annealing.* Pitman, 1987.

[Dig80] Digital Equipment Corporation, Intel and Xerox. *The Ethernet, A Local Area Network: Data Link Layer and Physical Layer Specifications (Version 1.0).* 1980.

[Est90] Estrin Deborah and Zhang Lixia. Design Considerations for Usage Accounting and Feedback in Internetworks. In *ACM Computer Communication Review, 20(5)*, October 1990.

[Fis] Fishwick Paul. SIMPACK: Getting started with simulation programming in C and C++.

[Hog91] Hogg Tadd and Huberman Bernardo A. Controlling chaos in distributed systems. *IEEE Transactions on Systems, Man, and Cybernetics*, 1991.

[Hub88a] Huberman Bernardo A. *The Ecology of Computation.* Elsevier Science Publishers B.V., 1988.

[Hub88b] Huberman Bernardo A. and Hogg Tad. The Behavior of Computational Ecologies. *Elsevier Science Publishers B.V.*, 1988.

[Hub95] Huberman Bernardo and Clearwater Scott H. A Multi-Agent System for Controlling Building Environments. In *ICMAS '95*, 1995.

[Jac88] Jacobson Van. Congestion Avoidance and Control. In *SIGCOMM '88*, 1988.

[Joh96] Johnson Howard W. *Fast Ethernet, Dawn of a New Network.* Prentice Hall, 1996.

[Kle75] Kleinrock Leonard and Tobagi Fouad A. Packet Switching in Radio Channels: Part I – Carrier Sense Multiple Access Modes and their Throughput-delay characteristics. In *IEEE Transactions on Communications, Vol. COM-23, No.12 December 1975, 1400-1416*, 1975.

[Koz92] Koza J. R. *Genetic Programming.* MIT Press, 1992.

[Koz94] Koza J. R. *Genetic Programming II.* MIT Press, 1994.

[Lan89] Langton Christopher G. *Artificial Life.* Addison Wesley, 1989.

[Lan95] Langton Christopher G. *Artificial Life: An Overview.* MIT Press, 1995.

[Loe92] Loepere Keith. *Mach 3 Kernel Interfaces.* Open Software Foundation and Carnegie Mellon University, 1992.

[Mac95] MacKie-Mason Jeffrey K., Murphy Liam and Murphy John. The Role of Responsive Pricing in the Internet. *MIT Workshop on Interent Economics*, 1995.

[McK95] McKnight Lee W. and Bailey Joseph P. An Introduction to Internet Economics. *MIT Workshop on Interent Economics*, 1995.

[Nak91] Nakajima T., Tokuda H., Ashida N. and Arakawa H. *Real-Time Mach 3.0 User Reference Manual.* Department of Computer Science Carnegie–Mellon University, May 1991.

[Pos80] Postel Jon. User datagram protocol. In *DARPA Internet Request For Comment 768*, 1980.

[Pos81] Postel Jon. Transmission control protocol. In *DARPA Internet Request For Comment 793*, 1981.

[She94] Shenker Scott. Making Greed Work in Networks: A Game-Theoretic Analysis of Switch Service Disciplines. In *ACM SIGCOMM, Computer Communications Review, Vol.24, No.4, October, 1994*, 1994.

[Tak85] Takagi Hideaki and Kleinrock Leonard. Throughput Analysis for Persistent CSMA Systems. In *IEEE Transactions on Communications, Vol. COM-33, No.7 July 1985, 627-638*, 1985.

[Tan89] Tanenbaum Andrew S. *COMPUTER NETWORKS SECOND EDITION.* Prentice Hall, 1989.

[Tas86] Tasaka Shuji. Dynamic Behaviour of a CSMA-CD System with a Finite Population of Buffered Users. In *IEEE Transactions on Communications, Vol. COM-34, No.6 June 1986, 576-586*, 1986.

[Tok91] Tokoro Mario. Toward Computing Systems for the 2000's. In *Operating Systems of the 90s and Beyond,* LNCS 563, Springer-Verag, July 1991.

[Top90] Topolcic C. Experimental Internet Stream Protocol, Version 2 (ST-II). In *DARPA Internet Request For Comment 1190*, 1990.

[Zha93] Zhang Lixia, Deering Stephen, Estrin Deborah and Schenker Scott and Zappala Daniel. RSVP: A New Resource ReSerVation Protocol. In *IEEE Network*, 1993.

付録 A

シミュレータの設計と実装

A.1 SimFrame 概論

3 章で設計したシミュレーションモデルは, 本章で述べるシミュレーション・フレームワークとそれに基づく汎用シミュレータ **SimFrame**[1]を利用することにより, 実際に計算機上で実行される. SimFrame は本稿で独自に設計および実装を行なった完全にオリジナルなシミュレータであり, 既存のシミュレータのコードやライブラリなどは一切使用していない.

SimFrame はオブジェクト指向のイベント・ドリブン型シミュレータを実装するためのフレームワークであり, 実際にシミュレーションを行う上で必要となる中核的な機能を実装した汎用ライブラリを提供する. 各シミュレーションのインプリメンタは, SimFrame のフレームワークに従って SimFrame プリミティブを利用することにより, 容易に各種シミュレーションを実装することが可能である.

SimFrame の基本的な構成要素は, 主に次の 3 つに分類される (図 A.1).

1. イベント・スケジューラ
   イベント・ドリブン型スケジューラ. イベントの管理, および, そのスケジューリングを行なう.

2. イベント・オブジェクト
   オブジェクト間の通信に用いられる特殊なオブジェクト. イベントが連鎖的に発生することにより, シミュレーションが遂行される.

3. オブジェクト
   SimFrame はオブジェクト指向シミュレータであり, オブジェクトがシミュレーションにおける基本的な実行単位となる.

A.2 フレームワーク

本節では SimFrame のフレームワークと, その実装ストラテジを概観する. SimFrame はオブジェクト指向のシミュレータであり, 全ての処理はオブジェクトを単位として実行される. ただし, これは現実のシステムのコーディングがオブジェクト指向言語で行なわれていることを必ずしも意味しない.[2] オブジェクト間の相互作用は通常のコーディングにおけるメソッド呼出しが基本であり, これは, イベントの存在を考慮しなければ, シミュレータの実装が, そのまま実際のシステムの実装の一部となりうることを示している.

[2]SimFrame は蓜秀久氏 (慶應) によって C++に移植された. しかし, SimFrame のオリジナルは C 言語で実装されており, 本稿のシミュレータもオリジナルの SimFrame に従っている.

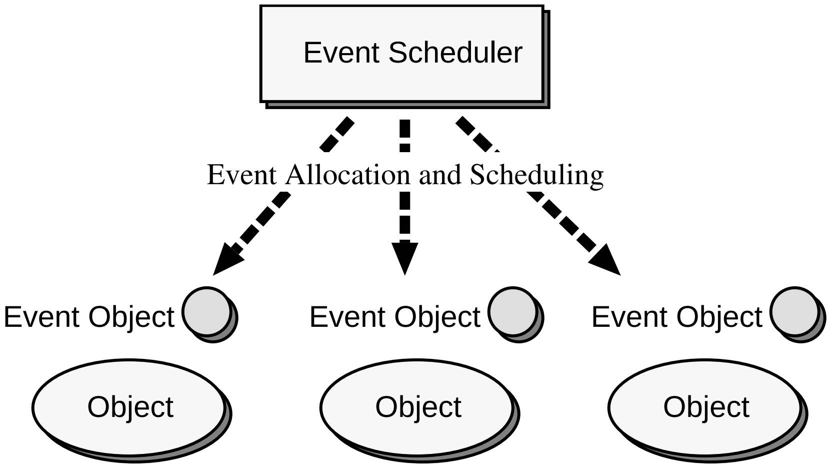

図 A.1: SimFrame の構成要素

A.2.1 イベント

SimFrame が提供するフレームワークの中で, シミュレータを設計する上での最も重要な概念はイベントであり, その配置方法とオブジェクトの設計方法がシミュレータの機能および性能を決定する. SimFrame におけるイベントは, 常にオブジェクトに対するメソッド呼出しと関係づけられて配置される.

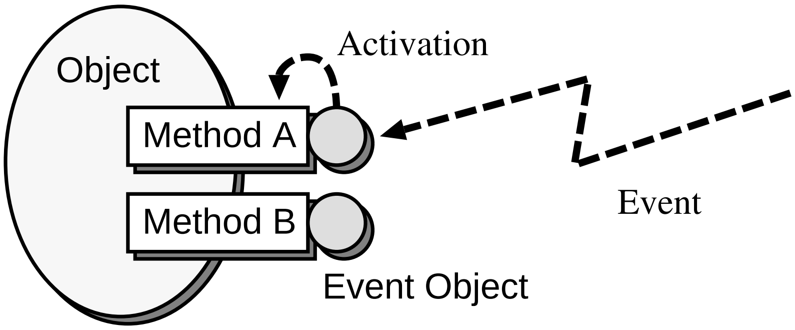

図 A.2: イベントとメソッド呼出し

図 A.2 に示すように, イベント・オブジェクトは特定のオブジェクトの特定のメソッドに対して配置する. イベント・オブジェクトは特定のイベントに対する監視役の役目を果し, そのイベントが発生した時には, これを受けとめて, 自動的に特定のメソッドを呼び出すという動作を行う. これによりオブジェクトの設計者は, イベントの監視についてのコーディングを省略することができるようになる.

あるイベントに対応するメソッドは唯一であるが, 逆に, あるメソッドに対応するイベ

ントは複数個, 配置できることが望ましい. 例えば, 複数の要求を発行したオブジェクトが, それぞれの要求に対して同種のイベント (終了の合図といったイベントなど) を受け取りたい場合, 一つのメソッドを複数のイベント・オブジェクトが起動できるようにしたい. そこで, 複数のイベントをメソッド側で判別するために, SimFrame ではイベント・オブジェクトに対して必ず名前[3]を割当てるようにしている.

さらに, 一つのイベント・オブジェクトが複数個のイベントを監視できることが望ましい. 例えば, あるオブジェクトの要求が正常に終了した場合や, 異常終了した場合などの, 複数の理由に対して別々のイベントを配置することは, オブジェクトの設計者にとって大きな負担となる. そこで SimFrame では, イベント・オブジェクトは複数のイベントを同時に待つことができ, そしてメソッドを呼出す際には発生したイベントの名前をメソッドへ通知できることにした.

イベントに関する付随的な事項として, SimFrame ではイベント・オブジェクトのタイムアウトを指定することができる. ある時間だけイベントを待ち, その期間にイベントが送られてこない場合, 待っていたイベントの代りにタイムアウトのイベントを発生させる. タイムアウトは他のイベントと同様に扱うことが可能であるため, イベント・オブジェクトのタイムアウトは容易に実装することができる.

以上によりオブジェクトは:

- どのイベント・オブジェクトが自分を呼出したのか?
- どのイベントが発生したのか? あるいは発生しなかったのか?

を判別することができるようになる.

A.2.2 イベントによる同期

オブジェクトがイベントを受けとる機構は上述の通りであるが, ここでは, イベントを発生させる側の機構について触れることにする. まず, イベントを発生させるためには, それを通知する相手としてイベント・オブジェクトが必要である[4]. そのため, イベントの送信者は, 前もって受信者からイベント・オブジェクトの名前を知らされていることが前提となる. イベント・オブジェクトの名前は必ずしも直接, 当事者間で受け渡しする必要はない. 中間にネームサーバを介在させれば, イベントを送るために必要な情報は必要になった時に獲得することができる. イベントの宛先さえ分かれば, イベントはいつどこで送信しても構わない. イベントの送信は, それ自体, イベントとして実装されているため, タイムアウトを設定することで, イベントを送信する時刻を指定することが可能である.

イベント・オブジェクトの名前は通常のメソッド呼出しで渡すのが一般的である. 図 A.3に示したように, オブジェクト A はオブジェクト B に処理を要求する時, 直接的なメ

[3] SimFrame では名前として整数のインデックスを割当てている.
[4] イベントをブロードキャストする場合に関しては, この限りではない.

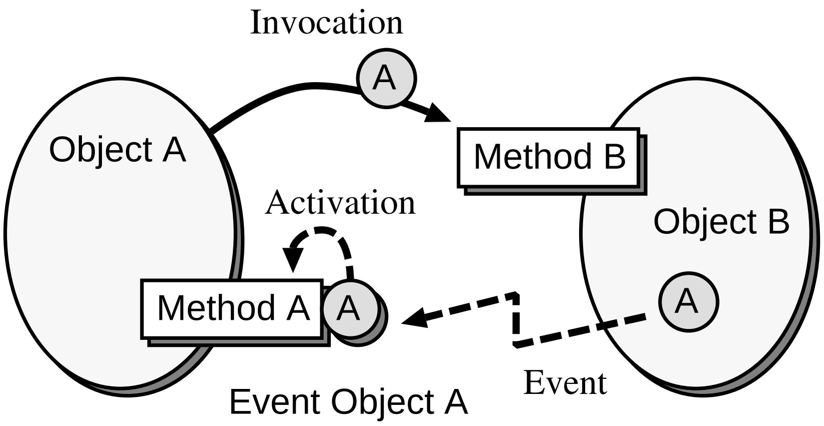

図 A.3: イベントによる同期

ソッド呼出しに付随させてメソッド A に配置したイベント・オブジェクトの名前 “A” をオブジェクト B に引き渡す．これは名前を渡すことが目的であるため, そのメソッドは実際に要求された処理をせずに終了する．オブジェクト B は “A” という名前を記憶し, 実際に処理が済んだところで “A” に対して処理が終了した旨のイベントを送ることになる．オブジェクト A の側から見れば, 最初のメソッド呼出しが終った時点ではなく, オブジェクト B がイベントを送信した段階で, その処理が終了したことを通知されるということである.

このように, イベントにはオブジェクト間の同期を取るという性質があり, イベントによる同期は, 現実のシステムの処理の連続性 (コンティニュエーション) を 2 つに分割するという重要な役割を果す．一般に, オブジェクト A がオブジェクト B に要求を送った後は, A は B から何らかのイベントが送られるのを待つため, A の実行はそこで中断される．A の処理が再開した時には, シミュレーション上の時刻は進んでいなければならないが, その時刻は B がイベントを発生させた時刻のはずである．SimFrame における時間の管理において, イベント・オブジェクトを配置し, イベントを待つことによって時間が進むという点で, イベントによる同期は SimFrame の核となる機能である．一方, この同期が現実の実装とシミュレーションを隔てることになるわけであるが, 逆に, イベントの存在以外, 両者の間に実装上の大きな違いはないと考えることもできる.

A.2.3 イベントのスケジューリング

SimFrame における時間の進み具合を支配しているのはイベントである．ここでは図 A.4を例にして, イベントの配置とスケジューリング, そして時間の概念について述べる.

まず初期状態として 2 つのオブジェクトが存在し, 時刻 t0 には 2 つのイベントが配置されている．オブジェクト A の初期イベントは時刻 t1 に, オブジェクト B の初期イベン

トは時刻 t2 にタイムアウトする設定になっているものとする. この状態でイベント・スケジューラを一回起動することにより, スケジューラの管理する時刻は t0 から t1 へと進み, 同時にオブジェクト A のイベントがタイムアウトする.

この際, 仮に Group A のオブジェクトの状態に変化がなかったとして, Group A では時刻 t4 にタイムアウトするイベント・オブジェクトを再び配置し, あるイベントを待つことにする. 次にスケジューラを起動すると, 今度は Group B のイベントが時刻 t2 にタイムアウトし, オブジェクト B の処理が始まる. ここでオブジェクト B がオブジェクト A の状態に変化を及ぼし得るような処理, 例えば共有資源の先取りを行ったとする. そこで B は A に対して preemption のイベントを送信して, その事実を A に通知する.

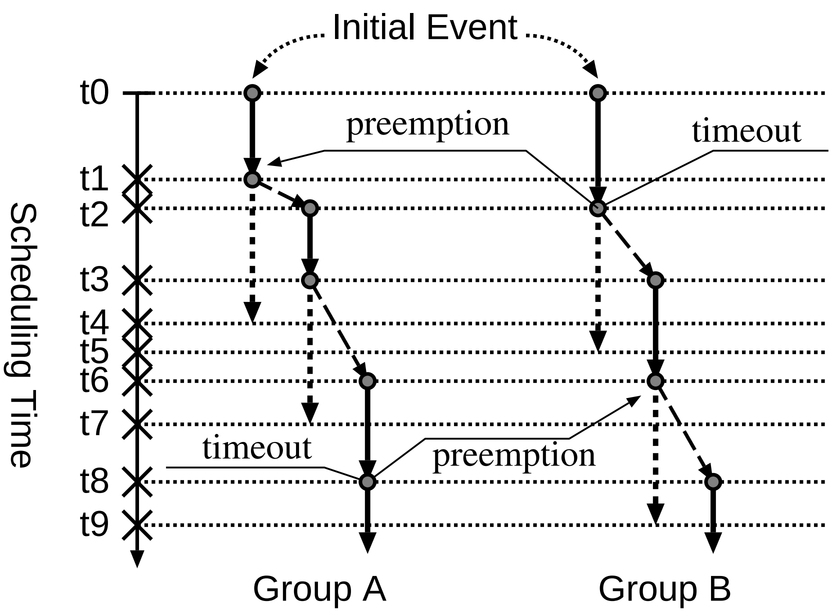

図 A.4: イベントのスケジューリング

この通知を行うイベントは, 概念的には B から直接 A に届くものであるが, 実際には次にスケジューラを起動した際に, 以前と同じ時刻 t2 に A に通知される. ここで A は時刻 t1 にて配置した t4 でタイムアウトすることになるイベント・オブジェクトをキャンセルし, 新たに t3 にタイムアウトするイベントを配置するとする.

さらにスケジューラを起動すれば, 時刻は t3 に進み, オブジェクト A がオブジェクト B に preemption のイベントを送り, オブジェクト B が新しいイベント・オブジェクトを配置し, 次々とイベントが配置されながら時刻が進んでゆくことになる.

SimFrame の世界における『時刻』は, スケジューラが持つ 1 つの double 型変数に集約されている. スケジューラはスケジューリング・キューで待っているイベント・オブジェクトの列の中で最も近い将来に起るべきイベントを取り出しては, 自分の時刻を進め (あるいは, 進めずに), そのイベントを発生させてゆく. そして呼び出されたオブジェ

クトが状況に合せて新しいイベントを配置し, それらが連鎖的にスケジューリングされてゆく. スケジューラのキューにイベント・オブジェクトが入っていなければ, それはシミュレーションが終了したことを意味している.

A.2.4 スケジューラ

本稿は SimFrame の実装そのものを主たるテーマとするものではないため, SimFrame の内部について詳しく説明することはしない. ここでは SimFrame のスケジューラのデータ構造を概説し, スケジューラの輪郭のみを示す.

まず, フレームワークで述べたイベント・オブジェクトは, 双方向ポインタで結ばれたスケジューリング・キューを形成する. スケジューリング・キューはイベントのタイムアウトの時刻順に並んでいる. このためスケジューラは常にキューの先頭のオブジェクトを取り出し, オブジェクトのデータフィールドに格納されている情報を元にしてイベントを発行するだけでよい (図 A.5).

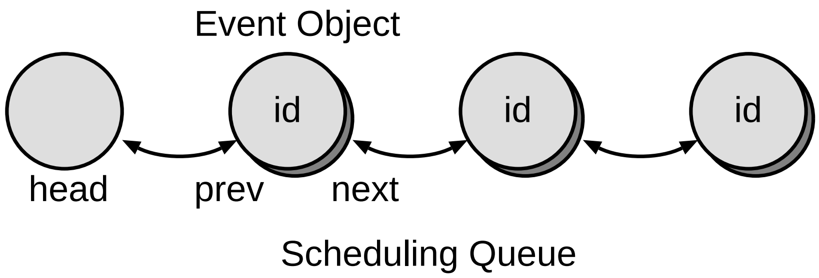

図 A.5: イベント・オブジェクト

イベント・オブジェクトは名前 (整数の id) でアクセスされるため, 名前から参照への変換が必要となる. SimFrame ではハッシュを用い, イベント・オブジェクトの名前に対する剰余の値をインデックスとする, 双方向のポインタで結ばれたハッシュ・キューを用意している. ハッシュ・キューに連結されたキュー・オブジェクトは, それぞれ正しい名前のイベント・オブジェクトへのポインタを持ち, 逆にイベント・オブジェクトは自分を指すキュー・オブジェクトに対するポインタを持っている. 名前から参照へ変換するには, まず id にハッシュ操作を施し, 数百個の配列となっているハッシュの先頭からキューを手繰り, ポインタが指し示すイベントの id を調べてゆく. しかし, 配列のサイズが大きいため, ほとんどの場合, 一回のアクセスでイベント・オブジェクトへ到達できる (図 A.6).

さらに各イベント毎に, そのイベントを待つイベント・オブジェクトを指し示すブロードキャスト・キューを持ち, それらのキュー・オブジェクトに対して, 逆にイベント・オブジェクトから自分を指すキュー・オブジェクトに対する別のキュー, 逆引きキューが双

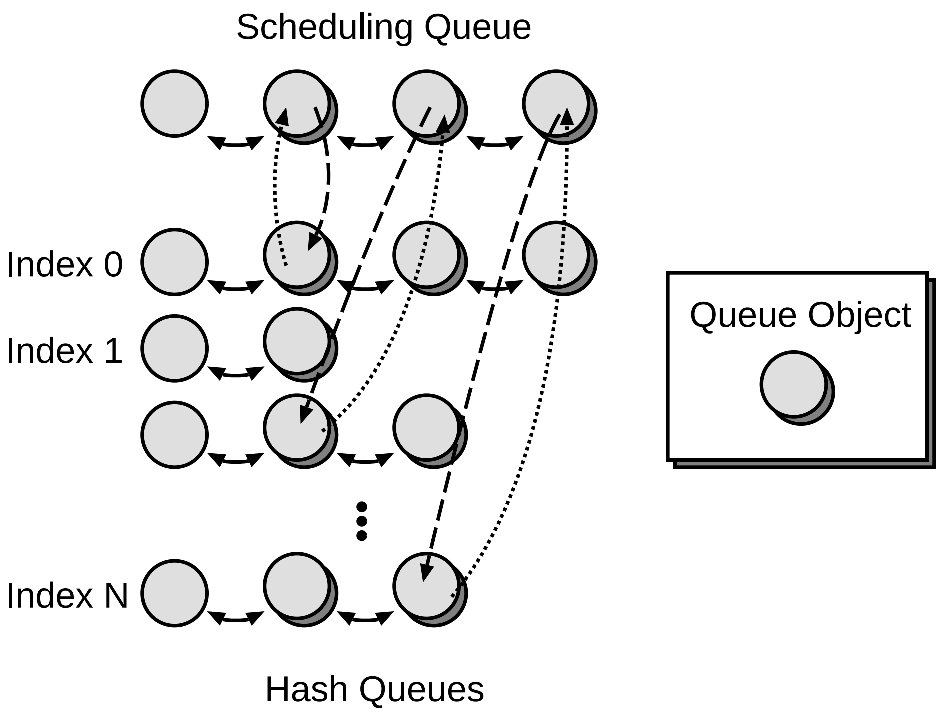

図 A.6: ハッシュ・キュー

方向ポインタで結ばれている[5] (図 A.7).

以上のデータ構造を配置することにより, これまでに述べてきたフレームワークを実現し, しかも, それらを高速に処理することが可能となっている.

A.3 プリミティブ

本節では, 前節で概説した SimFrame のプリミティブの一部 (リスト A.1) を列挙し, 実際に本稿のシミュレータで利用しているコードに近い形で, 簡単に使用方法を示すことにする.

A.3.1 イベントの配置

本稿で使用するシミュレータの最上部には User 層があり, User 層ではポワソン到着するパケットを Node に送信するという動作を繰り返している. この繰り返し動作は最も簡単なイベントの使用法の例であり, イベントを配置してタイムアウトを待ち, タイムアウト・イベントの発行に伴って再び同じイベントを配置するという動作を繰り返せば良い.

[5]この図では, 逆引きキューのオブジェクトを省略し, 両方向の矢印のみで示している. 実際には縦方向にもキュー・オブジェクトの列ができることになる.

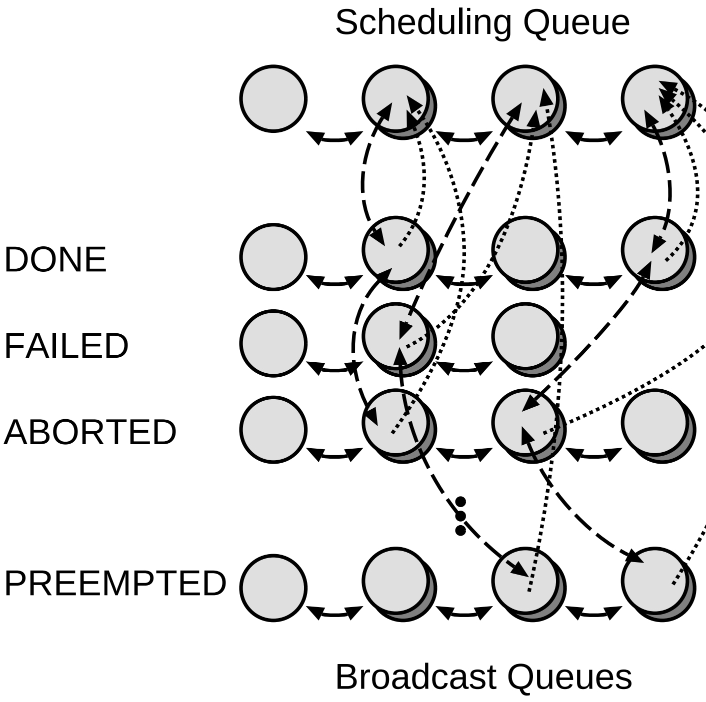

図 A.7: ブロードキャスト・キューと逆引きキュー

イベントの配置には, SimFrame のプリミティブ `EventWait()` を使用する. タイムアウト時刻は現在の時刻 `EventNow()` よりも `PoissonInterval()` だけ後の時刻に設定する. この動作は毎回, 同一であるため, イベントのフック関数として定式化する (リスト A.2).

`UserHookPoissonEvent()` によってフックされたメソッド `UserAction()` (9 行目) は, 時刻 `timeout` に次のようなシグネチャをもって呼出される.

```
public void UserAction( int id, void *ref, int reason );
```

`id` はイベント・オブジェクトの名前, `ref` はオブジェクトへの参照, `reason` は発行したイベントの名前である.

`UserAction()` の内部では, イベントのキャンセル (8 行目), Node からの結果報告のためのイベントの配置 (15 行目), バッファの配置とキューへの挿入 (20 ~ 24 行目), Node への出力 (26 行目), そして最後にフック関数の呼出し (36 行目) を行なう[6] (リスト A.3).

`UserHookPoissonEvent()` におけるイベントは, 繰り返しを目的としたもので, タイムアウト以外のイベントを待つことはない. しかし, `UserAction()` で配置するイベントは, Node からの結果報告を待つためのイベント・オブジェクトであるため, それぞれの結果

[6]つまり, このメソッドはスケジューラが動きつづける限り, 永久に呼出され続ける.

```
/* 現在の時刻を得る．*/
public double EventNow();

/* timeout に発行するイベントを生成する．cond はイベントの OR */
public int EventWait( int cond, double timeout,
                      action_function_t foo, void *ref );

/* イベントを deallocate する．*/
public boolean_t EventCancel( int id );

/* 特定の相手にイベントを送る．*/
public int EventSend( int id, int cond, double timeout );

/* cond を待つ全てのイベントに cond をブロードキャストする．*/
public int EventBroadcast( int cond, double timeout );
```

リスト A.1: SimFrame のイベント・プリミティブ

```
private void UserHookPoissonEvent( user_t *user )
{
  double timeout;

  timeout = EventNow()
          + PoissonInterval( user->lambda, user->tau );
  user->event = EventWait( TIMER_EXPIRE,
                           timeout,
                           UserAction,
                           (void *)user );
}
```

リスト A.2: フック関数

```
public void UserAction( int id, void *ref, int reason )
{
  user_t *user;
  buffer_t *buf;

  user = (user_t *)ref;

  EventCancel( id );

  switch( reason ){
  case TIMER_EXPIRE:
    /*
     * activate a node
     */
    id = EventWait( DONE | FAILED | ABORTED,
                    HUGE_VAL,
                    UserNodeReport,
                    (void *)user );

    buf = BufferAlloc( user->pacLen );
    buf->userEvent = id;
    buf->userTime  = EventNow();

    UserEnqueueBuffer( user, buf );

    if( FALSE == NodeInput( user->node, buf ) ){
      EventCancel( id );
      buf = UserDequeueTail( user );
      BufferDealloc( buf );
    }
  }

  /*
   * set up the next arrival
   */
  UserHookPoissonEvent( user );
}
```

リスト A.3: イベント・オブジェクトの配置と名前の受け渡し

に対応する複数のイベントを待つことにしている. 対照的なのは, こちらはタイムアウトを設定しない[7]ことである.

イベントはキャンセルしない限りスケジューリング・キューに残り, イベントの発生を監視し続ける. そのため, タイムアウトが来るまでは, 同じイベント・オブジェクトに対して, 何度でもイベントを発生させることが可能である.

A.3.2　イベントの送信

リスト A.3において, Node は NodeInput() によって User が配置したイベントの名前を受け取る. この名前を利用して, Node は User に結果を報告することができる.

```
EventSend( node->buf->userEvent, DONE, EventNow() );
```

この例は, 今すぐにイベント DONE を送る, という意味であるが, 適切なタイムアウトを設定することにより未来にイベントを送信することも可能である[8].

A.3.3　イベントのブロードキャスト

イベントを送信するためには送り先を指定する必要があるが, 一度に大勢の相手にイベントを送る場合, 全てのイベント・オブジェクトを列挙して EventSend() を発行するのはオブジェクトの設計者にとって大きな負担である. 本稿のシミュレータにおいても, 共有バスの利用に際してパケットの衝突が発生した場合や, キャリアが停止して送信可能な状態になった場合などに, 当事者全員に一度にイベントを送りたい場面が生ずる. そこで SimFrame では, イベント・オブジェクトを指定せずに, 特定のイベントを待つ全てのイベント・オブジェクトに対してイベントをブロードキャストできる機構を用意した.

```
timeout = EventNow() + mediumEpsilon;
EventBroadcast( READY, timeout );
```

この例では, ある伝播遅延 mediumEpsilon だけ待って, 共有バスを監視している全てのオブジェクトにイベント READY をブロードキャストしている. このように, タイムアウトを用いることで, イベントが相手に伝わるまでの伝播遅延を表現することも可能となっている.

[7]実際には HUGE_VAL を設定する.

[8]現在の SimFrame の実装では, イベントを送る際に, 特別な引数を送ることはできない. しかし, この例で言えば, User が生成した buf オブジェクトの中に値を格納することで, 間接的に渡すことはできる.

A.3.4 状態遷移マシン

オブジェクトの設計は, それ自体がシミュレータの設計であり, 特に定まった方法論はない. ただし, イベントの使用を考慮すれば, これまで述べてきたストラテジに加えて, SimFrame におけるオブジェクトの設計には, いくつかの共通点が挙げられる. その一つとして, 各オブジェクトは状態変数を保持し, 発行されたイベントにより次々と状態を変化させる状態遷移マシンとして捉えられる点が挙げられる. 本稿のシミュレータにおける CSMA エージェントも, イベントによる状態遷移マシンとして実装されている (図 A.8).

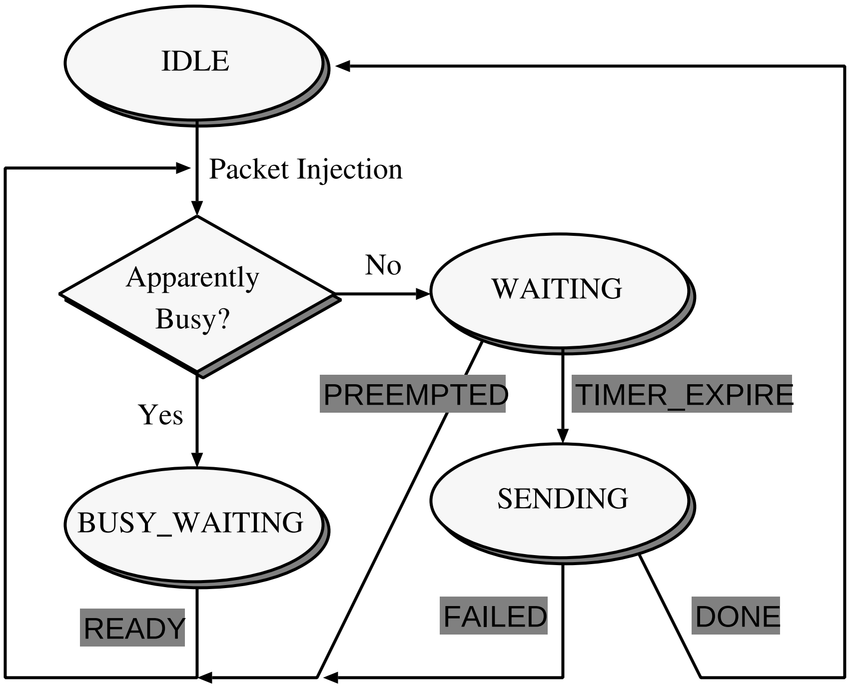

図 A.8: CSMA エージェントの状態遷移図

このように, オブジェクトの状態をダイアグラムの形で整理しておけば, イベントによる状態遷移を別々のメソッドではなく, 状態遷移専用のメソッドで受けとめることができるようになり, プログラムの見通しもずっと良くなる.

A.4 関連研究

フリーウェアのシミュレーション・パッケージとしては SimPack[Fis][Cub95] が有名である. SimPack は SimFrame と同様にイベント・ドリブン型のシミュレータであるが, オブジェクト指向ではない上に[9], イベントの取り扱いも安易なものである.

[9] C++版の SimPack ですら, オブジェクト指向とは言いがたい.

SimFrame におけるイベントの扱いは Mach オペレーティングシステム [Loe92][Nak91] における並行プログラミング [And83] に着想を得ている. しかし現在の実装ではマルチプロセッサ上で実行する際の性能に問題があり, 今後, 拡張を行なうとすれば, 相互排除プリミティブを準備し, 逐次的処理を並列化する点が考えられる.