博 士 論 文

無線ネットワークにおける
クロスレイヤ設計に基づく QoS フレームワーク

香川大学大学院　工学研究科
信頼性情報システム工学専攻

森　慎太郎

要　旨

本研究では，アプリケーション層から物理層までの Quality of Service（QoS）情報を統合的に取り扱うことを目的として，クロスレイヤ設計に基づく QoS フレームワークを提案する．具体的には，複数のプロトコルレイヤ間での QoS 情報の共有手法，クロスレイヤ設計に基づく適応制御手法，ヘテロジニアスネットワークへの対応手法，新たな計算機シミュレータの実装，および実装したシミュレータを用いた提案手法の評価である．

第 1 章では，提案手法が必要とされている背景について述べる．具体的には，近年，多様な形態のモバイル機器の普及に伴い，無線ネットワークを介したアプリケーションサービスは爆発的に増大している．一方，無線ネットワークにおける通信は，マルチパス伝搬に起因する固有の問題を有する．そのため，このような動的に変化する無線ネットワーク環境において，多彩なマルチメディアサービスを提供させる場合であっても快適な通信環境を実現させるためには，無線通信制御を適応的に行う必要がある．そこで，特定の無線ネットワークシステム，または特定のプロトコルの改善を目的とするのではなく，将来の無線ネットワークシステムに幅広く導入することを目的とした，QoS フレームワークが必要不可欠である．

第 2 章では，提案するクロスレイヤ設計に基づく QoS フレームワークに対するスキームを述べる．従来手法と提案手法の大きく異なる点は，Cross-layer converter を経由して受け取ったユーザの所望 QoS 情報に基づき，Cross-layer optimizer が適応的な無線伝送制御を実現する点である．また，ユーザの所望 QoS に基づく適応的な無線伝送制御を実現するためには，プロトコルレイヤごとに定義・分類が異なる QoS 情報の関係を明確にして，プロトコルレイヤ間で情報共有する必要がある．その QoS 情報を対応づけるために，提案手法では Cross-layer converter を導入する．

第 3 章では，クロスレイヤ設計に基づく適応的な無線伝送制御について述べる．とくに，適応パケット長制御および適応レート制御に焦点をあて，無線 LAN に導入する場合を想定して理論的な解析を行う．具体的には，MAC プロトコルの設計手法，パケット遅延の定義，適応制御パラメータの決定手法，適応パケット長制御および適応レート制御のアルゴリズムを提案する．

第 4 章では，複数の無線ネットワークシステム間で QoS 情報を共有するために必要である QoS 情報管理手法を述べる．具体的には，QoS 情報を QoS ポリシーとして規格化した上で，ブローカを通して管理データベースを用いてシステム間で共有する．その際に必要なネットワークプロトコルを，Session Initiation Protocol（SIP）を用いて設計する．提案手法を導入することにより，ユーザが他の無線ネットワークシステムに移動した場合であっても，その共有した QoS 情報に基づき適応的な無線伝送制御が可能になる．

第 5 章では，提案手法を評価するために，新たに計算機シミュレータを実装する．このシミュレータでは，将来の無線ネットワーク環境に幅広く対応できるようにするために，新たな無線通信モデルを作成する．その無線通信モデルに基づき提案クロスレイヤ適応制御を解析するために，C++ 言語を用いてデータリンク層および物理層のプロトコルスタックを実装する．また，トランスポート層およびネットワーク層のプロトコルスタックは，既存の広域ネットワークシミュレータ ns2 を用いて TCP/IP および UDP/IP を実現する．また，提案手法のシミュレーション結果より，単独のフローを伝送するとき，TCP および UDP に対して，パケット遅延は，ともに，50.0%，スループットは，各々，30.6% および 31.3% 改善できる点，2 本のフローを混在して伝送するとき，TCP および UDP のフローに対して，パケット遅延は，各々，33.2% および 33.3%，スループットは，各々，68.7% および 55.2% 改善できる点を確認する．さらに，クロスレイヤ設計をシステムに導入する際のオーバヘッドを加味しても，ネットワーク特性を改善できることを示す．

第 6 章では，本論文を結論付け，今後の課題を議論する．

目　次

第３章　クロスレイヤ適応制御

第４章　QoS 情報の管理手法

図番号 一覧

表番号 一覧

記　号

C	通信路容量
D	パケット遅延時間
D^*	ユーザの所望パケット遅延時間
D_T	パケットを伝送するために必要な時間
$D_{T,SAW}$	SAW-ARQ を用いた場合のパケット伝送遅延時間
$D_{T,SR}$	SR-ARQ を用いた場合のパケット伝送遅延時間
D_T^*	ユーザの所望パケット伝送遅延時間
J	最大フレーム再送回数（初回送信も含む）
k	畳込み符号の拘束長
L	フレームの全長
L_H	フレームヘッダの全長
L_P	フレームのゼロパディング領域の全長
M	多値変調方式における変調多値数
p_b	ビット誤り率
p_f	フレーム誤り率
R	符号化率
$T_{(N)ACK}$	*ACK* または *NACK* を伝達するために必要な時間
$T_{Backoff}$	バックオフ時間
T_{DIFS}	DIFS 時間
T_{SIFS}	SIFS 時間
U	ネットワークを流れるトラヒック量
v	データ伝送レート
V	シンボル伝送速度
x_h	パケットの全長
δ	フレーム分割に起因するオーバヘッド
$\overline{\delta}$	フレーム分割に起因するオーバヘッド（フレーム誤り率を考慮）
δ_{SAW}	SAW-ARQ を用いた場合のフレーム分割に起因するオーバヘッド
δ_{SR}	SR-ARQ を用いた場合のフレーム分割に起因するオーバヘッド
γ_s	Signal to Noise Ratio（SNR）
μ	フレーム分割数
ρ	無線ネットワークの帯域占有率

略　語

3GPP	3rd Generation Partnership Project
AAA	Authenication, Authorization and Accounting
AMC	Adaptive Modulation and Coding
ARQ	Automatic Repeat Request
ATM	Asynchronous Transfer Mode
AWGN	Additive White Gaussian Noise
BPSK	Binary Phase Shift Keying
CDMA	Code Division Multiple Access
CDN	Contents Delivery Network
CoA	Care-of Address
CRC	Cyclic Redundancy Check
CSI	Channel State Information
CSMA/CA	Carrier Sense Multiple Access with Collision Avoidance
D2D	Device-to-Device
DCF	Distributed Coordination Function
DIFS	Distributed Coordination Function Interframe Space
EDCA	Enhanced Distributed Channel Access
FA	Foreign Agent
FEC	Foprward Error Correction
FPLMTS	Future Public land Mobile Telecommunications Systems
HA	Home Agent
HCCA	HCF Controlled Channel Access
HLS	Home Location Server
HSDPA	High Speed Downlink Packet Access
HSS	Home Subscriber Server
HSUPA	High Speed Uplink Packet Access
HTTP	Hyper Text Transfer Protocol
ITU	International Telecommunicationi Union
LAN	Local Area Network
LSP	Label Switched Path

LTE	Long Term Evolution
MCS	Modulation and Coding Scheme
MPLS	Multi-Protocol Label Switching
NGN	Next Generation Network
PCF	Point Coordination Function
QAM	Quadrature Amplitude Modulation
QCI	QoS Class Indicator
QoE	Quality of Experience
QoS	Quality of Service
QPSK	Quaternary（Quadrature/Quadriphase）Phase Shift Keying
RACF	Resource and Admission Control Functions
RSVP	Resource Reservation Protocol
RTS/CTS	Request to Send/ Clear to Send
SAW-ARQ	Stop-and-wait ARQ
SIFS	Short Interframe Space
SIP	Session Initiation Protocol
SM	Superposition Modulation
SMTP	Simple Mail Transfer Protocol
SR-ARQ	Selective Repeat ARQ
TOS	Type of Service
UHF	Ultra High Frequency
UWB	Ultra Wide Band
VHF	Very High Frequency
VoD	Vide on Demand
VoIP	Voice over IP
XML	Extensible Markup Language

第１章
緒　言

通信は人間が社会生活を営むうえで必要不可欠な要素である．多様な通信環境（いつでも・どこでも）にいるユーザが，多様なモバイルサービス（誰でも・何でも）を高い信頼性を持ってストレスなく享受したいという人類の夢は，移動通信技術の急速な発展・普及によって今では誰でも手の届く範囲に入り，人々の生活習慣を大きく変えるまでの影響力をもつに至った．無線ネットワークシステムの発展の初期段階では，有線ネットワークシステムではカバーすることができない領域に対して，モバイルサービスを提供することが大きな目的であった．一方，無線ネットワークシステムは，伝送速度や信頼性の面において有線ネットワークに近づきつつある今日，無線通信固有の移動性などの利点に注目が集まり，情報収集・配信などの多様なシステムの担い手となっている．サービスの質的な面では，モバイル端末の小型化・軽量化，無線ネットワークシステムが提供するアプリケーションサービスの多様化，利用者層の拡大・利用目的の多様化に伴い，有線ネットワークシステムと並ぶ重要かつ独立した通信ネットワークシステムへの位置づけがなされている．

今後，無線ネットワークシステムの用途が増大し，社会全体の通信需要の中でさらに大きな役割を果たすことが予想される．このような状況において，ユーザの希望に沿った様々なアプリケーションサービスを提供するために，ユーザの所望 Quality of Service（QoS）に応じて，柔軟なアプリケーションサービスのを提供し，かつ適応的な無線伝送制御を実現する必要がある．

本研究では，アプリケーションサービスに対する QoS 情報を統一的に取り扱うことを目的とした QoS フレームワークの提案，およびクロスレイヤ設計に基づく適応的な無線伝送制御法を検討する．具体的には，複数のプロトコルレイヤ間における QoS 情報共有手法の提案，クロスレイヤ適応制御手法の提案，ヘテロジニアスネットワークへの導入に必要なプロトコル設計手法，提案手法を評価するために必要な計算機シミュレータの実装，および提案手法の評価を行う．

1.1 背　景

1.1.1 モバイル通信の概況

近年，社会・経済活動の高度化・多様化を背景に，マルチメディアコンテンツを提供するモバイルサービスの利用が拡大傾向にある．それに加え，携帯電話，スマートホン，タブレット端末など，インターネット接続を重視した多様な形態のモバイル端末が一般に普及してきたことに起因して，無線ネットワークを用いたアプリケーションサービスは爆発的に増大している．今日では，所有したくないという強い意志をもつ人を除いて，すべての人が当たり前のようにモバイル端末を所有し，本格的なモバイルサービスを利用することができる時代になった．既に実用化されているモバイルサービスには，例えば，ハイビジョン映像コンテンツの配信，映像教材のストリーミング配信，大容量データ伝送を伴う家電機器との連携などが挙げられる．また，最近では，移動性だけではなく位置情報サービスなどの無線ネットワークシステムの特徴を活かして，無線分散ネットワークを基盤インフラとした，モバイルクラウドサービスに対する需要も高まっている．このほかにも，大容量デジタルサイネージ情報の配信，医療画像伝送による遠隔医療などの新たなサービスが登場するなど，無線ネットワークシステムが取り扱うコンテンツが多種多様になり，かつ大容量化が進むことが想定される．このような状況に対応するために，さらなる高速・大容量，かつ利便性の高い無線ネットワークシステムの実現が必要不可欠である．

また，無線ネットワークのブロードバンド化が進展・普及し，大容量なマルチメディアコンテンツを用いた多彩なアプリケーションサービスの提供が行われることにより，さらなるネットワークトラヒックの増大が見込まれる．将来のネットワーク環境の展望に関する報告 [1] によると，2010 年から 2015 年までの 5 年間で，インターネット全体では年率 32%, 無線ネットワークに限定すると年率 92% の進行度にてネットワークトラヒックが増加することが予測されている．とくに，今後増加するネットワークトラヒックは，従来の音声トラヒックとは異なり，インターネット接続を利用したマルチメディアトラヒックが大部分を占める．従って，今後，多様な分野においてモバイルサービスの利用が促進した場合，マルチメディアトラヒック量の増加が著しく，さらに新たなモバイルサービスが拡大していくことを加味すると，この傾向は加速していくことが推測できる．

他方，今日では，将来の無線ネットワークシステムに求められている要求に対し，既にあらゆることが実現可能な無線ネットワークシステムにおける要素技術が研究され，出尽くしたのではないかと錯覚してしまうほどの技術革新を達成している．しかし，本当に必要とされていることが実現されているかというと必ずしもそうではない．例えば，無線ネットワークにおいてもブロードバンド化は進展しているが，利用可能な無線周波数資源を潤沢に利用できるわけではないので，有線ネットワークシステムとは異なり，無線ネットワークシステムが利用可能な通信帯域は制限されている．また，無線ネットワークシステムの通信環境では，無線チャネルの電波伝搬状況が時々刻々と変化するため，最高性能がうたわれている最新の無線ネットワークシステムであっても，必ずしも高速データ伝送を保証できていない．従って，確実性を突き詰めた場合，無線ネットワークシステムは，まだ改善の余地が多数存在している [2]．以上のことから，将来的には，より一層，社会の需要に対して的確に対応した無線ネットワークシステムを構築することが求められる．

1.1.2　無線ネットワークシステムの標準化動向

無線ネットワークシステムの中核である無線アクセス技術には，主にセルラシステムと無線 Local Area Network（LAN）システムがある．セルラシステムは，1953 年に横浜と神戸での 150 MHz 帯港湾電話，1964 年に日本沿岸での内航船舶電話，1965 年に新幹線での列車公衆電話，1979 年に自動車電話を起源とした，広域モバイルサービスである．一方，無線 LAN システムは，有線 LAN システムを無線化することを目的として，1997 年に IEEE 802.11 標準規格を踏襲する構内モバイルネットワークサービス，またはホームネットワークサービスである．図 1.1 に示すように，無線ネットワークシステムは，そのデータ伝送速度が有線ネットワークシステムに近づきつつある中，多種多様なアプリケーションサービスを無線ネットワークシステムに収容し，急激なネットワークトラヒック増大に対処するために，国際標準化団体，ならびに各国・各地域の標準化機関などが密接に連携している．従って，無線ネットワークシステムを検討する場合，標準化動向は無視することができない．そこで，本節の残りの部分において，セルラシステムおよび無線 LAN システムに対する標準化動向について概観する．

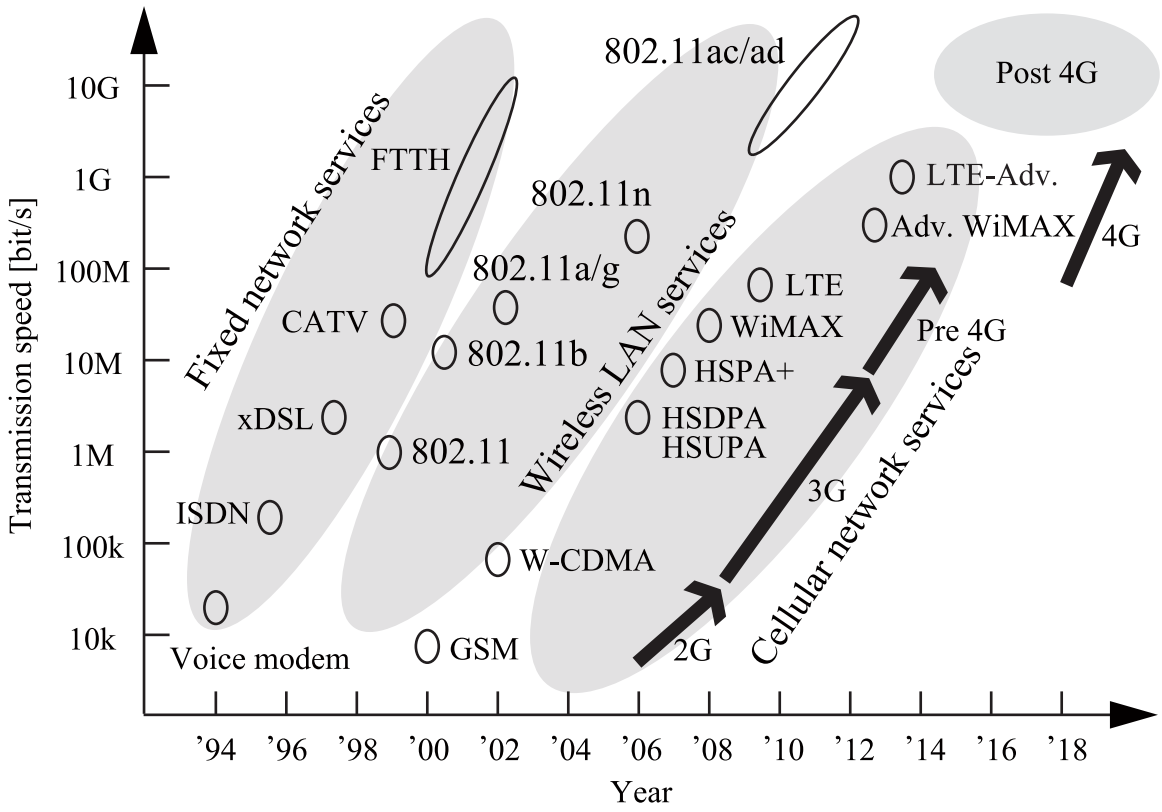


図 1.1　無線ネットワークシステムの変遷

セルラシステム　セルラシステムにおける国際標準化の動きは，1980 年代半ばごろより，International Telecommunication Union（ITU）[3] の主導により，Future Public Land Mobile Telecommunications Systems（FPLMTS）として，後に IMT-2000 に名称変更され標準化活動が行われた．IMT-2000 の基本概念は，無線通信を有線通信と同程度の品質で提供することを目指し，かつ異なる無線ネットワークシステムを切れ目なく統合することにある．すなわち，高速インターネット接続サービスの提供，利用者が世界のどこに移動してもサービスが受けられる国際ローミングサービスの実現である [4]．具体的には，米国，欧州，日本を中心として 10 のシステムが提案され，第 3 世代セルラシステムとして，IMT-CDMA-DS，IMT-CDMA-MC，IMT-CDMA-TDD，IMT-TDMA-SC，IMT-FDMA/TDMA，IMT-OFDMA/TDD-WMAN が 1999 年に仕様化された [5].

第 3 世代セルラシステムのうち，商用化の中心となったのは，国際標準化団体 3rd Generation Partnership Project（3GPP）・3GPP2[6] が検討した Release 99/4

W-CDMA 方式と cdma2000 方式である．とくに 3GPP では，Release 5 HSDPA，Release 6 HSUPA，Release 7/8 HSPA とは異なり，第 4 世代セルラシステムへのスムーズな移行を見据え，第 3 世代セルラシステムの長期発展 3rd Generation-Long Term Evolution（3G-LTE）と呼ばれる抜本的な通信方式の検討を継続した．そして，Release 8/9 LTE として 2009 年に仕様化，日本では 2010 年に実用化された [7].

また，ITU において，IMT-2000 の高度化を目的として，IMT-Advanced と呼ばれる第 4 世代セルラシステムの検討が行われた [8][9]．これを契機として，3GPP において LTE-Advanced[10]，IEEE において IEEE 802.16m[11] の検討が行われ，前者は LTE-Advanced，後者は Wireless MAN Advanced として 2011 に仕様化された [12]. 3GPP における LTE-Advanced に関しては，Release 10 LTE-Advanced として 2011 年に仕様化され，Release 10 LTE-Advanced を補完した Release 11 LTE-Advanced が 2012 年に仕様化されている．ただし，2014 年現在において，日本では実用化には至っていない [13].

現在の標準化活動においては，第 4 世代セルラシステムを拡張することを目的として，Release 12 Beyond LTE-Advanced の検討が継続的に行われている [14]．とくに，急増するネットワークトラヒックに対応するために，基地局がカバーするエリアの大きさが異なる複数のネットワーク構成をもつヘテロジニアスネットワークへの対応が重要とされている．ヘテロジニアスネットワークでは，周波数が低い電波をマクロセルに，周波数が高い電波をスモールセルに用いる．その理由は，高い周波数の電波は低い周波数の電波と比較して広帯域なデータ伝送帯域を確保することが可能であるが，電波の到達範囲が狭い欠点がある．そこで，ヘテロジニアスネットワークは，広範囲をカバーできるマクロセルと，高速なスモールセルを組み合わせることにより，両者の欠点を補完し合っている．

無線 LAN システム　無線 LAN システムは，LAN の標準化機関である米国 IEEE の 802 委員会の配下にある 802.11 ワーキンググループが標準化活動を行っている．1997 年に IEEE 802.11 規格が仕様化された後，物理層の高速化を目的として，最大 11 Mbit/s のデータ伝送速度を実現する 2.4 GHz 帯を用いた IEEE 802.11b が 1999 年に仕様化された．また，5 GHz 帯を用いた 20 Mbit/s 以上のデータ伝送速度を実現する IEEE 802.11a が 1999 年に仕様化され，日本では電波法の制約を受け，IEEE

802.11j として IEEE 802.11a を部分修正した独自規格が策定された．

IEEE 802.11b との後方互換性を担保しつつも 20 Mbit/s 以上の高速データ伝送を実現する IEEE 802.11g が 2003 年に仕様化された．今日では，IEEE 802.11b/g は Wi-Fi の名称にて，世界中で幅広く利用されている．一方，ネットワークトラヒックの増大に対処するために，2.4 GHz 帯と 5 GHz 帯を用いて 100 Mbit/s 以上のデータ伝送速度を実現する IEEE 802.11n が 2009 年に仕様化された．IEEE 802.11n は，新たな無線アクセス技術を導入しながらも，IEEE 802.11a/b/g に対する後方互換性に配慮されている．

現在の標準化活動においては，要素技術の革新にとどまらず，広帯域な無線伝送帯域を確保することが可能なミリ波における検討が継続的に行われている．とくに，1 Gbit/s 以上のデータ伝送速度を実現するために，IEEE 802.11n に後方互換性を持たせ 5 GHz 帯を用いた IEEE 802.11ac，60 GHz 帯を用いた IEEE 802.11ad が 2012 年に仕様化された．IEEE 802.11ac/ad に関しては，高速無線 LAN システムに対する社会的ニーズの高まりをうけ，仕様化には至っていない Draft 規格の段階であっても，2011 年には実用化された．

1.1.3　無線ネットワークシステムの通信環境

無線ネットワークシステムは，空間を電波伝搬させることによりデータ伝送を実現している．空間を伝わる電波は，アンテナから空間に送信されてからの伝搬特性に大きく影響されるのは当然ながら，送信環境や受信環境の影響を受けやすい特徴がある．従って，無線ネットワークシステムを設計する場合は，有線通信では想定されない無線通信固有の問題を考慮する必要がある．電波伝搬特性は，電波の周波数と地理的要因によって大きく異なる．そこで，本節では，今日のモバイル通信の主戦場である VHF/UHF 帯（30 ～ 3,000 MHz）の無線通信環境について概観する．

理想的な無線伝搬環境を想定すれば，送信アンテナから送信された電波は自由空間を経て受信アンテナに届くが，現実的には何らかの障害物の存在を考えなければならない．すなわち，アンテナ同士を直接的に伝搬する直接波以外に，屋外では建物による反射波，回折波，散乱波，屋内では壁を通り抜ける透過波がある．従って，様々な伝搬路を経由して受信アンテナに到達することにより，時間，振幅，位相が異なる到来波が合成されて受信される．また，伝搬路に存在する障害物が人や自動車のように

移動する場合，これらの移動による伝搬環境の変動が電波伝搬特性に影響を与える．従って，無線ネットワークシステムの通信環境において，無線信号は，空間伝搬中の伝搬特性の変動，雑音，他のユーザ端末との干渉の影響を受ける．

以上のことから，マルチパス伝搬に起因する固有の問題を有する無線チャネルは，高いビット誤りによるスループットの低下，大きなパケット遅延によるデータ伝送遅延を考慮する必要がある．また，無線ネットワークシステムを用いて多彩なマルチメディアサービスを提供するためには，無線チャネルの特性を加味して，アプリケーションサービスごとに，ユーザの所望 QoS に応じた適応的な無線伝送制御を行う必要がある．

1.1.4　QoS 技術

多彩なモバイルサービスの中心には，インターネットの存在がある．インターネットは平等主義の設計思想であるため，たとえネットワークトラヒックが増大してネットワークが輻輳した場合でもすべてのユーザを平等に扱う．すなわち，接続性を保証する反面，ユーザの QoS に対してはベストエフォートの姿勢を崩さない．一方，インターネットの量的拡大とともに，インターネットの利用範囲の拡大，および利用形態は高度になっている．従って，本来のベストエフォートに基づき単にデータ伝送が行われるだけではなく，一定の品質を保証する必要がある．さらに，今後，様々な社会・経済活動がインターネット上で展開されることを加味すれば，インターネット接続を前提としたモバイルサービスであっても，QoS を付与することは不可避である [15].

既存の通信ネットワークシステムでは，アプリケーションサービスに対する QoS を実現するために，過剰設計手法，選択的転送手法，フロー制御手法が用いられている．過剰設計手法は，ネットワークトラヒック量と比べて十分に大きな設備容量をもつ通信ネットワークシステムを用いることにより，余力を持ってネットワークトラヒックを処理する．過剰設計手法は実装が簡単である反面，設備投資が過剰になることから経済性に難点がある．そこで，選択的転送手法は，物理的に別の回線を用意して，QoS 要求が厳しいネットワークトラヒックを区別させる．それにより，過剰設計手法の経済性に関する問題点を解決することは可能である．しかし，通信ネットワークシステムの構成が複雑になった場合には対処することができない．そこで，フロー制御手法では，単一の通信ネットワークを共用しながらも，アプリケーションサービスご

とにフローを分離し,かつ各々のフローに対してQoSを付与する．フロー制御手法は，高度な技術的検討が必要とされているが，有線ネットワークシステムにおいては実用化に至っている [16].

無線ネットワークシステムにおいて，マルチメディアサービスの提供は，上述したフロー制御手法に基づき厳密で異なるQoS要求を考慮し，かつ絶えず変化する無線チャネルの状況を鑑みた制御が必要であり，困難な技術課題を伴っている．とくに，通信ネットワークシステムが一般に採用している階層化された通信プロトコル体系は，プロトコルレイヤごとに独立した機能設計を行うことにより開発・管理・保守に関する実装容易性を高めている反面，従来のプロトコル構造は柔軟性に欠ける．すなわち，各プロトコルレイヤは動的に変化する状況に適応させるのではなく，想定されうる最悪のケースに対応できるようにオーバースペックで設計されている．また，プロトコルの最適化に関しては，主として単一のプロトコルレイヤに焦点をあてているが,局所的な最適化は必ずしも全体的なシステム性能の最適化につながるとは限らず，限られた無線資源の非効率的な利用につながる可能性がある.

以上の状況を鑑みて，無線ネットワークシステムにおいてQoS技術を設計する場合，レイヤ独立の通信プロトコル体系にとらわれない，柔軟性に富んだ新たなネットワークアーキテクチャの導入が必要不可欠である.

1.2　クロスレイヤ設計

レイヤ独立設計に基づく通信ネットワークシステムにおけるプロトコル設計は，先述したような構造上の問題点が存在する．これらの問題点を解決するために，プロトコルレイヤ同士を柔軟に結びつけるクロスレイヤ設計が注目されている [17][18]．クロスレイヤ設計では，プロトコルのモジュール化を実現するために禁止されていた複数のプロトコルレイヤ間通信を許容することにより，プロトコルレイヤの壁を越えた情報共有を可能にする．すなわち，クロスレイヤ設計は複数のプロトコルレイヤを結合的に最適化させることにより，プロトコルレイヤにまたがった垂直統合型の最適化を可能にする技術である．従って，クロスレイヤ設計を通信ネットワークシステムに導入することにより，アプリケーションサービスに応じた適応的な無線通信システムの制御が実現できる.

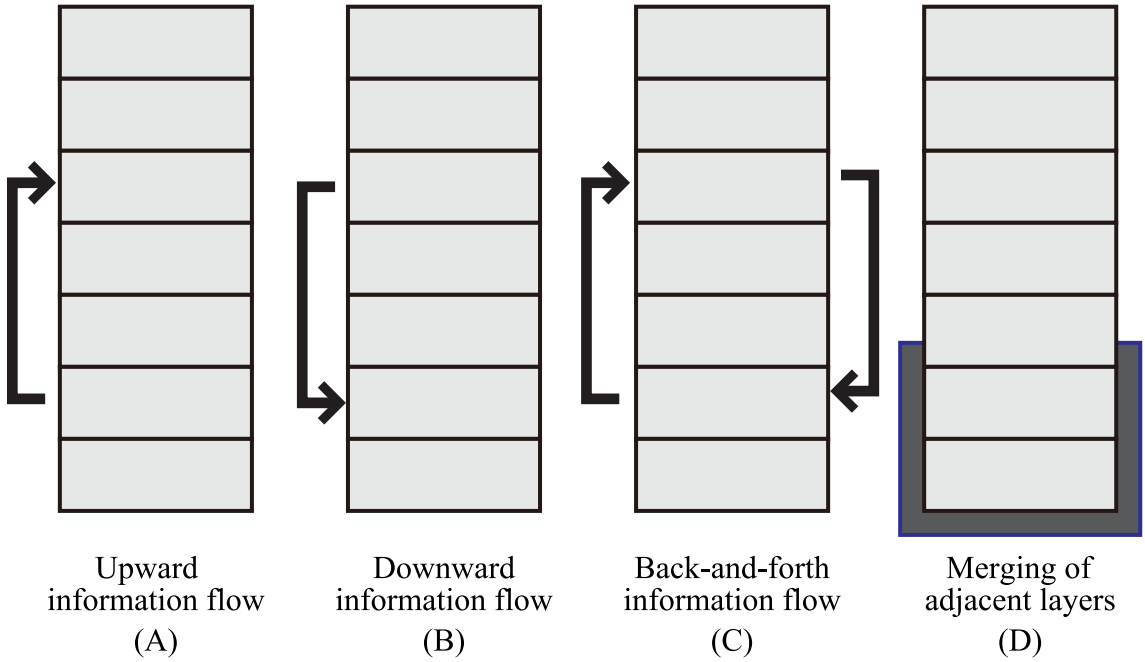


図 1.2　クロスレイヤ設計の主な設計手法

1.2.1　クロスレイヤ設計法

図 1.2 に示すように，プロトコルレイヤ間で情報共有を行う場合，その情報伝達の方向に着目するとき，クロスレイヤ設計システムの典型的な設計手法として，(A) Upward information flow，(B) Downward information flow，(C) Back-and-forth information flow，(D) Merging of adjacent layers に分類することができる．本節の残りの部分において，これら 4 手法の詳細を述べる．

Upward information flow　ネットワークの変化や無線チャネルの変動に適応してアプリケーションサービスの動作を調節するために，上位レイヤが下位レイヤの情報に基づき適応制御を実現する設計手法である．例えば，無線チャネルに応じた輻輳制御を行うことによる TCP 特性の改善がある．‘Upward information flow’ は，システムに対して独立，かつアプリケーションサービスに対して特有の方法で行われる．従って，上位レイヤにおいて，異なる QoS を要求する不均質アプリケーションサービスを支援するための高いスケーラビリティを提供することができる．

上位レイヤにおけるアプリケーションサービスは，下位レイヤからの情報を受け取った場合にのみ, 自身のパラメータを正確に調整することが可能である．そのため, 下位レイヤの情報は個々の下位レイヤのプロトコルの機構を実現しながら上位レイヤに伝達されるため，時々刻々と変化する下位レイヤの情報に対して迅速な適応制御を実現することは困難である. しかし, 上位レイヤのデータ粒度(マルチメディアフロー, データパケット）は下位レイヤのデータ粒度（ビット，変調シンボル）と比較して粗く動作するため，最適なシステム性能を実現するために瞬時的に適応させることは可能である.

Downward information flow　無線資源の利用効率を改善するために，アプリケーションサービスに応じてデータリンク層および物理層のプロトコルの調整を実現する設計手法である．例えば，アプリケーションサービスの QoS に応じた周波数の割り当て制御がある．‘Downward information flow’ は，動的な無線チャネルの変動に対して迅速に応答しながら，アプリケーションサービスの情報を加味した効率的な適応制御を実現できるため, 優れたシステム性能を生み出すことが可能である. ‘Downward information flow’ を実現するためには，下位レイヤがアプリケーションサービスにおける種々の QoS 要求が既知である前提が必要である．そのため，上位レイヤの情報を下位レイヤに伝達するための機構を導入することにより，システムの複雑化をもたらし，結果的には様々なアプリケーションサービスを 1 つのネットワークプロコトルに統合することを困難にする.

Back-and-forth information flow　‘Upward information flow’，および ‘Downward information flow’ は，高いプロトコルスタックの透明性を維持しつつ，多様なアプリケーションサービスに対して適応制御を実現することが可能である．しかし，プロトコルレイヤ間の情報伝達の遅延に伴い，ネットワーク変動に対するプロトコル調整の動作が追従しない問題点がある．そこで，‘Upward information flow’ と ‘Downward information flow’ を協調的に連動させ，その相互作用により最適化された制御の実現を図る手法が ‘Back-and-forth information flow’ である．‘Back-and-forth information flow’ は，‘Upward information flow’ または ‘Downward information flow’ を，それぞれ適切なタイミングで動作させ，動作遅延および冗長性の低減を実現する.

Merging of adjacent layers　‘Back-and-forth information flow’ では，プロトコルレイヤ間の共有情報を迅速に伝達することにより，最適化に向けた適応制御を遂行できるようにしている．とくに，無線デバイスに依存して特定のプロトコルが必然的に決定する場合，例えば，物理層とデータリンク層を対象としたクロスレイヤ設計は，隣接レイヤを融合した結合レイヤとして取り扱われる．‘Merging of adjacent layers’ は，‘Back-and-forth information flow’ におけるプロトコルレイヤ間の結びつきを究極的に強めた設計手法と考えられる．一方，プロトコル設計の見地から述べると，‘Merging of adjacent layers’ に基づく結合レイヤは，元のプロトコルレイヤと置き換えることが可能である．そのため，新たなプロトコル設計を行う手法として捉える場合，クロスレイヤ設計の研究領域ではないと判断されることがある．

1.2.2　クロスレイヤ設計の研究動向

本節において，クロスレイヤ設計に基づく無線ネットワークシステムの高度化に関する研究を概観する．クロスレイヤ設計手法は，無線ネットワークシステムの最適化に限定された考え方ではなく，システム全体の大域的な最適化を通じて最大のシステム性能の改善を得るための手法である．従って，無線ネットワークシステムだけではなく，有線ネットワークシステムを含めて，多彩な通信ネットワークシステムを対象として種々の方式が提案されている．

既存研究に関しては，適応的な無線伝送制御を実現するために，アクセス制御方式，ネットワーク・トランスポートプロトコルの改良が議論の中心である．具体的には，文献 [19][20][21][22][23] はセルラシステム，文献 [24][25][26] は IEEE 802.11 無線 LAN システム，文献 [27][28] は IEEE 802.16 WiMAX システム，文献 [29][30] は衛星通信システム，文献 [31] は衛星通信システムと WiMAX システムを融合した新たな通信ネットワークシステムに対してクロスレイヤ適応制御を実現している．また，無線ネットワークシステムの要素技術に対して，文献 [32][33] はコグニティブネットワーク技術，文献 [34] はアドホックネットワーク技術，文献 [35][36] はメッシュネットワーク技術，文献 [37] は Ultra Wide Band（UWB）ネットワーク技術を対象とした適応制御手法が提案されている．

他方，自律分散制御型無線ネットワークシステムを対象として，文献 [38][39][40] はセンサネットワークシステム，文献 [41][42] は車車間通信ネットワークシステムに対する適応制御手法が検討されている．また，近年，新たに研究されはじめた無線ネットワークシステムとして，文献 [43] は無線分散ネットワークシステム，文献 [44] は水中無線通信システム，文献 [45] はミリ波通信システムへの適用手法がある．さらに，有線ネットワークシステムを対象とした手法には，文献 [46][47] は光ファイバー通信，文献 [48] は無線ネットワークシステムの有線コアネットワークを対象とした Contents Delivery Network（CDN）に対する適応制御手法が提案されている．

先述の研究においては，各々の通信ネットワークシステムに対するクロスレイヤ適応制御として，リソース割り当て制御またはスケジューリングなどの通信ネットワーク資源の制御が議論の中心である．一方，その応用研究として，クロスレイヤ適応制御を導入することにより，QoS 差別化を図る手法も提案されている．具体的には，文献 [49][50][51][52] は ‘Upward information flow’ 設計手法を用いて，下位レイヤから得られる共有情報に基づき，ネットワーク層・トランスポート層における適応的なスケジューリング，文献 [53][54][55] は ‘Downward information flow’ 設計手法を用いて，上位レイヤから得られる共有情報に基づき，物理層・データリンク層における適応的なリソース割り当て制御を実現している．また，QoS 差別化手法を進展させ，通信ネットワーク内のアプリケーションサービスに対して QoS を保証することを目的として，文献 [56] は IEEE 802.11e を対象としたクロスレイヤ設計に基づく QoS 保証法が提案されている．

1.2.3　クロスレイヤ設計の研究課題

既存のクロスレイヤ設計に関する研究では，個々の実用技術の中でクロスレイヤ設計の考え方自体は取り入れられているが，その対象は特定の無線ネットワークシステムにおける一部のレイヤに限られている．例えば，物理層とデータリンク層に着目した適応的なリソース割り当て制御を行うことによる MAC 特性の改善，ネットワーク層とトランスポート層に着目した適応的なスケジューリングを行うことによる TCP 特性の改善が挙げられる．また，アプリケーション層に着目して，情報源符号化（ビデオコーデック）の内部信号処理を適応制御することにより，アプリケーションサービスの品質改善も検討されている．

さらに，クロスレイヤ設計システムを解析できる計算機シミュレーション環境は限られている．すなわち，既存の計算機シミュレータを利用することにより，プロトコルレイヤ独立設計に基づく無線ネットワークシステムは解析することが可能である．しかし，複数のプロトコルレイヤにまたがるような信号処理を必要とするクロスレイヤ設計システムに関しては解析できない．そのため，既存のクロスレイヤ設計に関する研究では，確率統計モデルを用いた理論的解析，または，ハードウェアを用いた実機によるシミュレーション解析を中心にシステム評価がなされている．

1.3　目　的

先述した状況を鑑みて，クロスレイヤ設計に対して求められている研究課題を再考すると，すべてのプロトコルレイヤを対象とした QoS フレームワークの提案，および新たな QoS フレームワークを評価するための計算機シミュレーション環境の構築が必要不可欠である．そこで，本研究では，アプリケーション層から物理層までを対象とした，新しいクロスレイヤ設計に基づく QoS フレームワークを提案する．また，提案フレームワークを解析するために必要な計算機シミュレータを実装する．ただし，計算機シミュレータは，特定のプロトコルに対する解析を与えるのではなく，将来の無線通信システムに幅広く導入できることを目指している．

以上をまとめると，本研究において，具体的には次の 4 点を検討する．

- 複数のプロトコルレイヤ間で QoS 情報を共有するための QoS フレームワークの提案
- クロスレイヤ設計に基づく適応的な無線伝送制御手法の提案
- ヘテロジニアスなネットワーク環境に対して提案手法を導入する際のプロトコル設計の提案
- 提案手法を評価するために必要な計算機シミュレータの実装，および提案手法の評価

本節の残りの部分において，4 点の検討課題について簡単に述べる．

クロスレイヤ設計に基づく QoS フレームワークの提案に関して，従来手法と提案手法の大きく異なる点は，QoS 情報を共有するために QoS converter を導入している

点である．すなわち，各々のプロトコルレイヤによって定義・分類が異なる QoS 情報の関係を明確にして，QoS 情報をプロトコルレイヤ間で共有するために，提案手法では QoS converter を用いる．そして，QoS converter を経由して受け取った QoS 情報に基づき，Cross-layer optimizer が適応的な無線伝送制御を実現する．クロスレイヤ設計に基づく適応的な無線伝送制御法の検討に関して，Cross-layer optimizer にて実現される適応パケット長制御法および適応レート制御法を提案する．とくに，無線 LAN システムに導入した場合を想定して，MAC プロトコルの設計，制御パラメータの決定手法を提案する．

また，将来の無線ネットワークシステムにおける重要検討課題であるヘテロジニアスネットワークへの対応に関連して，提案手法をヘテロジニアスネットワーク環境に導入する場合に必要なプロトコル設計を検討する．すなわち，複数の通信ネットワークシステム間で QoS 情報を共有するために，提案手法では SIP を用いた QoS 情報の管理法を提案する．さらに，既存のクロスレイヤ設計の研究課題である，提案クロスレイヤ設計システムを解析するための新たな計算機シミュレーション環境を構築する．実装した計算機シミュレータでは，物理層・データリンク層のプロトコルは C++ 言語 [57], ネットワーク層・トランスポート層のプロトコルは広域ネットワークシミュレータ ns2[58] を用いる．また，提案手法の有効性を示すために，この計算機シミュレータを用いて解析を行う．

1.4 論文構成

本論文の論文構成は次の通りである．第 2 章において提案する QoS フレームワークについて述べた後，第 3 章ではクロスレイヤ設計に基づく適応無線伝送制御法について述べる．また，第 4 章では提案手法をヘテロジニアスなネットワーク環境に導入する場合において，実装に必要なプロトコル設計を述べる．第 5 章では提案手法を評価するための計算機シミュレータを実装し，提案手法を評価する．最後に第 6 章でまとめを行う．

第 2 章
QoS フレームワーク

本章では，本論文で取り扱う QoS の定義を与えた後，従来の QoS フレームワークについて概観する．それから，提案するクロスレイヤ設計に基づく QoS フレームワークについて述べる．

2.1　はじめに

2.1.1　QoS の定義

一般的に，通信ネットワークシステムが提供するアプリケーションサービスを利用するのはユーザであり，通信に関わる品質はユーザの立場での善し悪しを図る尺度で定義されるべきである．すなわち，通信ネットワークシステムにおける QoS とは，対象となる通信ネットワークシステムが提供するアプリケーションサービスの使いやすさの程度，または均一性である．このような QoS の考え方を発展させ，ITU[3] では，ユーザの立場における QoS を Quality of Experience（QoE）と定義している [59]．具体的には，QoE をユーザによって主観的に知覚される総合的なアプリケーションサービスの受容性と定義する．また，QoE には，ユーザ，ユーザ端末，通信ネットワーク，アプリケーションサービスなどが与える影響が含まれる．

以上の状況を鑑み，本論文では，広義に捉えた QoS，または QoE を対象とするのではなく，QoE の構成要素のひとつである通信ネットワークのメディア品質に着目して QoS を考える．すなわち，典型的な通信の 3 品質として議論される，接続品質，伝送品質，安定品質に焦点をあてる．例えば，インターネットの場合，伝送レート，パケット誤り率，パケット遅延時間が挙げられる．

2.1.2　QoS の実現法

通信ネットワークシステムに対して QoS を実現するために，基本的な設計手法として，プライオリティ制御を加味したスケジューリング法，通信ネットワーク資源の予約法，ネットワーク輻輳の抑制法がある．本節の残りの部分において，これら 3 手法の詳細を述べる．

プライオリティ制御を加味したスケジューリング法　優先度が異なる複数のネットワークトラヒックを単一の通信ネットワークシステムで取り扱う場合，QoS を実現するために，その優先度に応じてデータ伝送処理を行う．例えば，高優先サービスと低優先サービスのフローを同時に取り扱う場合，高優先フローのパケットを優先的に取り扱う．このようなプライオリティ制御に基づくスケジューリングを導入することにより，高優先フローに対する QoS を実現することができる．ただし，低優先フローの QoS は過度に劣化する可能性があり公平性の面で課題は残る．さらに，高優先フローのパケットの処理を希望するときに，高優先（または低優先）のパケットを処理している場合, そのパケットの処理が終了するまで待機しなければならない．従って，プライオリティ制御であっても，多少の待ち合わせに伴う遅延は生じ，とくにネットワーク負荷が大きい場合には QoS 低下は避けられない．

通信ネットワーク資源の予約法　通信ネットワークシステムの資源は有限であるため，ネットワークがどのような状況においても，アプリケーションサービスが所望する QoS を必ず提供できるとは限らない．例えば，プライオリティ制御を加味したスケジューリング法を通信ネットワークシステムに導入した場合であっても，ネットワーク負荷が高い場合には，すべてのトラヒックに対し同等の QoS を結果的に提供せざるを得ない状況が想定される．そこで, 確実な QoS を求める場合, 通信ネットワークシステムに対して所望 QoS を提供することができるかという点を問い合わせ，システム側はネットワーク輻輳の状況を勘案して受付の可否を決める．その結果，通信ネットワーク資源の予約に係るオーバヘッドは増大するが，QoS を確実に提供することができるようになる．

ネットワーク輻輳の抑制法　通信ネットワークシステムにおいて QoS を導入する場合，ネットワーク輻輳の考慮は必要不可欠である．一般的にネットワーク輻輳とは，通信ネットワークの負荷が増大することに伴い，多数のアプリケーションサービスにおける QoS が低下する状態である．技術的な観点から述べると，通信ネットワークの特定エリアに流入するネットワークトラヒックが，その許容する通信システムの容量を超えることである．とくに，有線ネットワークシステムにおいては，ネットワーク輻輳は QoS に直接的に影響を与えることから，QoS の差別化，または QoS を保証するためにネットワーク輻輳を抑制することは重要である．

2.2　従来手法

本節では，有線ネットワークシステム，および無線ネットワークシステムにおける QoS の実現手法を概観した後, クロスレイヤ設計を導入した既存手法について述べる．

2.2.1　有線ネットワークシステムにおける QoS

有線ネットワークシステムにおいて，通信事業者内のバックボーンを支えるコアネットワークでは，高度な QoS を保証するネットワークが構築されている．例えば，Asynchronous Transfer Mode（ATM）技術を導入したネットワークでは，すべての情報をセルと呼ばれる固定長ブロックに収容してデータ伝送を実現している．ATM ネットワークでは, 情報が同一形状のセルに収容されている点を利用して, 多重･分離･交換の処理を一元的に行うことができる．そのため，通信制御が容易になる点，ネットワーク内の伝送路の違いを意識することなく処理できる点が期待できるので，任意の帯域を保証した QoS を与えることが可能である．

また，パケット交換方式によるデジタルデータ回線と回線交換方式によるアナログ音声回線を統合的に取り扱うことを目的とした，Next Generation Network（NGN）が 2008 年に実用化されている．NGN では ATM ネットワークのような特殊なセルを用いるのではなく，広く一般的に用いられている IP を用いて通信ネットワークシステムを構築している．NGN における QoS 保証は，Resource and Admission Control Functions（RACF）を用いて実現されている．ATM ネットワークおよび NGN においては，共に ‘ 通信ネットワーク資源の予約法 ’ に基づき QoS を提供している．

他方，インターネットにおける QoS を実現する代表的な手法として，DiffServ および IntServ がある．DiffServ は‘プライオリティ制御を加味したスケジューリング法’に基づき，ネットワークトラヒックを QoS が異なる QoS クラスに分類して，クラス間の相対的な QoS を実現する手法である．従って，ユーザ端末におけるエンドツーエンドに対する QoS を厳密に保証することはできないが，新たな制御用プロトコルを導入する必要がないため，通信ネットワークシステムに導入する際に高いスケーラビリティを持つ．DiffServ は IP パケットの Type of Service（TOS）フィールドの情報を用いて，アプリケーションサービスに応じたプライオリティ制御に基づくスケジューリングを実現する [60].

また，IntServ は‘通信ネットワーク資源の予約法’に基づき，Resource Reservation Protocol（RSVP）を用いてアプリケーションサービスが通信ネットワーク資源の利用を予約した後，実際のデータ伝送を行う．具体的には，RSVP に基づくセットアッププロトコルでは，トラヒック制御を行うために，受付制御，流量制御，スケジューリングを行う．すなわち，受付制御によって通信ネットワーク資源が適切なサービスを提供できるかどうかを判断する．それから，流量制御によってアプリケーションサービスのフローに対する QoS クラスを分別し，その QoS クラスに応じたスケジューリングを行う．IntServ は，通信ネットワークの資源を予約するため，エンドツーエンドに対する絶対的な QoS を保証することが可能である．ただし，現実的な通信ネットワークシステムに導入する場合，ネットワークトラヒックに対する状態を管理する必要があるため，スケーラビリティに欠ける問題点がある．

2.2.2　無線ネットワークシステムにおける QoS

無線ネットワークシステムとして，セルラシステム，および無線 LAN システムにおける QoS の実現手法について概観する．セルラシステムにおいて，基地局とユーザ端末間の無線伝送区間に関しては，QoS Class Indicator（QCI）と呼ばれるアプリケーションサービスごとの QoS クラスを決定し，基地局とユーザ端末の双方にて共有している．そして，‘プライオリティ制御を加味したスケジューリング法’を用いて QCI に基づく無線チャネルに対するリソース割り当て制御を実現している．一方，基地局間の有線伝送区間に関しては，第 3 世代セルラシステムまでは ATM ネットワークが用いられている．また，第 3 世代セルラシステム以降は IP ネットワークへの完全移

行が図られ，DiffServ を用いた QoS の提供に併せて Multi-Protocol Label Switching（MPLS）も用いられている．MPLS は特定の通信ネットワークに負荷がかからないようにするために，ネットワークトラヒックを束ね，Label Switched Path（LSP）と呼ばれる集約ルートに自動的に割り当てる制御を駆使して，‘ネットワーク輻輳の抑制法’に基づく QoS を実現する．

また，無線 LAN システムにおいては，IEEE 802.11e が仕様化されている．IEEE 802.11e は，‘プライオリティ制御を加味したスケジューリング法’を用いて，アプリケーションサービスの QoS に応じて，優先度の高いトラヒックに対して送信機会を多く与える．具体的には，MAC 層に QoS の機能を付加することを目的として，Enhanced Distributed Channel Access（EDCA），および HCF Controlled Channel Access（HCCA）が具備されている．EDCA および HCCA では，基地局とユーザ端末は，優先度に対応したパケットを一時的に蓄積するメモリであるキューをもち，優先度が高いトラヒックを優先的に選別して送信することにより QoS を実現する．

2.2.3　既存のクロスレイヤ設計における QoS

既存のクロスレイヤ設計を用いた QoS を実現する研究では，‘プライオリティ制御を加味したスケジューリング法’に基づき，個々の実用技術の中でクロスレイヤ設計の考え方が取り入れられている．また，そのクロスレイヤ設計に関しては，‘Upward information flow’設計手法，または ‘Downward information flow’設計手法のいずれかに分類することができる．‘Upward information flow’設計手法に関しては，マルチメディアコンテンツの伝送品質を向上させるために，ある程度限定されたシナリオの下で検討されている [49][50][51][52]．一方，‘Downward information flow’設計手法に関しては，1 ホップの無線ネットワークシステムを対象とした，データリンク層または物理層のいずれかのプロトコルレイヤについてのみ焦点があてられている [53][54][55][56]．このとき，ネットワーク層またはトランスポート層は重要な役割を担わず，データリンク層または物理層と併せて関連づけた研究はなされていない．

2.3 提案手法

2.3.1 問題提起および提案手法の概要

有線ネットワークシステムにおいては，QoS を実現する手法は確立されている．一方，無線ネットワークシステムにおける従来手法は，完成された有線ネットワークシステムの QoS 技術を単に導入している状況である．とくに，無線ネットワークシステムでは，将来の多様な QoS を収容するだけではなく，無線チャネルの状況が動的に変化する点も加味しなければならない．従って，有線ネットワークシステムと同様の手法（すなわち，プライオリティ制御を加味したスケジューリング法，通信ネットワーク資源の予約法，ネットワーク輻輳の抑制法）を用いて QoS を導入するだけでは不十分である．さらに，有線ネットワークシステムでは，過剰設計手法の考え方に基づき，インフラの敷設コストを無視すれば必要十分なだけの通信チャネルを確保することが可能である．しかし，無線ネットワークシステムでは，電波伝搬に必要な空間は唯一の共有資産であり，かつ無線チャネルの周波数資源は限られている．従って，無線ネットワークシステムに最適化された，クロスレイヤ設計に基づく統合的な QoS フレームワークの構築は必要不可欠である．

以上の状況を鑑みて，すべてのプロトコルレイヤを対象としたクロスレイヤ設計に基づき，現実的に実現可能な新たな無線ネットワークシステムにおける QoS フレームワークを提案する．図 2.1 に提案する QoS フレームワークを示す．提案手法では，Cross-layer converter がクロスレイヤ制御情報に基づき，アプリケーションサービスの種類から適切な無線伝送制御パラメータを選択して，それに基づき Cross-layer optimizer が適応的な無線伝送制御を行う．本章の残りの部分において，提案手法の詳細を述べる．

2.3.2 Cross-layer converter

アプリケーションサービスに応じて，クロスレイヤ設計に基づく適応的な無線伝送制御を実現するためには，プロトコルレイヤの壁をこえた情報共有が必要不可欠である．一般的に階層化された情報ネットワークシステムでは，プロトコルスタックと同様に，無線伝送制御パラメータも階層的に定義される．単純にプロトコルレイヤ間の情報共有を実現するだけであれば，パケットのヘッダ領域に QoS 情報を追加する手

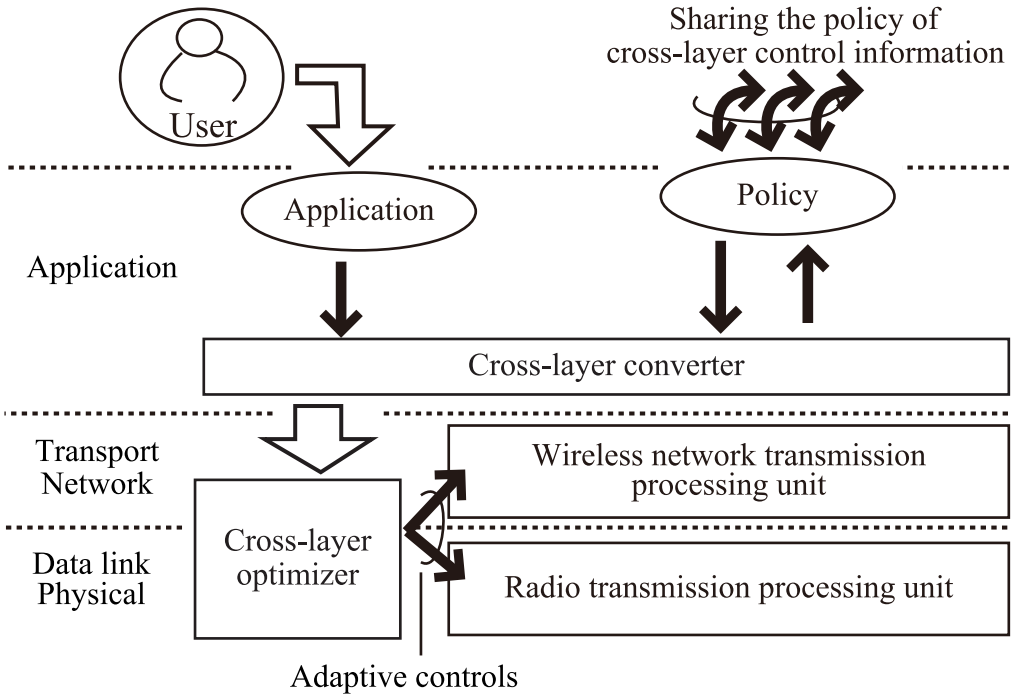


図 2.1　提案 QoS フレームワーク

法，または狭帯域な専用通知チャネルを用いて独自仕様の通知プロトコルを設計する手法を導入することにより実現可能である [61][62]. これらの手法は, 特定の無線ネットワークシステムに対しては有効であるが，汎用性の点で課題がある．この汎用性の問題点を克服するために，任意のプロトコルレイヤ間で情報共有を実現する CEAL が提案されている [63]．CEAL と提案手法は，共有情報を汎用化している点において類似している．一方，CEAL ではユーザが共有情報の定義をしなければならないが，提案手法では Cross-layer converter が QoS の対応付けを行うことによる抽象化を実現している点，およびクロスレイヤ制御を実現している点が異なっている．

従来手法における汎用化および抽象化に関する問題点を改善するために，提案手法では Cross-layer converter を用いる．すなわち, 提案手法では, アプリケーションサービスの情報と無線伝送制御パラメータ間の対応付けは Cross-layer converter を介して行う．Cross-layer converter では，定義が異なるパラメータ情報の対応付け規則をクロスレイヤ制御情報として規格化する．従って，提案手法を導入することにより，アプリケーション設計者（ユーザ），ネットワーク設計者，無線通信システム設計者のそれぞれの立場において，次に挙げるような抽象化に伴う利点が生じる．

- **アプリケーション設計者（ユーザ）**：無線ネットワークに関する詳細な知識がなくても，利用するアプリケーションサービスに適した無線ネットワーク環境を利用できる．
- **ネットワーク設計者**：クロスレイヤ制御情報を変更するだけで，任意のプロトコルスタックを柔軟に導入できる．
- **無線通信システム設計者**：アプリケーションサービスの詳細な知識がなくても，クロスレイヤ制御情報に基づき，無線通信システムの調整ができる．

Cross-layer converter は，プロトコルに依存するアプリケーションサービスに応じた制御情報をプロトコルに依存しない形式に変換する役割を担う．本論文では，そのプロトコルに依存しない共通の QoS 情報をクロスレイヤ制御情報と定義する．すなわち，上位レイヤにおけるアプリケーションサービスに応じた QoS 情報は，下位レイヤに直接伝達されるのではなく，Cross-layer converter を用いてクロスレイヤ制御情報に変換して伝達する．クロスレイヤ制御情報に関しては, 次節にて詳細を述べる．

2.3.3　クロスレイヤ制御情報

QoS converter において，ネットワーク設計者はクロスレイヤ制御情報として，プロトコルレイヤ間の対応付け規則を定義する必要がある．プロトコルレイヤ間の対応付け規則を決定する場合，ネットワーク設計者は，そのプロトコルレイヤの機能・動作を十分に理解した上で設定しなければならない．表 2.1 に，アプリケーションサービスに応じた無線伝送制御パラメータの対応規則の一例を示す．この対応規則は，提案手法を導入するネットワーク環境ごとにネットワーク設計者が決定する．例えば，Voice over IP（VoIP）通信, または Video on Demand（VoD）配信のようなアプリケーションサービスは，パケット遅延が発生した場合，音声や映像が不自然になるため，安定したスループットと低遅延が求められる．一方，ファイル転送や蓄積型メディア配信サービスは，あらかじめコンテンツデータをバッファなどにダウンロードし，ダウンロード済みのデータを実際に利用する．そのため，パケット遅延に対する要求は高くないので，パケット遅延にとらわれない効率の良い伝送方式が利用できる．

表 2.1　アプリケーションサービスに対する無線伝送制御パラメータの対応規則

Application	Transport	Priority	Packet length	Rate	Resource
Real-time	TCP	Higher priority	QoS aware	QoS aware	Wideband
Transaction	TCP	Higher priority	QoS aware	QoS aware	Narrowband
File transfer	TCP	Lower priority	Best effort	Best effort	Wideband
Streaming	UDP	Higher priority	QoS aware	QoS aware	Wideband
VoD	UDP	Higher priority	QoS aware	QoS aware	Wideband
VoIP	UDP	Higher priority	QoS aware	QoS aware	Narrowband
DTN	UDP	Lower priority	Best effort	Best effort	Narrowband
Stored media	UDP	Lower priority	Best effort	Best effort	Wideband

提案手法では，クロスレイヤ制御情報は Extensible Markup Language（XML）に基づく表記法に従い記述する．例えば，表 2.1 に対するクロスレイヤ制御情報の記述例を図 2.2 に示す．当然，表 2.1 に掲載していないアプリケーションサービス，または設定項目に関しては，図 2.2 と同様の表記法に従い定義することにより拡張することが可能である．

```
01:  <?xml version="1.0" encoding="utf-8" ?>
02:  <CrossLayerControlInformation>
03:    <RuleDefinition num="1">
04:      <application type="RealTime" />
05:      <Transport type="TCP" />
06:      <Priority type="Higher" />
07:      <PacketLength type="QoSAware" />
08:      <Rate type="QoSAware" />
09:      <ResourceAllocation type="Wideband" />
10:    </RuleDefinition >
11:    <RuleDefinition num="2">
12:      <application type="OnlineTransaction" />
13:      <Transport type="TCP" />
14:      <Priority type="Higher" />
15:      <PacketLength type="QoSAware" />
16:      <Rate type="QoSAware" />
17:      <ResourceAllocation type="Narrowband" />
18:    </RuleDefinition >
19:    </RuleDefinition >
20:    <RuleDefinition num="3">
21:      <application type="FileTransfer" />
22:      <Transport type="TCP" />
23:      <Priority type="Lower" />
24:      <PacketLength type="BestEffort" />
25:      <Rate type="BestEffort" />
26:      <ResourceAllocation type="Wideband" />
27:    </RuleDefinition >
28:    <RuleDefinition num="4">
29:      <application type="Streaming" />
30:      <Transport type="UDP" />
31:      <Priority type="Higher" />
32:      <PacketLength type="QoSAware" />
33:      <Rate type="QoSAware" />
34:      <ResourceAllocation type="Wideband" />
35:    </RuleDefinition >
36:    <RuleDefinition num="5">
37:      <application type="VoD" />
38:      <Transport type="UDP" />
39:      <Priority type="Higher" />
40:      <PacketLength type="QoSAware" />
41:      <Rate type="QoSAware" />
42:      <ResourceAllocation type="Wideband" />
43:    </RuleDefinition >
44:    <RuleDefinition num="6">
45:      <application type="VoIP" />
46:      <Transport type="UDP" />
47:      <Priority type="Higher" />
48:      <PacketLength type="QoSAware" />
49:      <Rate type="QoSAware" />
50:      <ResourceAllocation type="Narrowband" />
51:    </RuleDefinition >
52:    <RuleDefinition num="7">
53:      <application type="DTN" />
54:      <Transport type="UDP" />
55:      <Priority type="Lower" />
56:      <PacketLength type="BestEffort" />
57:      <Rate type="BestEffort" />
58:      <ResourceAllocation type="Narrowband" />
59:    </RuleDefinition >
60:    <RuleDefinition num="8">
61:      <application type="StoredMediaTransfer" />
62:      <Transport type="UDP" />
63:      <Priority type="Lower" />
64:      <PacketLength type="BestEffort" />
65:      <Rate type="BestEffort" />
66:      <ResourceAllocation type="Wideband" />
67:    </RuleDefinition >
68:  </CrossLayerControlInformation>
```

図 2.2　XML を用いたクロスレイヤ制御情報の記述例

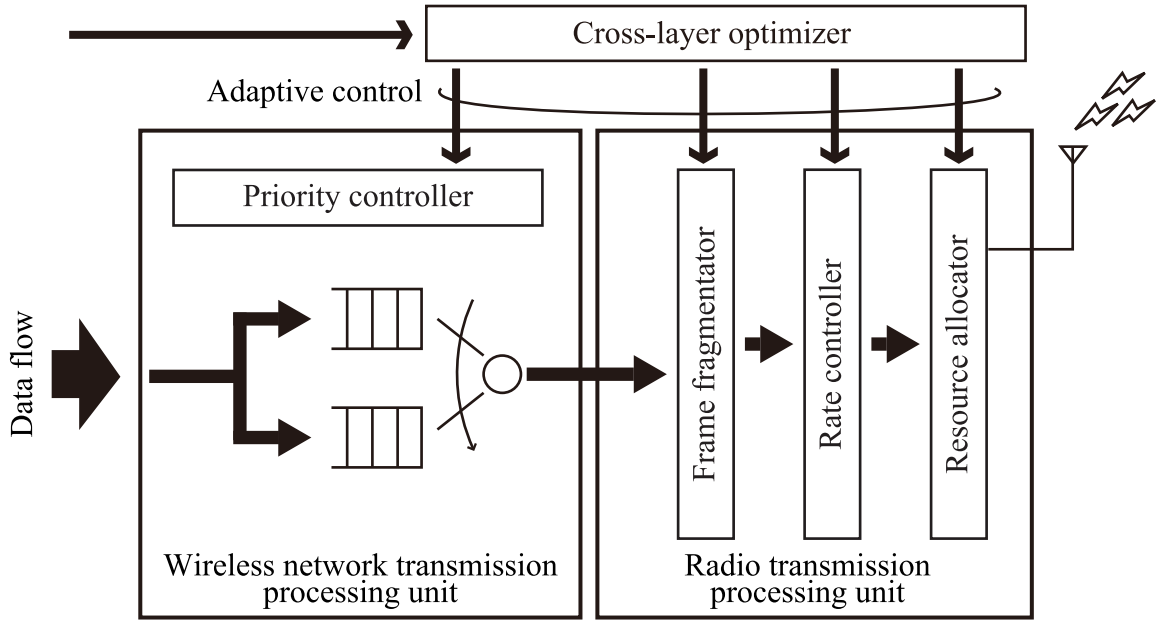


図 2.3　クロスレイヤ適応制御を実現するための無線伝送信号処理手順

2.3.4　Cross-layer optimizer

クロスレイヤ設計に基づく適応伝送制御を実現するために，提案手法では Cross-layer optimizer を導入する．図 2.3 に，Cross-layer optimizer を用いた適応制御の対象となる，図 2.1 における Wireless communications processing unit の構成，および Cross-layer optimizer との関係を示す．提案手法では，上位レイヤにおけるアプリケーションサービスの QoS 情報は，下位レイヤに直接伝達されるのではなく，クロスレイヤ制御情報に基づき，Cross-layer converter において無線伝送制御パラメータに変換される．そして，Cross-layer optimizer は無線伝送制御パラメータに応じて適応的な無線伝送信号処理を実現する．クロスレイヤ設計に基づく適応伝送制御の対象となる無線伝送信号処理には，プライオリティ制御，パケット長制御，レート制御，リソース割り当て制御がある．提案手法では，これらの適応無線伝送信号処理を実現するために，Cross-layer optimizer を用いて，各々，Priority controller，Frame fragmentator，Rate controller，Resource allocator　の伝送パラメータを適応的に制御する．具体的な制御手法に関しては後述する．

2.4　おわりに

本章では，本論文で取り扱う QoS を定義し，従来の QoS フレームワークを概観した後，提案する QoS フレームワークについて述べた．提案手法では，QoS 情報をクロスレイヤ制御情報として取り扱う．また，Cross-layer converter を用いてプロトコルレイヤ間の QoS に関する定義の差違を吸収する．さらに，クロスレイヤ制御情報に基づき，Cross-layer optimizer は適切なパラメータ設定を決定し，適応的な無線伝送制御を実現する．無線伝送制御におけるクロスレイヤ設計に基づく適応制御は提案手法の中でも重要な位置づけにあるため，第 3 章において詳細を述べる．

第３章
クロスレイヤ適応制御

本章では，提案手法の中でも重要な位置づけにあるクロスレイヤ適応制御に関して，無線伝送制御手法を述べる．とくに，パケット長制御およびレート制御に焦点をあて，無線 LAN システムに対して提案手法を導入した場合の数値例を示す．

3.1　はじめに

第２章では，提案手法における QoS フレームワークについて述べた．提案 QoS フレームワークを無線ネットワークシステムに導入する場合，クロスレイヤ設計に基づく適応無線伝送信号処理を実現する必要がある．具体的には，レート制御を加味したパケット分割に基づくパケット分割手法を提案する．

パケット長制御を実現するために，パケットをいくつかのフレームに分割することで仮想的にパケット長を変える．すなわち，無線チャネルの状況が悪い場合，フレーム全長が短いフレームを使うことにより，再送しなければならないデータ量を減らすことができる．その理由は，フレーム全長が長いフレームと比べて，短いフレームの方が受信側へのフレーム到達率が高いためである．しかし，短いフレームは長いフレームと比べて，ヘッダ情報等の付加に伴うオーバヘッドの比率が大きいため，チャネル状況が良い場合，長いフレームを用いた方が送信データの総量を小さくすることができる．従って，無線チャネルの状況に応じた最適な全長のフレームを用いることで，データ伝送効率を改善することが可能になる．

将来の無線ネットワークシステムが取り扱うことが重要視されるマルチメディアサービスを考慮して，提案クロスレイヤ適応制御手法において，最適なフレーム長および最適なレートはパケット遅延に基づき決定する．また，無線チャネルの動的な変化も加味するために，Channel State Information（CSI）も併せて考慮する．

提案クロスレイヤ適応制御手法の数値例は，無線 LAN システムに導入した場合を想定して評価する．その理由は，無線 LAN システムを用いたヘテロジニアスネット

ワークにおけるスモールセルが，その広帯域なデータ伝送帯域を持っている長所を活かして，マルチメディアサービスの伝送を担うことが期待されているからである．ただし，提案手法とヘテロジニアスネットワークとの関係の詳細は第 4 章で述べる．

3.2　プロトコル設計

3.2.1　MAC プロトコル

本節では，MAC プロトコルについて概観した後，提案手法における MAC プロトコル設計について述べる．無線 LAN のデファクトスタンダード規格として幅広く用いられている IEEE 802.11 では，同一の無線チャネルを複数のユーザ端末で共有するためのアクセス制御機能が具備されている．具体的には，無線 LAN の基本アクセス手順である Distributed Coordination Function（DCF），およびオプションとして実装されている Point Coordination Function（PCF）と呼ばれる 2 種類の機構がある．

DCF はフレームの衝突をできるだけ回避するために，無線チャネルの使用状況を見てからフレームを送信するかどうか決定する Carrier Sense Multiple Access with Collision Avoidance（CSMA/CA）方式が用いられている．CSMA/CA 方式では，フレームの送信を試みようとする各々のユーザ端末は，事前に無線チャネルが他のユーザ端末に使用されていないかを確認する．もし，他のユーザ端末が無線チャネルを使用している場合，送信を見合わせることによって衝突を回避する．一方，PCF では，無線基地局に在圏するユーザ端末をポーリングに基づく集中制御によるアクセス制御が行われている．すなわち，無線基地局が各々のユーザ端末に対し順番にポーリング信号を送信し，ポーリング信号を受け取ったユーザ端末がフレーム送信を許可されるアクセス制御方式である．PCF は，無線基地局に在圏するユーザ端末同士の衝突は発生しないが，周辺に同一無線チャネルを利用する無線基地局が存在する場合，ポーリング信号同士が衝突する可能性が残る．

図 3.1 に，提案手法において採用する DCF に基づくベーシックアクセス方式を用いた MAC プロトコルの信号処理手順を示す．提案手法は隠れ端末問題については取り扱っていないため，Request to Send/Clear to Send（RTS/CTS）は導入しない．また，解析簡単化のために，無線基地局とユーザ端末が直接的に通信を行うインフラストラクチャ型の無線 LAN モデルを用いる．IEEE 802.11 規格に基づき，無線基地局

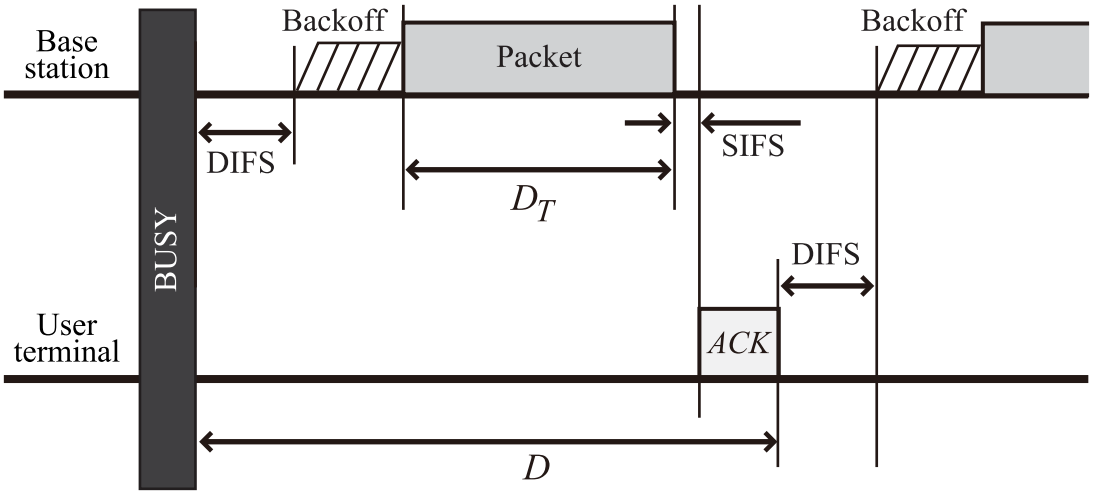


図 3.1　DCF に基づく MAC プロトコルの信号処理手順

は Distributed Coordination Function Interframe Space（DIFS），バックオフの後にパケットの初回送信を行う．もし，ユーザ端末にパケットが正しく届いた場合，ユーザ端末は *ACK* メッセージを無線基地局に Short Interframe Space（SIFS）の時間をあけて返信する．一方，パケット損失またはパケット衝突によりパケット欠損が生じた場合，*NACK* メッセージを返信するとともに，パケット再送制御を行う．

3.2.2　パケット分割モデル

現在の通信ネットワークシステムでは，情報源から出力されるメッセージは，いくつかのひとかたまりのデータとして取り扱われる．プロトコルレイヤにおける階層化モデルとして，国際標準化機構 International Organization for Standardization（ISO）が提唱する Open Systems Interconnection（OSI）参照モデルにおいては，η – レイヤにおけるデータ単位として，η –Protocol Data Unit（PDU）が定義される [63]．本論文では，PDU に関する用語の定義を明確にするために，パケットおよびフレームを用いる．図 3.2 にパケットおよびフレーム分割モデルを示す．伝達されるメッセージは，ネットワーク層においてパケットに分割され，そのパケットをデータリンク層においてフレームに分割する．また，各々のフレームは，分割したパケットの情報と制御情報を保有する．

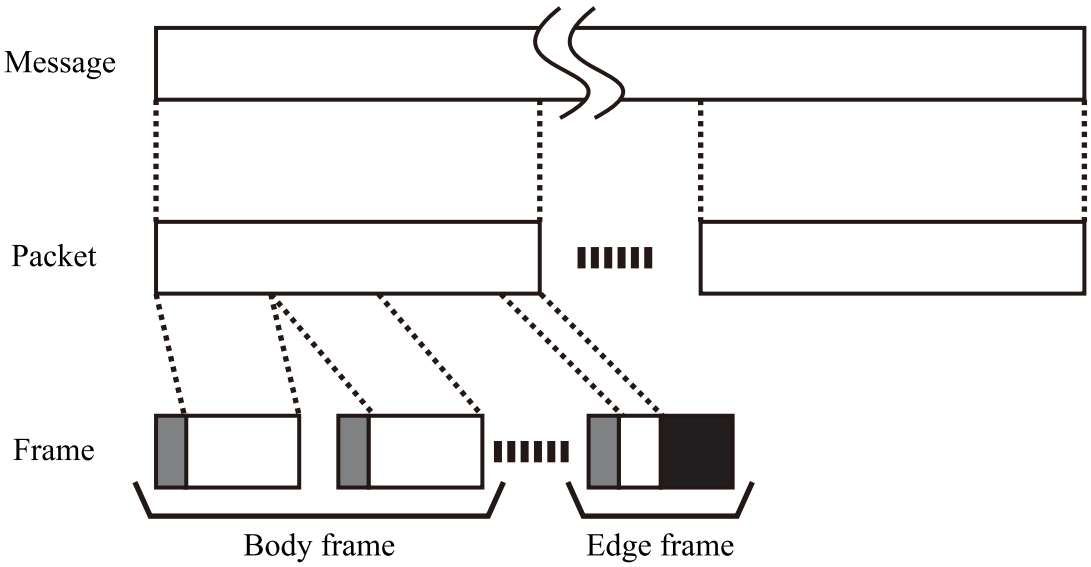


図 3.2　パケット分割モデル

パケットをフレーム分割する際のフレーム長を適応的に制御することにより，適応パケット長制御を実現する．図 3.2 に示しているように，パケットは ‘Body frame’ および ‘Edge frame’ と呼ぶ２種類のフレームに分割する. 図3.3にパケットおよびフレームの構成を示す．‘Body frame’ および ‘Edge frame’ は，各々フレームヘッダを含んでいる点は同じであるが，‘Edge frame’ はフレームの余剰スペースをゼロパディングすることによって埋めている点が異なっている．従って，各々のパケットは，いくつかの ‘Body frame’ と 1 個の ‘Edge frame’ に分割される．

パケットの全長を$x_h(x_h \in \mathcal{X})$と定義する．ただし，x_h は非負整数，$\mathcal{X}$ は x_h が取り得る集合とする．また，フレームの全長を L ，フレームヘッダ長を L_H , ‘Edge frame’ におけるゼロパディング領域の全長を L_P と定義する．このとき, 1 個のパケットを μ 個のフレームに分割するとき，分割数 μ は式（3.1）で計算できる．ただし，μ は非負整数，$\lceil a \rceil$ は，a と等しいまたは a より大きい最小の整数を与える演算子である．

$$\mu = \lceil \frac{x_h}{L - L_H} \rceil \tag{3.1}$$

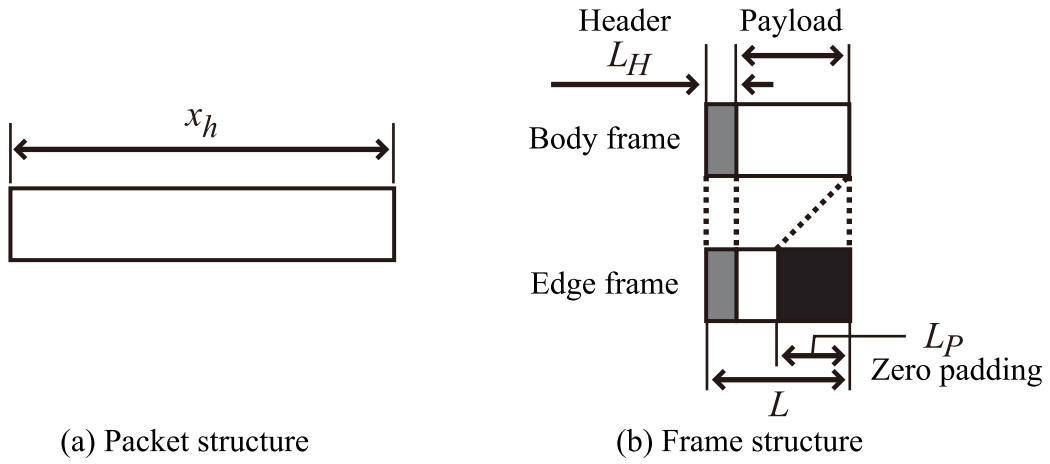


図 3.3　パケットおよびフレームの構成

3.2.3　フレーム再送制御

無線チャネルの雑音によって生じるビット誤りに対してどのように制御するかは，無線ネットワークにおいて重要な懸案事項である．現在の情報ネットワークにおいて幅広く用いられているビット誤り制御手法は，Automatic Repeat Request（ARQ）と Forward Error Correction（FEC）に大別される．ARQ は信頼性の高いデータ伝送を行うために，送受信側において，*ACK* および *NACK* を用いた送達確認を行う．そのため，受信データの品質を維持しやすいが，再送による遅延により対話性やリアルタイム性が損なわれる．とくに，無線ネットワークのような資源が限られたネットワーク環境では無視することができない．一方，FEC はデータに冗長性を与えることにより，受信側でビット誤りがあった場合に訂正を行うことができる．そのため，ARQ における再送遅延に起因する問題点を解決することが可能であるが，FEC の核となる誤り訂正符号化方式によっては，複数ビットをまとめてブロックとして符号化処理を行うため，それに関わるオーバヘッドを考慮しなければならない．

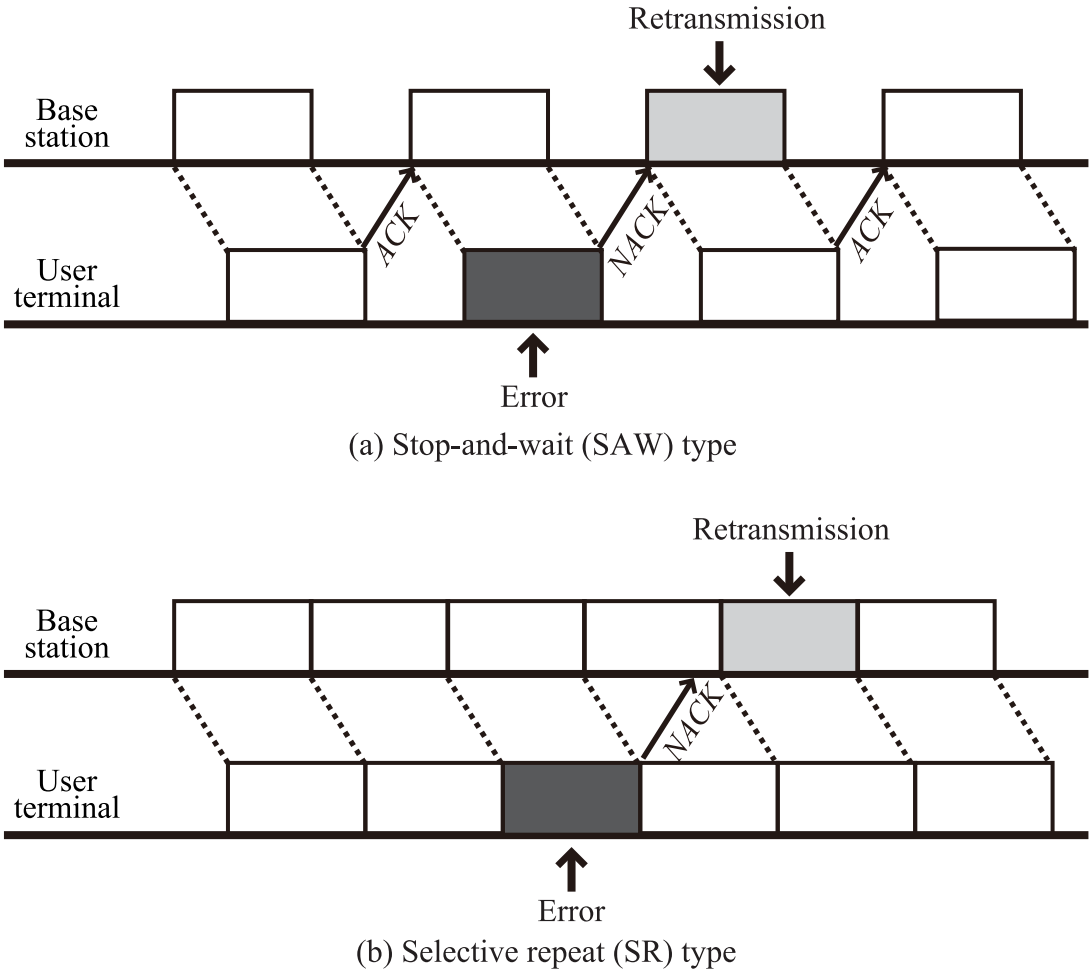


図 3.4　SAW-ARQ および SR-ARQ におけるフレーム（再）送信手順

無線ネットワークにおいては，ARQ と FEC を組み合わせた Hybrid ARQ が用いられている [64]．Hybrid ARQ 手法において，フレーム再送のタイミングによって Stop-and-wait（SAW）–ARQ および Selective Repeat（SR）–ARQ の 2 種類に分類することができる．図 3.4 に，SAW-ARQ および SR-ARQ におけるフレーム送信・再送信手順を示す．SAW-ARQ では，無線基地局はユーザ端末から *ACK* または *NACK* を受け取った後，初回フレームまたは再送フレームを直ちに送信する．一方，SR-ARQ では，無線基地局は *NACK* を受け取ったときのみ再送フレームを再送し，それ以外はフレームを送信し続ける．そのため，SAW-ARQ と比較して効率的なデータ伝送が実現できる．本論文では，IEEE 802.11 で用いられているこれら 2 種類の Hybrid ARQ 手法を取り扱う．

3.2.4　パケット遅延時間

図 3.1 に示すように，提案手法では IEEE 802.11 無線 LAN において，DCF に基づく MAC プロトコルを用いる．このとき，パケット遅延 D を式（3.2）で定義する．

$$D = T_{\mathrm{DIFS}} + T_{\mathrm{Backoff}} + D_T + T_{\mathrm{SIFS}} + T_{\mathrm{(N)ACK}} \tag{3.2}$$

ただし，DIFS 時間を T_{DIFS}，バックオフ時間を T_{Backoff}，SIFS 時間を T_{SIFS}，*ACK* または *NACK* を伝達するために必要な時間を $T_{\mathrm{(N)ACK}}$ と定義する．これらのパラメータは, IEEE 802.11 規格に基づく固定的なパラメータとして与えられる．また，パケットを伝送するために必要な時間を D_T と定義する． D_T は，パケット長制御およびレート制御に従って変化する．さらに，言うまでもなく， D は D_T に依存している．

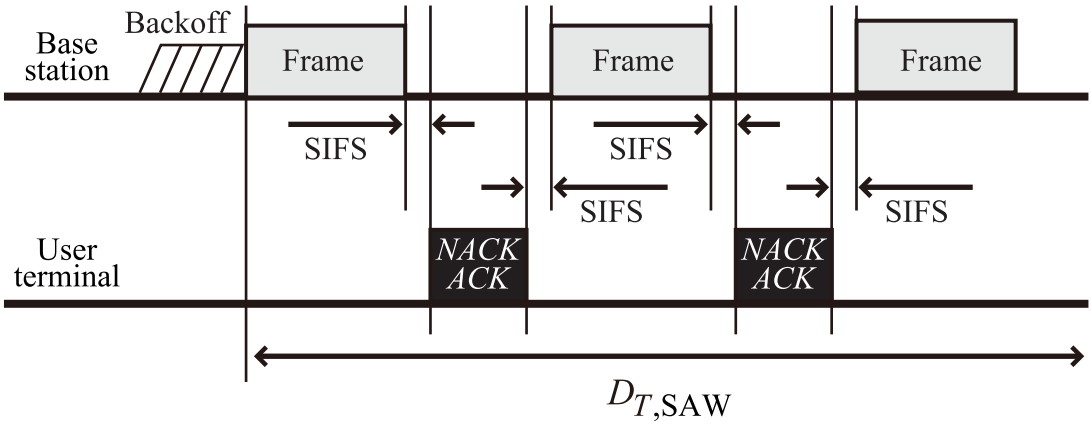
本節の残りの部分において，Hybrid ARQ およびフレーム欠損を考慮する場合におけるパケット伝送遅延時間 D_T の算出を行う．

Hybrid ARQ およびフレーム欠損が無視できる場合　もし，Hybrid ARQ に係る構造上のオーバヘッドを無視することができ，かつフレーム欠損がない場合，パケット伝送遅延 D_T は式（3.3）で計算することができる．

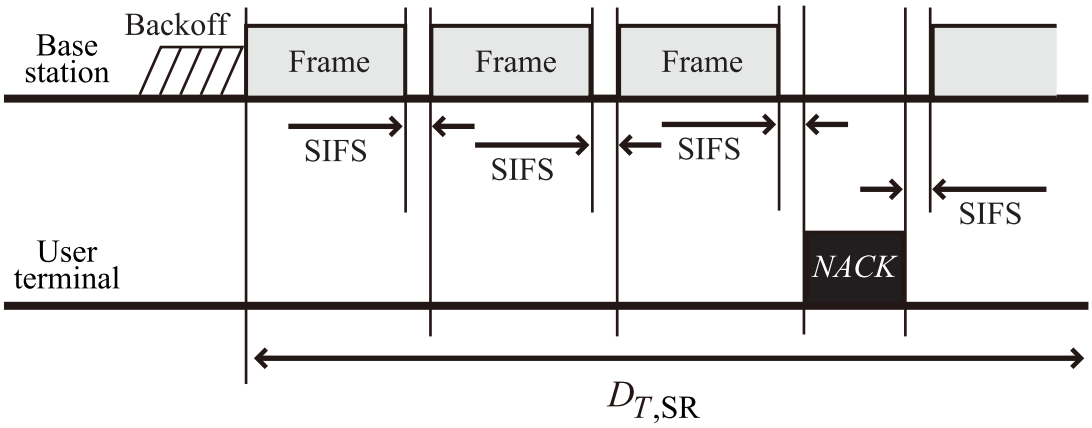
$$D_T = \mu \frac{L}{v} \tag{3.3}$$

ただし，データ伝送レートを v と定義する． v は M 値変調方式と符号化率 R の誤り訂正符号を利用するとき，無線チャネル固有のシンボル伝送速度を V と定義する場合， v を式（3.4）で定義できる．

$$v = V \cdot R \cdot \log_2 M \tag{3.4}$$



(a) Stop-and-wait (SAW) type



(b) Selective repeat (SR) type

図 3.5　Hybrid ARQ を加味したフレーム伝送処理手順

Hybrid ARQ は考慮するが，フレーム欠損は無視できる場合　SAW-ARQ および SR-ARQ の 2 種類の Hybrid ARQ を用いてデータ伝送を行う場合において，そのフレーム伝送処理手順を図 3.5 に示す．このとき，SAW-ARQ および SR-ARQ を用いる場合のパケット伝送遅延時間を $D_{T,\mathrm{SAW}}$ および $D_{T,\mathrm{SR}}$ と定義するとき，図 3.5 に基づき式（3.5）および式（3.6）で計算できる．

$$D_{T,\mathrm{SAW}} = D_T + \mu \times \left[2 \cdot T_{\mathrm{SIFS}} + T_{\mathrm{(N)ACK}}\right] \tag{3.5}$$

$$D_{T,\mathrm{SR}} = D_T + \mu \times T_{\mathrm{SIFS}} \tag{3.6}$$

Hybrid ARQ およびフレーム欠損を共に考慮する場合　もし，Hybrid ARQ に係るオーバヘッドおよびフレーム欠損を共に考慮するとき，式（3.5）および式（3.6）は式（3.7）および式（3.8）のように書き換えられる．ただし，最大フレーム再送回数（初回送信も含む）を J，フレーム誤り率を p_f と定義する．

$$\overline{D}_{T,\mathrm{SAW}} = D_{T,\mathrm{SAW}} \times \left[1 + \sum_{j=1}^{J-1} p_f^j\right] \tag{3.7}$$

$$\begin{aligned}\overline{D}_{T,\mathrm{SR}} = D_{T,\mathrm{SR}} \times &\left[1 + \sum_{j=1}^{J-1} p_f^j\right] \\ &+ (T_{\mathrm{SIFS}} + T_{\mathrm{(N)ACK}}) \times \sum_{j=1}^{J-1} p_f^j\end{aligned} \tag{3.8}$$

また，$J \to \infty$ としたとき，$0 < p_f < 1$ であるため，式（3.7）および式（3.8）の最終項は等比級数の和であることに着目すると，式（3.9）に示すように簡単化できる．

$$\lim_{J\to\infty}\left[1 + \sum_{j=1}^{J-1} p_f^j\right] = \lim_{J\to\infty}\left[\frac{1-p_f^J}{1-p_f}\right] = \frac{1}{1-p_f} \tag{3.9}$$

3.3　適応パケット長制御

提案する適応パケット長制御では，分割フレームを適応的に切替えることにより実現する．このときに用いるフレームの全長は次に示す 2 段階の手順を経て決定する．

- フレーム分割に起因するオーバヘッドを最小化するときの候補フレームの列挙
- 候補フレームの中から，所望パケット伝送遅延時間を満たす最大の全長のフレームを最適フレームとして選択

本節の残りの部分において，提案手法における最適フレーム長の決定手法に関して述べる．

3.3.1 フレーム分割に起因するオーバヘッドの定式化

パケットをフレーム分割するときに生じるオーバヘッドには，フレームヘッダ，およびゼロパディングの２種類がある. フレームヘッダに起因するオーバヘッドは, ユーザが送信したいコンテンツ情報とは無関係なフレームヘッダを追加することによる冗長性である．フレームヘッダは，すべてのフレームに対して一律に付加されるためフレームごとに生じる．一方, ゼロパディングに起因するオーバヘッドは, エッジフレームにおいて，余白部分をゼロパディングすることによる冗長性である．ゼロパディングは，エッジフレームにのみ付加されるためパケットごとに生じる.

フレーム分割に起因するオーバヘッドをδと定義するとき，式（3.10）で表せる.

$$\delta = \mu \frac{L_H}{v} + \frac{L_P}{v} \tag{3.10}$$

式（3.10）で示されるフレーム分割に起因するオーバヘッドδに対してフレーム誤り率を加味する場合，式（3.11）で示すように，フレーム誤り率を加味したフレーム分割に起因するオーバヘッド$\overline{\delta}$を定義できる.

$$\overline{\delta}(p_f, J) = \delta \times \left[1 + \sum_{j=1}^{J-1} p_f^j \right] = \delta \times \frac{1 - p_f^J}{1 - p_f} \tag{3.11}$$

また，$0 < p_f < 1$，および$J \to \infty$とする場合，式（3.9）に基づき，式（3.11）は式（3.12）に示すように簡単化できる.

$$\overline{\delta}(p_f)\big|_{J\to\infty} = \lim_{J\to\infty} \overline{\delta}(p_f, J) = \delta \times \frac{1}{1 - p_f} \tag{3.12}$$

一般的に，無線チャネルにおいてビット誤りがランダムに生じる場合，フレーム誤り率p_fはビット誤り率p_bに基づき，式（3.13）を用いて計算することができる.

$$p_f(L, \gamma_s) = 1 - [1 - p_b(\gamma_s)]^L \tag{3.13}$$

本論文では，ビット誤り率はモンテカルロ法に基づく計算機シミュレーションによって導出する．数値例は 3.5.2 節で示す．

3.3.2　候補フレームの算出

式(3.12)における $\overline{\delta}$ は，δ および p_f をパラメータにもつ．式(3.10)に示すように，δ を構成する L_H は固定的に与えられ，v はレート制御に従い，μ ，L_P は分割フレーム長に基づき決定する．レート制御に関する詳細は 3.4 節にて述べるが，最適なレートは SNR に基づき決定する．一方，式（3.13）に示すように，p_f は分割フレーム長および SNR に依存して与えられる．従って，$\overline{\delta}$ は，分割フレーム長および SNR に依存した指標である．

分割フレームの候補 $\widetilde{L}_i(\widetilde{L}_i \in \widetilde{\mathcal{L}})$ は，式（3.12）に基づく $\overline{\delta}$ を最小とするフレーム長として，式（3.14）から計算できる．ただし，$\widetilde{L}_i$ は非負整数，$\widetilde{\mathcal{L}}$ は $\widetilde{L}_i$ が取り得る集合とする．

$$\widetilde{L}_i = \arg\min_{L \in \mathbb{N}} \overline{\delta}\left(p_f(L, \gamma_s)\right) \tag{3.14}$$

3.3.3　最適フレームの決定

第 2 章で述べた QoS フレームワークを用いてクロスレイヤ設計に基づく適応制御が可能である場合，ユーザの所望パケット遅延時間に応じた分割フレーム長を適応的に切替えることにより，適応的なパケット長制御を実現することができる．3.2.4 節で述べたように，パケット遅延時間 D およびパケット伝送遅延時間 D_T は式（3.2）によって示される．このとき，D を構成する D_T を除くパラメータは IEEE 802.11 規格に基づき固定的に与えられる．従って，式（3.2）は，式（3.3）も加味して，式（3.15）に書き換えられる．

$$D = D_T + \mathrm{const} = \mu \frac{L}{v} + \mathrm{const} \tag{3.15}$$

ユーザの所望パケット遅延時間が既知であるとき，所望パケット伝送遅延時間は式（3.15）に基づき計算することができる．また，L に対しても所望パケット伝送遅延時間を満たしているかどうか判別することが可能である．すなわち，式（3.15）に基づき，L がユーザの所望パケット遅延時間を満たしているかどうかを判別する関数$\mathfrak{D}(L)$を定義するとき，式（3.16）に従い，SNR およびユーザの所望パケット遅延時間を共に考慮した最適フレーム長L^*を計算することができる．

$$L^* = \arg\max_{\widetilde{L}_i \in \widetilde{\mathcal{L}}} \mathfrak{D}(\widetilde{L}_i) \tag{3.16}$$

3.3.4 Hybrid ARQ を加味した最適フレームの決定

3.3.3 節では，Hybrid ARQ に関わるオーバヘッドを考慮しない場合を想定して，最適なフレーム長の決定手法を述べた．Hybrid ARQ を加味する場合，3.3.2 節で述べた候補フレームを選別する際に，オーバヘッドを評価する指標$\overline{\delta}$に対し Hybrid ARQ を導入するコストを加える必要がある．具体的には，式（3.14）において，$\overline{\delta}$の代わりに，SAW-ARQ はδ_{SAW}，SR-ARQ はδ_{SR}に置き換えて計算する．δ_{SAW}およびδ_{SR}は，式（3.17）および式（3.18）で定義する．

$$\delta_{\mathrm{SAW}} = \delta + \mu \times \left[2 \cdot T_{\mathrm{SIFS}} + T_{\mathrm{(N)ACK}}\right] \tag{3.17}$$

$$\delta_{\mathrm{SR}} = \delta + \mu \times T_{\mathrm{SIFS}} \tag{3.18}$$

式（3.14）における$\overline{\delta}$は，フレーム誤りを考慮した指標である．SAW-ARQ に関しては，式（3.17）と等価的に置き換えることが可能である．しかし，SR-ARQ に関しては，フレーム誤り率を加味するために，式（3.8）と同様に，式（3.18）を部分修正した，式（3.19）を用いる必要がある．

$$\overline{\delta}_{\mathrm{SR}} = \delta_{\mathrm{SR}} \times \left[1 + \sum_{j=1}^{J-1} p_f\right] + \left(T_{\mathrm{SIFS}} + T_{\mathrm{(N)ACK}}\right) \times \sum_{j=1}^{J-1} p_f \tag{3.19}$$

表 3.1　AMC に基づく適応レート制御における MCS セット

Index	Modulation scheme	Channel coding rate	Data rate [Mbit/s]
1	BPSK	1/2	6
2	BPSK	3/4	9
3	QPSK	1/2	12
4	QPSK	3/4	18
5	16-QAM	1/2	24
6	16-QAM	3/4	36
7	64-QAM	2/3	48
8	64-QAM	3/4	54

3.4　適応レート制御

適応レート制御を実現するために，Adaptive Modulation and Coding（AMC）手法を導入する [65]．AMC に基づく適応レート制御では，変調方式と通信路符号化方式を組み合わせた Modulation and Coding Scheme（MCS）セットを SNR に基づき切替えることにより伝送レートを適応的に制御する．本論文では，無線 LAN 規格として，IEEE 802.11a を対象として，表 3.1 に示している 8 つの MCS セットを用いる．各 MCS セットに対応したデータレートは，式（3.4）において，V = 12 Mbaud とした場合の理論値である．

また，MCS セットを切替えるしきい値を決定するために，すべてのフレーム長に対して SNR 対 MAC スループットを計算機シミュレーションし，MAC スループットを最大化するときの SNR を算出する．数値例は 3.5.4 節で示す．

3.5 数値例

3.5.1 シミュレーション環境

実際の IP ネットワークを流れるネットワークトラヒックを解析した研究の結果，パケット長の統計的な分布 $\mathcal{X}$ は，式（3.20）に示すように，いくつかのピークを持つ離散分布として近似できることが知られている [66].

$$X(x) = \sum_{\kappa} \omega_{\kappa} \cdot u(x - \lambda_{\kappa}) \quad (x \in \mathcal{X}) \tag{3.20}$$

ただし，$u(a)$ は式（3.21）で定義される単位ステップ関数である．

$$u(a) = \begin{cases} 1 & (a \geq 0) \\ 0 & (a < 0) \end{cases} \tag{3.21}$$

本論文では，文献 [67] における国内の IP ネットワークトラヒックの解析結果に基づき，式（3.20）におけるパラメータとして，{ (ω_{κ} , λ_{κ}); $\kappa = 3$ } = { (0.6, 12,000), (0.2, 10,240), (0.2, 4,608) } を設定する．すなわち，L = 12,000 bit, 10,240 bit，4,608 bit は，RFC1191 におけるイーサネットのデフォルト MTU 値，RFC879 における IPv4 の MTU 値，RFC2460 における IPv6 の最小 MTU 値として一般的に用いられているパケット長である．

表 3.2 にシミュレーション諸元を示す．表 3.2 の各パラメータは，IEEE 802.11a 規格に準拠している．ただし，T_{Backoff} は，$CW_{\min}$ を最小コンテンションウィンドウサイズ，Δ_T をタイムスロット時間と定義するとき，式（3.22）から計算できる．

$$T_{\mathrm{Backoff}} = CW_{\min} \times \Delta_T = 31 \times 9 \quad [\mu\mathrm{s}] \tag{3.22}$$

また，表 3.2 のパラメータを用いることにより，式（3.15）における定数項が決定する．従って，D_T^* を所望パケット伝送遅延時間，D^* をユーザの所望パケット遅延時間と定義するとき，式（3.15）は式（3.23）に書き換えられる．

表 3.2　シミュレーション諸元

Parameter	Value
Data rate	6－54 Mbit/s
(N)ACK duration	28 μs
SIFS duration	16 μs
DIFS duration	34 μs
Backoff duration	279 μs

$$D_T^* = D^* - 357 \quad [\mu\mathrm{s}] \tag{3.23}$$

さらに，本章における計算機シミュレーションでは，解析の簡単化のために次の条件を想定する．無線チャネルにおける無線伝搬損失およびシャドウイングの影響は無視する．また，無線伝送信号処理に必要な無線チャネル推定および無線チャネル等化は理想的に実現される．これらの無線伝搬環境を想定する場合，無線チャネルは Additive White Gaussian Noise（AWGN）が支配的になる．すなわち，式（3.13）が成立するための前提条件である，無線チャネルにおけるビット誤りがランダムに生じる必要性がある点を満たしている．従って，3.3 節で述べた適応パケット長制御を適用することが可能である．

3.5.2 ビット誤り率

本節では，物理層において，提案手法におけるビット誤り率を計算する．図 3.6 に SNR 対ビット誤り率のシミュレーション結果を示す．本シミュレーションは，無線通信を評価するための計算機シミュレータを MATLAB[68] を用いて実装し，モンテカルロ法に基づくシミュレーション結果を示している．また，AWGN 通信路に対し，IEEE 802.11a のすべての PHY モードとして，表 3.1 に示す MCS セットを評価した．具体的には，変調方式は BPSK，QPSK，16-QAM，64-QAM，ビット対シンボルのマッピング規則はグレイ符号を用いている．また，通信路符号化方式は，送信側は拘束長 k = 7 の畳込み符号に基づく符号化，受信側は硬判定ビタビアルゴリズムに基づく復号を行う [69]．畳込み符号の符号化率 R は，パンクチャリングを用いてレート調整を実現している．すなわち，R = 1/2 の畳込み符号，R = 3/4 および R = 2/3 のパンクチャド畳込み符号を用いている [70]．

本章の残りの部分で示している評価は，このビット誤り率を用いて解析している．

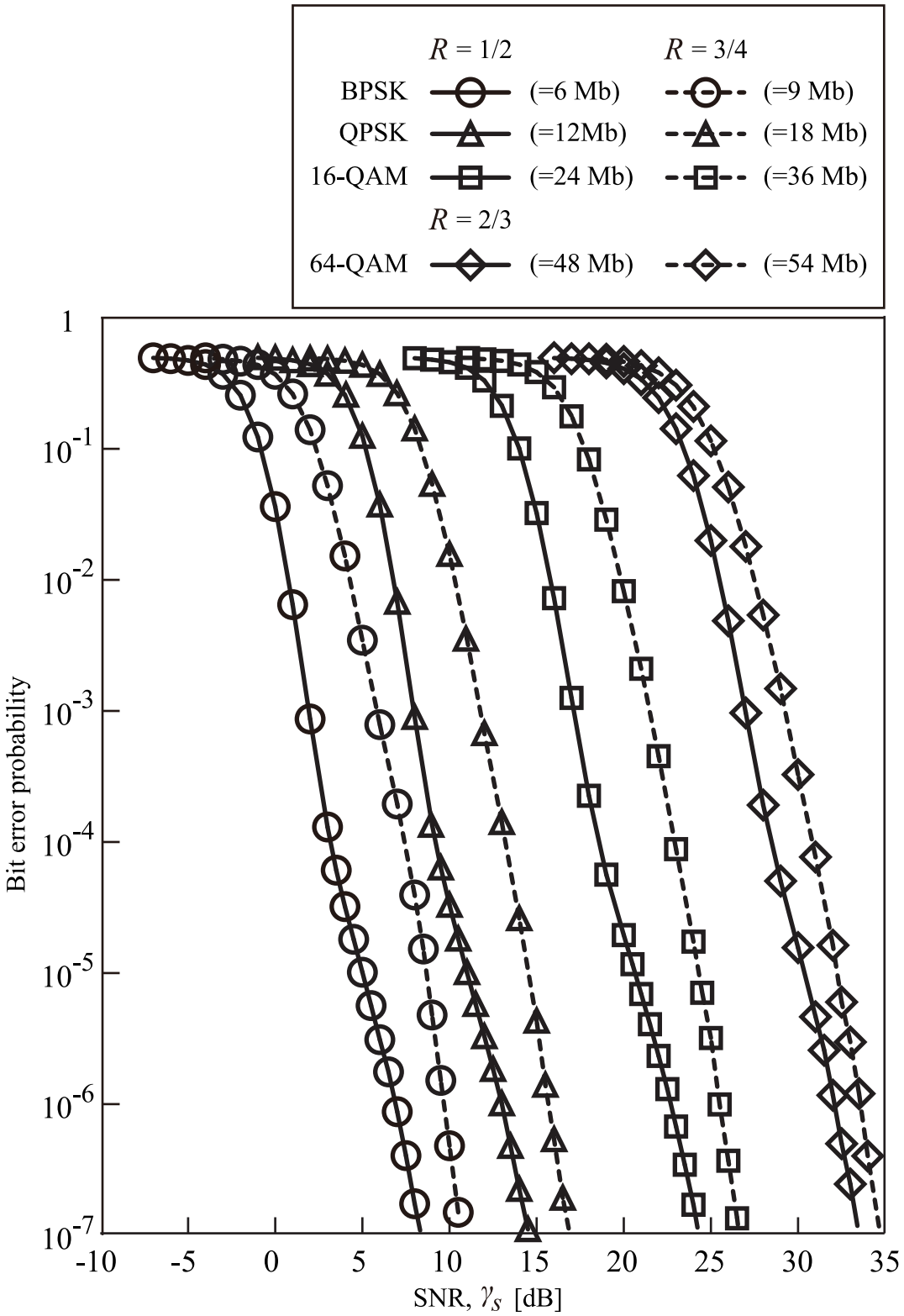


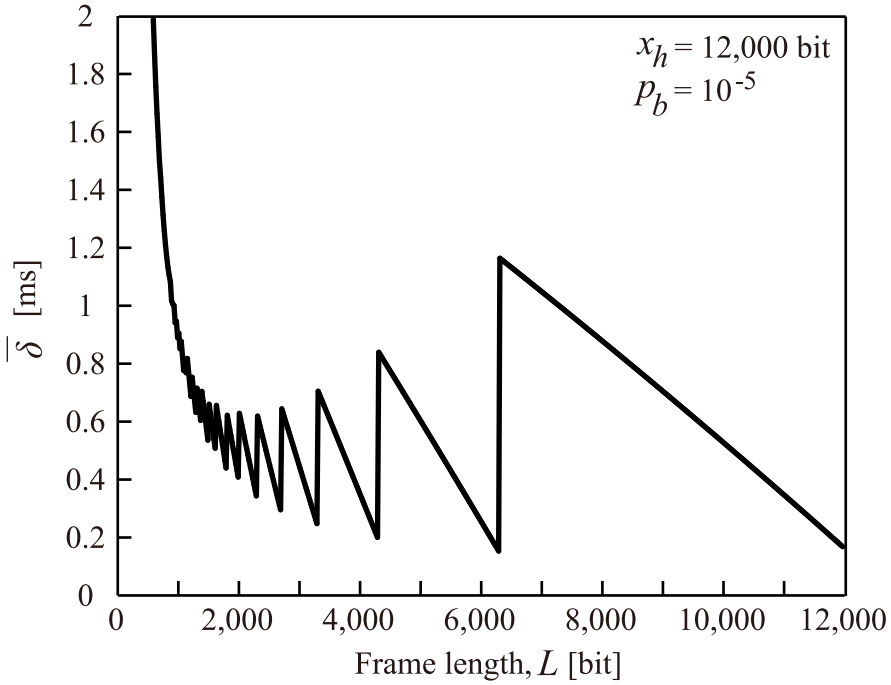
図 3.6　ビット誤り率特性

3.5.3　最適パケット長制御

本節では，数値計算により最適フレーム長を導出する．図 3.7，3.8，3.9 に，フレーム長 L 対フレーム分割に起因するオーバヘッド $\overline{\delta}$ を示す. 図3.7, 3.8, 3.9 では, パケット長 x_h = 12,000 bit，10,240 bit，4,608 bit に対して，無線チャネルのビット誤り率 p_b = 10^{-3}, および p_b = 10^{-5} の結果をプロットしている．ただし, $\overline{\delta}$ は式（3.12）に基づき計算している.

ビット誤り率に関して，一般的に，p_b = 10^{-3} は音声またはストリーミングサービス，p_b = 10^{-5} は信頼性が求められるデータ通信を想定した場合に要求される値である．このビット誤り率の算出根拠は，携帯電話システムで用いられている QoS パラメータを参考に導出しているため，評価環境である IEEE 802.11 の無線 LAN では異なっていることが考えられる．この点に関して，提案手法を IEEE 802.11 の無線 LAN に導入することにより，無線リンクのビット誤り率が p_b = 10^{-3} の場合であっても，ネットワークとして利用できることを期待している．その理由は，標準的な IEEE 802.11 では通信が困難な状況にあったとしても，クロスレイヤ設計を導入することによる適応無線伝送制御により，音声等のアプリケーションサービスは利用可能であると考える．また，少なくとも携帯電話システムでは，音声通話は p_b = 10^{-3} を想定して設計されているため，クロスレイヤ設計に基づく最適化により IEEE 802.11 の環境でも，同様に利用可能であると考えられる.

フレームヘッダは IEEE 802.11 規格に準拠させ，フレームヘッダ長 L_H は 288 bit に設定した．すなわち，MAC フレームヘッダ長が 256 bit，およびフレームチェックシーケンス長が 32 bit である場合を想定している．また，フレーム分割に関わる情報はフレームヘッダに含めることができる場合を想定した．具体的には，パケット分割を実現するために必要な制御情報としては，フレーム分割した対象フレームであるかを判別するフラグ情報，およびフレームのシーケンス番号の情報が挙げられる．これらは，各々，MAC フレームヘッダ内の ‘more fragments’ および ‘sequence control’ 領域が利用可能である.



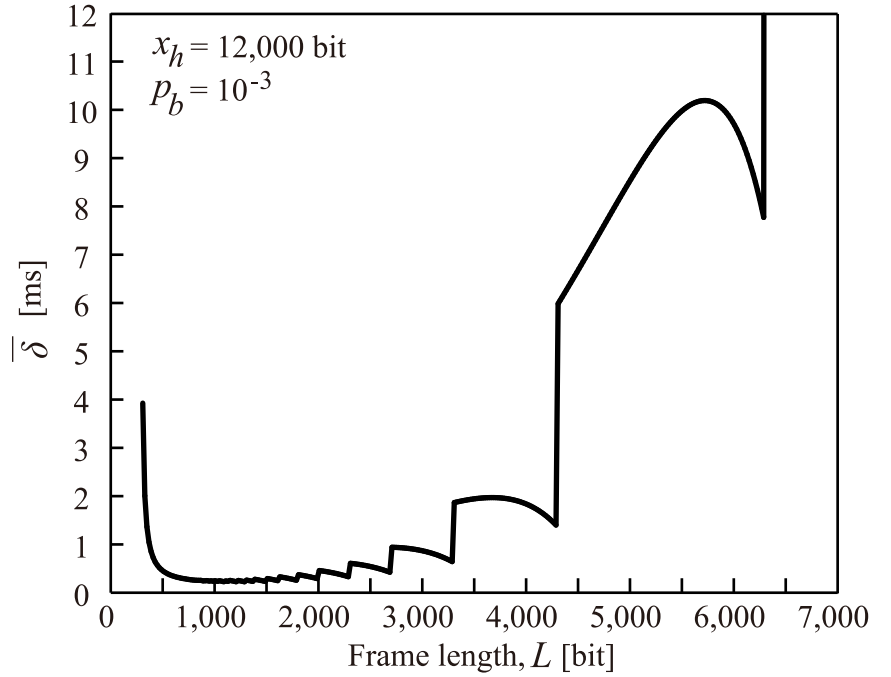
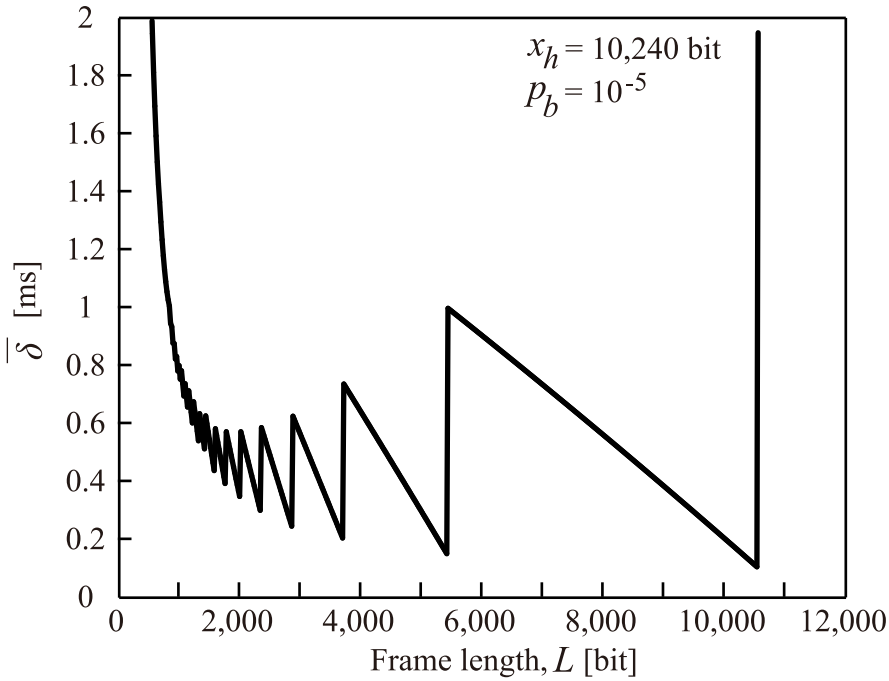


図 3.7　フレーム長対フレーム分割オーバヘッド（x_h = 12,000 bit）



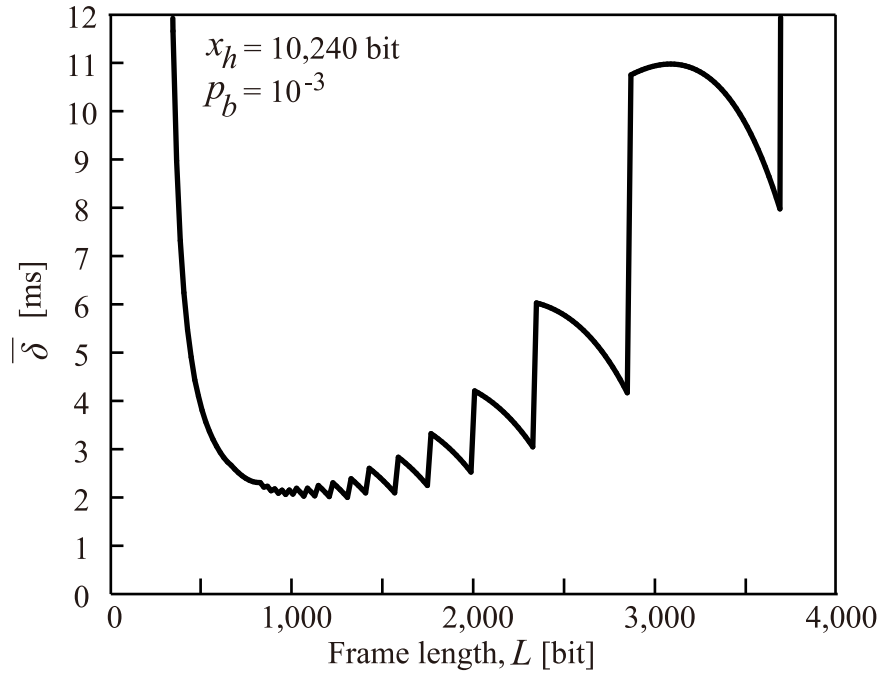
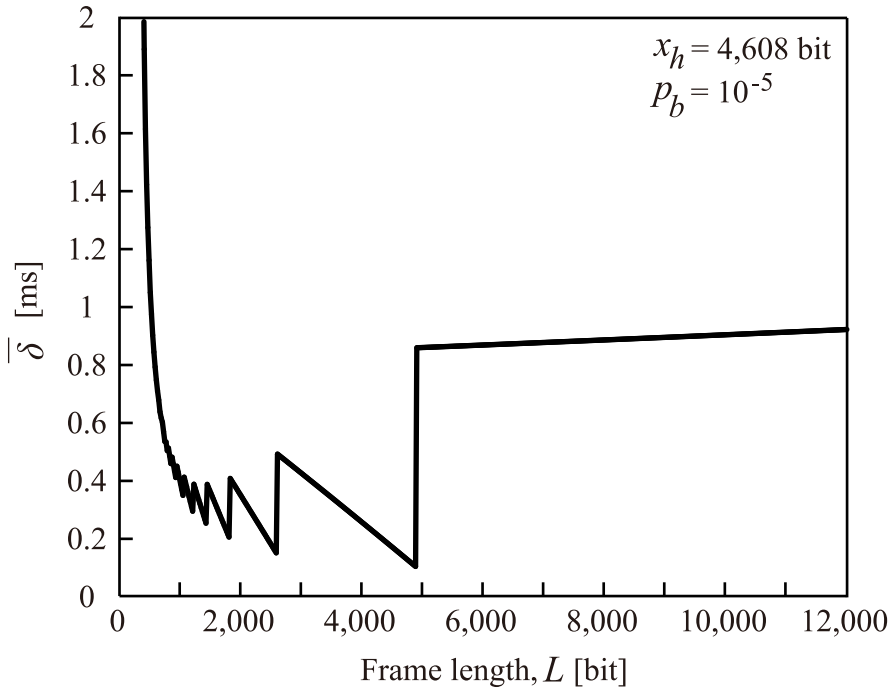


図 3.8　フレーム長対フレーム分割オーバヘッド（x_h = 10,240 bit）



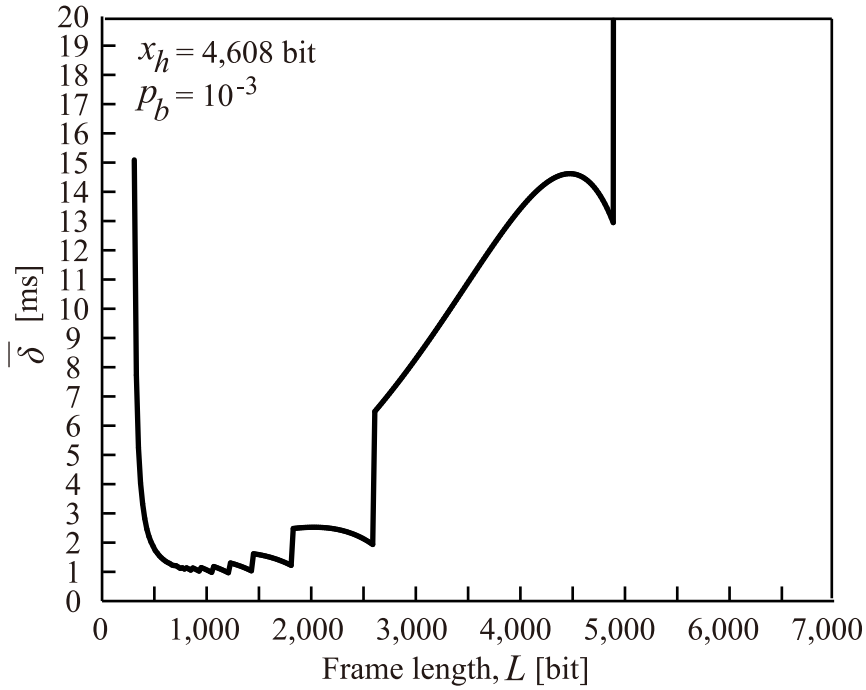


図 3.9　フレーム長対フレーム分割オーバヘッド（x_h = 4,608 bit）

定性的な観点から述べると，図 3.7，3.8，3.9 におけるフレーム長 L 対フレーム分割に起因するオーバヘッド $\overline{\delta}$ の曲線は, (I)ヘッダオーバヘッドが支配的な領域, (II)均衡がとれた領域，（III）ゼロパディングのオーバヘッドが支配的な領域，（IV）ビット誤りが支配的な領域の 4 つの領域に分類することができる．フレーム長が小さいとき, フレームの全長に対してフレームヘッダ長が占める割合が大きくなるため, ヘッダオーバヘッドが与える影響が大きい. すなわち, 領域(I)では, このヘッダオーバヘッドがパケット遅延時間を増大させる主要な要因である．一方，フレーム長がある程度大きくなると，ヘッダオーバヘッドの代わりに，エッジフレームにおけるゼロパディングの影響が大きくなる．すなわち，領域（III）では，フレーム分割による余白部分の影響が大きくなる．従って，ゼロパディングができる限り生じないようなフレーム分割を実現することにより, このゼロパディングに係るオーバヘッドを最小化できる. 領域（II）は，領域（I）と（III）の間に存在する領域である．さらに，フレーム長が十分に大きくなるとき，ビット誤りが増大するため，フレーム再送制御処理の頻度が大きくなる．そのため，領域（IV）では，ビット誤りに起因するフレーム再送によるパケット伝送遅延の低下を招く.

定量的な観点から述べると，図 3.7 における p_b = 10^{-3} の結果に関しては，先述した領域（I）～（IV）は，各々，$L < 700$ bit，$700 \leq L < 1,500$，$1,500 \leq L < 6,300$，$6,300 \leq L$ に対応している．また，$6,500 < L$ の領域では，フレーム長が大きくなることによるビット誤りに起因して，$\overline{\delta}$ は大幅に増大した．一方, 図 3.7 における p_b = 10^{-5} の結果に関しては, 領域（I）～（III）は, 各々, $L < 870$，$870 \leq L < 1,200$，$1,200 \leq L$ に対応している．p_b = 10^{-5} の結果においては，$L < 12,000$ においてもビット誤りが十分に小さいため，領域（IV）はみられなかった．図 3.8 および，図 3.9 の結果に関しても，図 3.7 と同様の結果が得られた.

3.3.2 節で述べた候補フレームの決定手法に従い，式（3.14），および図 3.7，3.8，3.9 に基づき候補フレームを決定した．その結果を表 3.3 にまとめた．また，3.3.3 節の最適フレーム長の決定手法に従い，表 3.3 から最適フレーム長を決定する．例えば，x_h =12,000 bit，および p_b = 10^{-3} の場合，もし所望パケット伝送遅延時間が 12,000 μ s であれば，最適フレーム長として，SAW-ARQ に対して L^* = 1,088 bit, SR-ARQ に対して L^* = 1,288 bit が得られる.

表 3.3　候補フレームの算出結果

x_h [bit]	p_b	L [bit]	$\overline{\delta}$ [us]	$\overline{D}_{\mathrm{T,SAW}}$	$\overline{D}_{\mathrm{T,SR}}$
12,000	10^{-3}	1,288	2,264	12,954	10,994
		1,208	2,273	12,252	10,293
		1,088	2,281	11,382	9,436
		1,488	2,340	15,014	13,020
		1,368	2,342	13,583	11,636
	10^{-5}	12,288	109	4,767	4,673
		6,288	153	3,540	3,402
		4,288	200	3,234	3,052
		3,288	248	3,142	2,916
		2,668	313	3,110	2,841
10,240	10^{-3}	1,308	1,979	11,318	9,646
		1,208	1,996	10,502	8,837
		1,068	2,005	9,700	7,991
		1,128	2,011	9,966	8,290
		968	2,040	9,328	7,545
	10^{-5}	10,528	107	4,032	3,939
		5,408	152	3,044	2,907
		3,688	206	2,800	2,619
		2,828	261	2,733	2,508
		2,328	302	2,751	2,482
4,608	10^{-3}	1,208	969	5,251	4,470
		1,048	982	4,687	3,890
		928	1,026	4,346	3,522
		1,428	1,035	6,218	5,440
		848	1,059	4,233	3,367
	10^{-5}	4,888	102	1,837	1,747
		2,588	149	1,513	1,378
		1,808	170	1,472	1,293
		1,428	204	1,511	1,289
		920	252	1,296	1,029

表 3.4　適応レート制御における SNR のスレショルド値

MCS set *m*	Data rate [Mbit/s]	*L* = 1,288 bit [dB]	*L* = 12,288 bit [dB]
BPSK+*R*=1/2	6	~ 6.7	~ 8.1
BPSK+*R*=3/4	9	6.7 ~ 8.8	8.1 ~ 10.1
QPSK+*R*=1/2	12	8.8 ~ 12.5	10.1 ~ 13.8
QPSK+*R*=3/4	18	12.5 ~ 18	13.8 ~ 19.8
16-QAM+*R*=1/2	24	18 ~ 22.3	19.8 ~ 23.6
16-QAM+*R*=3/4	36	22.3 ~ 27.9	23.6 ~ 29.7
64-QAM+*R*=2/3	48	27.9 ~ 30.8	29.7 ~ 32.3
64-QAM+*R*=3/4	54	30.8 ~	32.3 ~

3.5.4　最適レート制御

本節では，表 3.3 に示している x_h = 12,000 bit における，p_b = 10^{-3}およびp_b = 10^{-5}の最初の候補フレーム L = 1,288 bit および L = 12,288 bit に対し，適応レート制御を実現するための SNR のスレショルド値を示す．図 3.10，3.11 に，IEEE 802.11a のすべての PHY モードに対し，無線チャネルとして AWGN を想定した場合における SNR 対 MAC スループットの計算機シミュレーション結果を示す．計算機シミュレーションは，モンテカルロ法に基づき，MATLAB を用いて実装した．

例えば，図 3.10 において，$0 \leq \gamma_s \leq 6.7$ の範囲では，MAC スループットの最大値は 4.65 Mbit/s になる．また，$6.7 < \gamma_s$ の範囲では，MAC スループットを最大化するために，MCS セットを次のステップに切替える．すなわち，BPSK と R = 1/2 から BPSK と R = 3/4 に切替える．

図 3.10，3.11 に基づき，同様の規則に従い，表 3.4 に示すように，SNR に応じた MCS セットを適応的に切替えることにより，伝送レートを適応的に制御する．

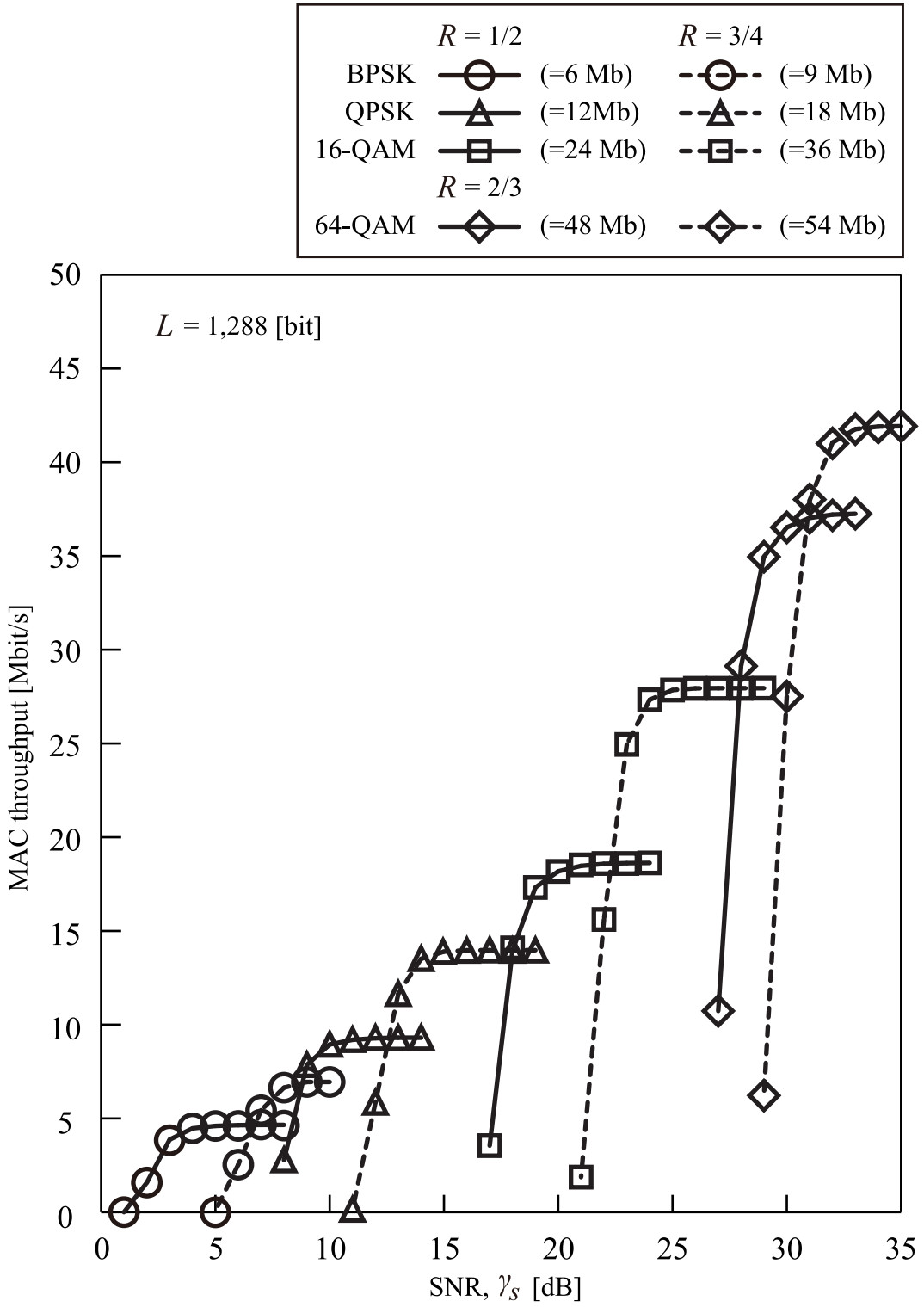


図 3.10　SNR 対 MAC スループット（L = 1,288 bit）

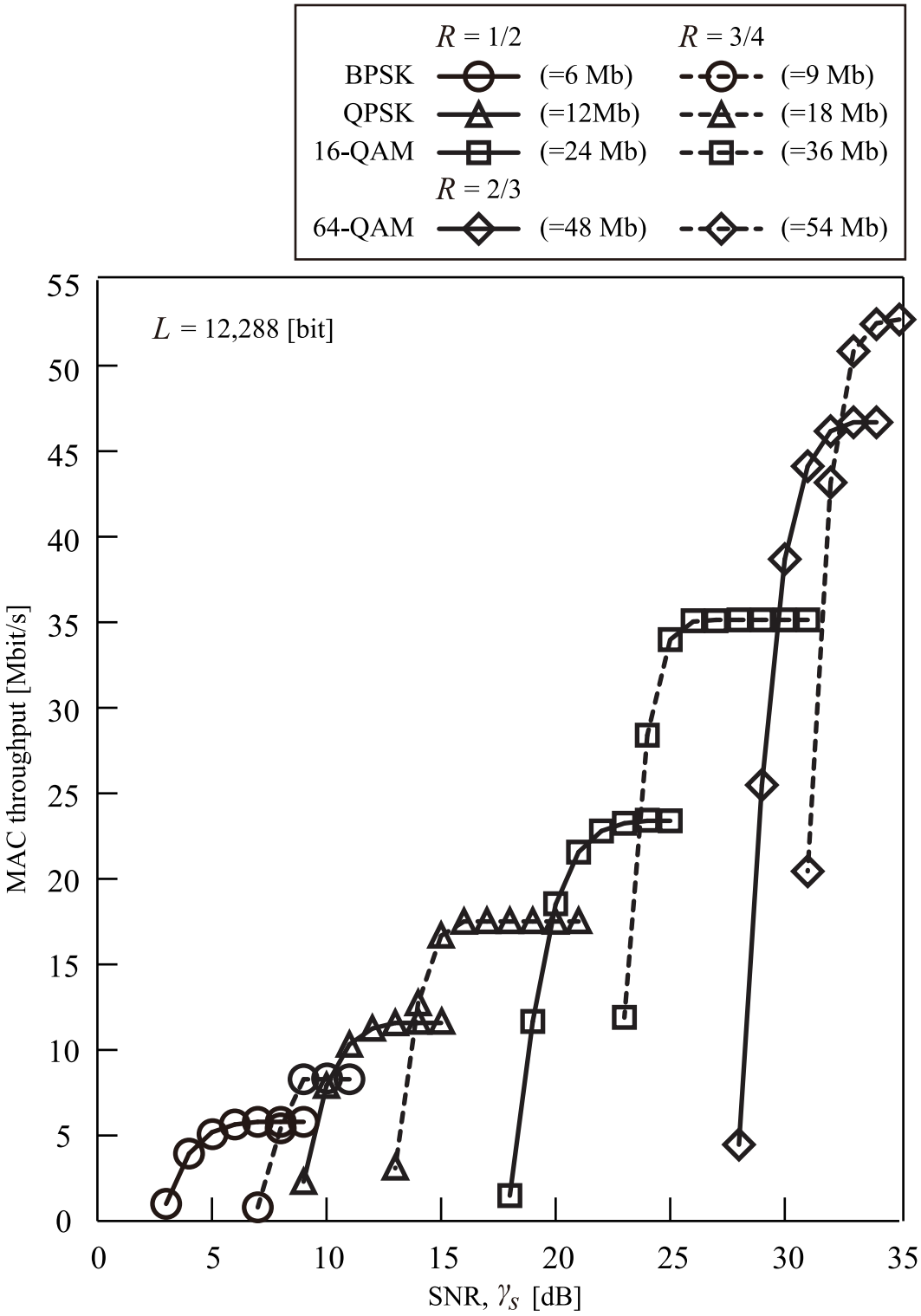


図 3.11　SNR 対 MAC スループット（L = 12,288 bit）

表 3.5　従来手法におけるフレーム分割オーバヘッドおよびパケット伝送遅延時間

x_h [bit]	p_b	δ [μs]	$D_{T,\mathrm{SAW}}$ [μs]	$D_{T,\mathrm{SR}}$ [μs]
12,000	10^{-3}	3,421	17,280	15,200
	10^{-5}	527	2,663	2,305
10,240	10^{-3}	2,927	15,120	13,331
	10^{-5}	451	2,330	2,017
4,608	10^{-3}	2,213	3,935	1,962
	10^{-5}	386	999	865

3.5.5　従来手法との比較

提案手法の有効性を示すために，従来手法 [71] と比較検証を行う．文献 [71] では，無線 LAN システムを対象として，MAC スループットを改善するためのパケット長制御手法を提案している．パケットをフレーム分割することによりパケット長制御を実現している点は提案手法と類似している．しかし，従来手法では，分割に伴うオーバヘッドを考慮していない点，適応レート制御を考慮していない点，最適フレーム長の算出に計算機シミュレーションに基づく全探索を行っている点が異なる．従来手法においては，分割フレーム長は 160 bit，1,600 bit，16,000 bit のいずれかを用いることにより MAC スループット特性を最大化できる．この知見に基づき，パケット分割に伴うオーバヘッド，およびパケット伝送遅延時間を表 3.5 にまとめた．

また，表 3.6 に，x_h =12,000 bit において，提案手法および従来手法に対するパケット分割に伴うオーバヘッド，およびパケット伝送遅延時間の比較を示す．表 3.6 より，提案手法はパケット分割に伴うオーバヘッドを $p_b = 10^{-3}$ のとき

表 3.6　従来手法と提案手法の比較

x_h [bit]	p_b	Conventional	Proposed	Percentage
Fragmentation overhead	10^{-3}	3,421	2,264	33.8%
	10^{-5}	527	109	79.4%
Packet delay	10^{-3}	4,850	3,960	18.4%
	10^{-5}	1,400	1,390	0.714%

34.0%，$p_b = 10^{-5}$ のとき 79.4%，パケット伝送遅延を $p_b = 10^{-3}$ のとき 18.4%，$p_b = 10^{-5}$ のとき 0.714% 改善できる．従って，提案手法における無線 LAN システムに対するクロスレイヤ適応制御の有効性が示された．

3.6　おわりに

本章では，クロスレイヤ設計に基づく適応制御手法として，レート制御を加味したパケット分割に基づくパケット分割手法を提案した．最適なフレーム長および最適なレートは，将来の無線ネットワークシステムが取り扱うことが重要視されるマルチメディアサービスを考慮して，パケット遅延および CSI に基づき決定した．また，提案手法を無線 LAN システムに導入した場合を想定して数値例を示した．

第４章
QoS 情報の管理手法

本章では，ヘテロジニアスネットワークについて概観した後，実際に提案手法を導入する場合に必要なプロトコル設計を提案する．具体的には，ヘテロジニアスネットワーク環境のモデル化，通信ネットワークの構成，および QoS 情報を共有するために必要なプロトコル設計について述べる．

4.1　はじめに

無線ネットワークシステムが利用される分野が多様化するにつれ，無線ネットワークシステムに対する要求は，導入する環境や目的に応じて異なってくる．一方，あらゆる情報はネットワークを介して伝送されるため，通信ネットワークシステムは必然的に多様なネットワークが統合されたヘテロジニアスネットワークになる．とくに，ユーザに最も近いラストワンマイルをコードレス化する目的で用いられている無線ネットワークシステムは幅広く利用されているため，特定の無線ネットワークシステムだけを対象とする設計では不十分である．従って，将来の無線ネットワークシステムに対して提案手法を導入することを考えると，異なる無線ネットワークシステムにおいて，連携したクロスレイヤ適応制御を実現する必要がある．

以上の状況を鑑み，提案手法では，第 2 章で述べた QoS フレームワークの中で，QoS 情報を規格化したクロスレイヤ制御情報を QoS ポリシーとして取り扱う．そして，QoS ポリシーを共有することにより，単一の無線ネットワークシステムにおける複数のレイヤ間の情報共有にとどまらず，複数無線ネットワークシステムにおける複数のレイヤ間に対して QoS 情報を共有できるように拡張を図る．具体的には，ヘテロジニアスネットワークを想定して，複数の異なる無線ネットワークシステムをユーザが移動する場合において，QoS ポリシーを SIP を用いて共有する手法を述べる．これにより，Cross-layer optimizer が適応制御する際に，柔軟かつオープンなクロスレイヤ制御情報へのアクセスが実現できる．

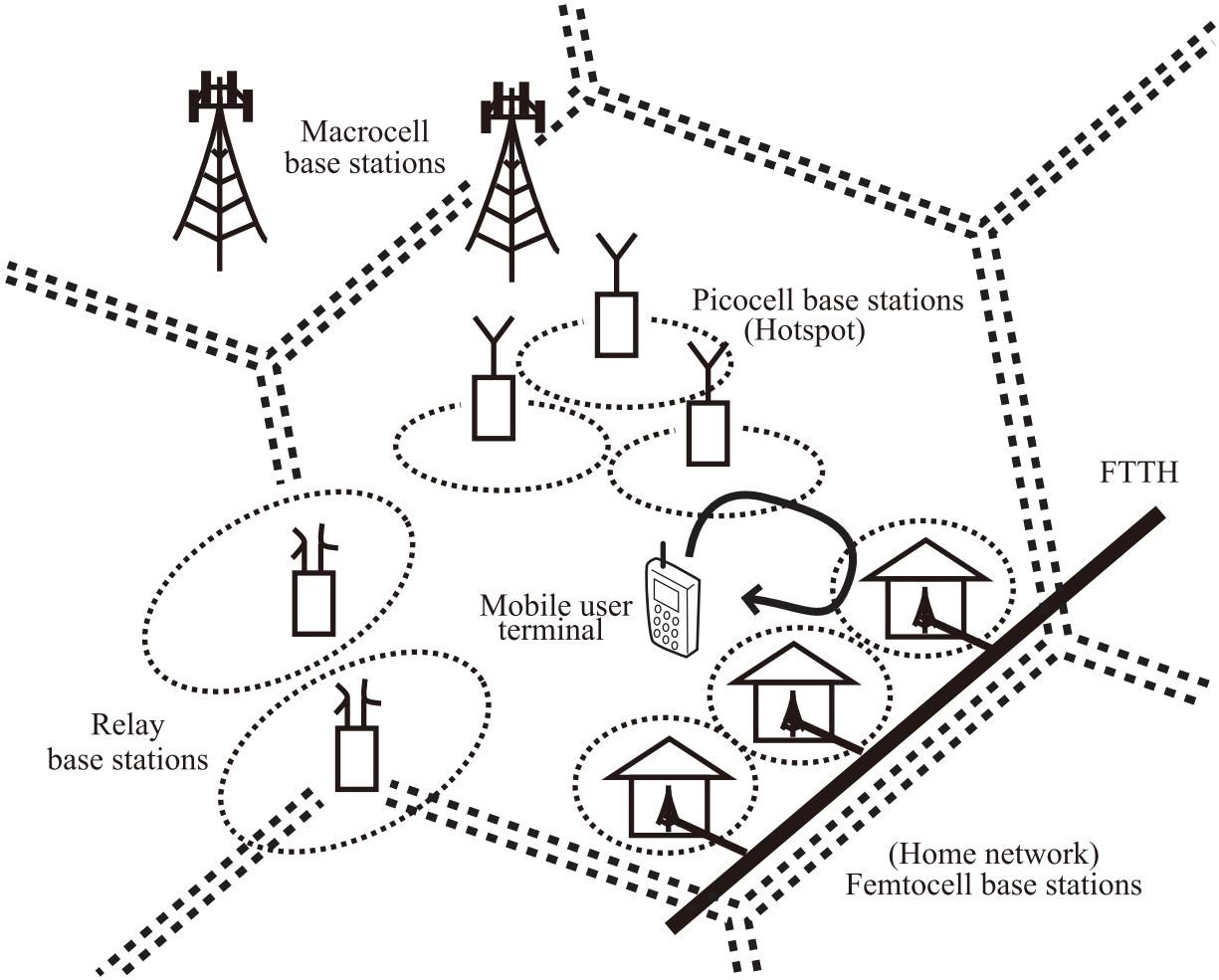


図 4.1　ヘテロジニアスネットワークのモデル

4.2　ネットワークモデル

4.2.1　ヘテロジニアスネットワークのモデル化

将来の無線ネットワークシステムでは，ヘテロジニアスネットワークと呼ばれる複数の異なる無線通信規格が混在するネットワークで構成されている．すなわち，ユーザが通信可能なエリアは，データ伝送帯域は狭いが広くカバレッジを確保できるマクロセル，および広帯域なデータ伝送帯域を確保できるが通信エリアが小さいスモールセルを併用する．一般的に，マクロセルはセルラシステム，スモールセルはセルラシステムと無線 LAN システムを用いて運用されている．

提案手法を導入するヘテロジニアスネットワークのネットワークモデルを図 4.1 に示す．セルラシステムにおけるマクロセルの構成は，無線基地局において初期の固定周波数を繰り返しから，第 3 世代セルラシステム以降は同一周波数の利用へと発

展した．これに対し，同一周波数の 1 セル（または 1 セクタ）繰り返し利用を原則としながら，高送信電力の基地局が形成するマクロセル内に，低送信基地局が形成するピコセルと呼ばれるスモールセルを重畳したセル構成をとっている．ピコセルを導入することにより，マクロセル内でネットワークトラヒックが集中するホットスポットを解消することができるため，ネットワーク輻輳に伴うデータ伝送品質の低下を避けることができる．しかし，マクロセルのセル端エリアでは，無線伝送に必要な電波強度が十分に得られないことにより，データ伝送品質の低下が生じる．そこで，セル端エリアの無線チャネルの状況を改善するためにリレー局を導入したピコセルを用いる．

他方，限られた周波数資源の中で十分なデータ伝送帯域を確保するために，データオフローディングがなされている．データオフローディングは，フェムトセルにネットワークトラヒックの一部を誘導する技術である．フェムトセルは，ユーザの自宅に引かれている光ファイバーを用いた有線ネットワークシステムで構成されるホームネットワークにおいて，そのラストワンマイルに対し，無線 LAN システムを用いて無線伝送化することにより実現する．フェムトセルにデータオフローディングを行うことにより，通信事業者の立場では，ひっ迫するセルラシステムのデータ伝送帯域を少しでも解放される利点がある．一方，ユーザの立場では，自宅のホームネットワークにフェムトセルを設置しているため，広帯域なデータ伝送帯域を安価に利用することができる．

本章の残りの部分において，本節で述べたヘテロジニアスネットワークの中をユーザが自由に移動する場合を想定し，各ネットワークを縦断的に移動する場合において，ユーザが在圏するネットワークに対して QoS 情報を統一的に取り扱うための QoS 情報管理手法を検討する．とくに，マクロセルとピコセルは同じセルラシステムを用いて運用されているため大きな問題は生じないが，マクロセルとフェムトセルはセルラシステムと無線 LAN システムを縦断しなければならないため，異種システム間での調停が必要不可欠である．

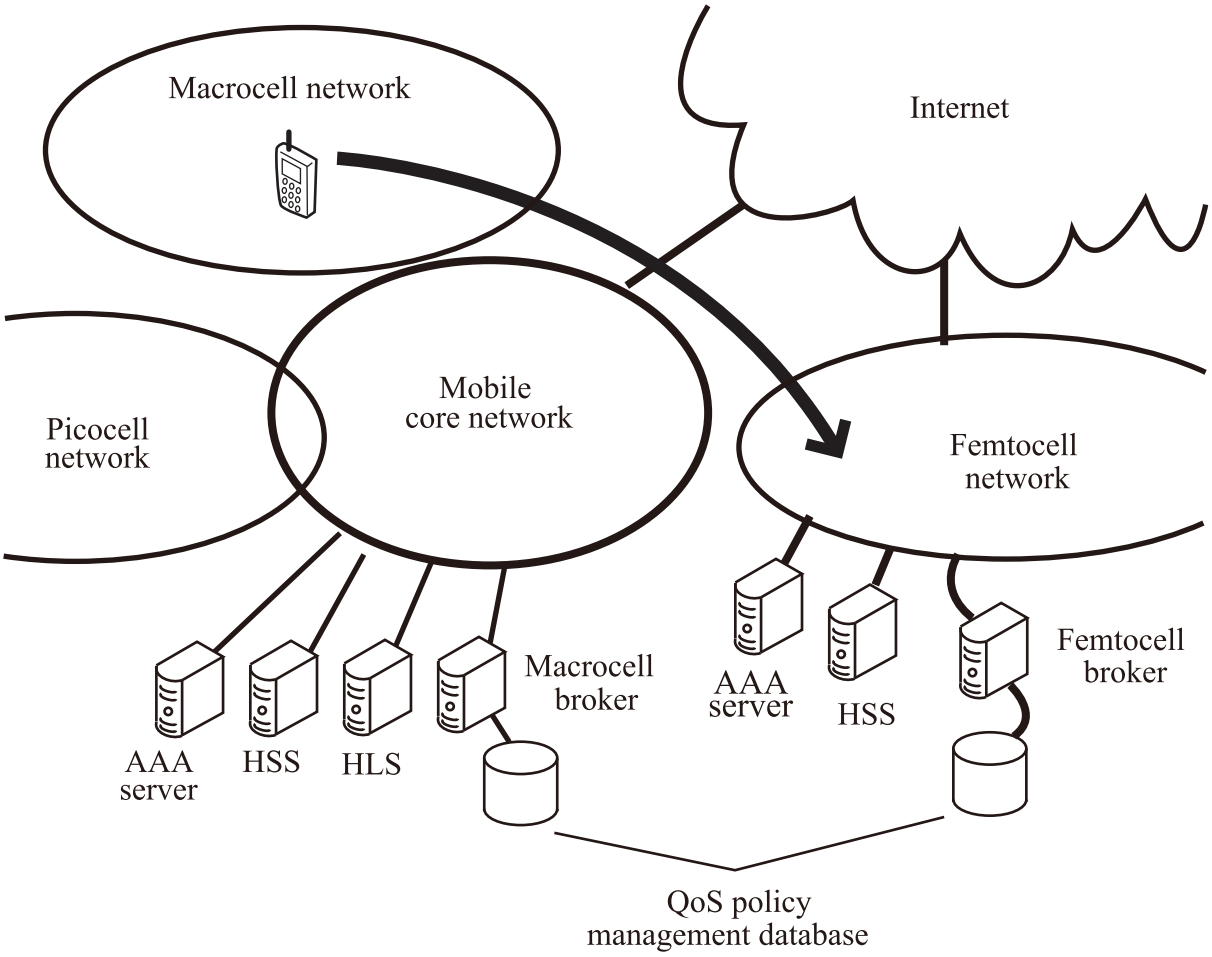


図 4.2　提案手法におけるネットワーク構成

4.2.2　ネットワーク構成

複数のセルから構成された無線ネットワークシステムは，複数の基地局と，それらの基地局を収容した複数のスイッチを用いて，最終的にはインターネットに接続する．図 4.2 に提案手法のネットワーク構成を示す．マクロセルおよびピコセルの基地局は，各々，マクロセルネットワークおよびピコセルネットワークに収容される．また，マクロセルネットワークとピコセルネットワークは，セルラシステムを運用する通信事業者のコアネットワークを経て，インターネットに接続される．一方，フェムトセルの基地局はフェムトセルネットワークに収容される．このフェムトセルネットワークは，自宅のホームネットワークだけではなく，公衆無線 LAN システムを提供する通信事業者のネットワークの場合もある．また，フェムトセルに在圏するユーザ端末は，フェムトセルネットワークを経てインターネットに接続する．

ヘテロジニアスネットワークの中を自由にユーザが移動する場合，ユーザの移動先のネットワークにおいても，移動元のネットワークと同様のモバイルサービスを享受できなければならない．すなわち，ハンドオーバが生じた場合における，シームレスなネットワーク切替え手法を考慮する必要がある．無線ネットワークシステムでハンドオーバを実現するために，提案手法では，ネットワークに接続可能なユーザの認証などを行うために Authentication, Authorization and Accounting（AAA）サーバ, ユーザに関する契約情報を管理するために Home Subscriber Server（HSS），ユーザ端末の位置情報を管理するために Home Location Server（HLS）を設置する．

AAA サーバは，無線ネットワークシステムにおけるユーザの利用資格の有無の認証，ユーザのネットワーク利用実績の記録，およびネットワークの利用が有料の場合の課金を一元的に行う．ヘテロジニアスネットワークにおいては，ユーザ端末がハンドオーバする際に，そのユーザが移動先のネットワークの正統なユーザであることを識別する機能を提供する．ユーザの認証は，ユーザ端末から直接的に依頼するのではなく，そのユーザが在圏する基地局が代行して AAA サーバに問い合わせを行う．

HSS はユーザ端末におけるユーザ契約情報，ユーザが在圏するネットワーク（または，セルの ID 番号）の情報を管理する．HSS が保持するユーザ情報は，ヘテロジニアスネットワークに属する無線ネットワークシステムで共用され，基地局からの依頼に応じて各サーバに転送される．また，HLS はユーザの位置情報を管理し，ユーザが在圏しているセルのスイッチを適切に切替えてネットワークへの接続を保証する．また，HSS と連携して，ユーザが在圏するネットワークシステムを円滑に利用できるようにする．一方, ヘテロジニアスネットワーク内で統一的にユーザの位置を管理し，必要に応じてユーザの位置情報が各サーバに転送される．HSS および HLS は，セルラシステムのコアネットワークに配置する．その理由は，カバレッジが最も広いマクロセルは，ユーザ端末を確実に捕捉可能であるため，ユーザに関する情報はマクロセルが接続するコアネットワークに配置するのが効率的であるからである．

また，各ネットワークに，QoS ポリシーを管理するブローカと，QoS ポリシーの情報を保管する QoS ポリシー管理データベースを設置する．すなわち, QoS ポリシーはブローカを通して QoS ポリシー管理データベースに保存され，必要に応じてユーザ端末はブローカを経由して QoS ポリシーの情報を引き出す．

4.3 QoS ポリシー管理法

本節では，通信のセッションを管理する SIP について概観した後，ユーザがヘテロジニアスネットワーク内を移動する場合における，ハンドオーバ手順およびネットワークの再構築手法について述べる．

4.3.1 Session Initiation Protocol（SIP）の概要

移動通信ネットワークにおいて，VoIP や動画のストリーミング配信のようなマルチメディアサービスを提供するためには，ユーザ端末は着信待ち受けのために移動通信コアネットワークに登録されている必要がある．そこで，一般的に，セッションを管理するために，SIP を用いたセッションを確立する手法が利用されている．SIP はテキストベースのプロトコルで，Web コンテンツの転送に用いられる Hyper Text Transfer Protocol（HTTP），電子メールの送信に用いられる Simple Mail Transfer Protocol(SMTP)に類似する機能を具備する. すなわち, アドレス解決機能, およびセッション確立機能がある．

アドレス解決機能はユーザ端末における固有 ID に基づき，IP アドレスとユーザ端末を対応づける．すなわち，ユーザ端末に対して，任意の IP アドレスやユーザ名を関連づけられるため，ユーザ端末が移動した場合であっても無線ネットワークシステムを利用可能な状態を維持することができる．セッション確立機能は，無線伝送を開始するために，必要とされるセッションを確立することを目的としたセッションへの招待依頼処理，システム障害に対して修復依頼処理，セッション確立に失敗した場合の再送依頼処理である．

4.3.2 ハンドオーバ手順

ユーザ端末の位置情報は, 随時, HLS において管理されているため, マクロセルネットワークの無線基地局は，ユーザがフェムトセルネットワークに在圏していることを検知することが可能である．そして，マクロセルネットワークの無線基地局は，ユーザに対してフェムトセルネットワークへのハンドオーバが可能であることを示唆する通知を伝達することにより，ユーザはハンドオーバが可能であることを知ることができる．さらに, ユーザは端末自体が保有する位置情報, またはフェムトセルネットワー

クの無線基地局が送信しているビーコンの検知することにより，フェムトセルネットワークに在圏していることを知ることができる．このような状況において，ユーザがマクロセルネットワークからフェムトセルネットワークに移動する場合，提案手法におけるハンドオーバ手順を図 4.3 に示す．

本節の残りの部分において，ハンドオーバに必要な手続きについて述べる．

まず，①ユーザはマクロセルネットワークの無線基地局に対して，移動先のフェムトセルネットワークへのハンドオーバ要求を出す．そして，②マクロセルネットワークの無線基地局は，ユーザの契約情報および位置情報を取得・更新するために，HLS および HSS に対してユーザ情報の更新要求を出す．具体的には，HLS に対してユーザの位置情報，HSS に対してユーザが在圏する無線ネットワークの情報（現在属しているネットワークおよびセルの ID 番号）を更新する．また，③フェムトセルネットワークにおけるユーザ認証に必要なユーザの契約情報を，マクロセルネットワークの無線基地局は HSS より受け取る．

マクロセルネットワークにおけるユーザの転出に関する手続きが完了した後，④マクロセルネットワークの無線基地局はフェムトセルネットワークの無線基地局に対して，ユーザの登録要求を出す．すなわち，当該ユーザが移動先のフェムトセルネットワークを利用可能であるかどうかを確認する．そこで，⑤フェムトセルネットワークの無線基地局は，ユーザの契約情報に基づきフェムトセルネットワークを利用可能なユーザであるかどうかを認証するために，フェムトセルネットワークの AAA サーバに対して照会・認証を行う．当該ユーザがフェムトセルネットワークを正統に利用することができることを確認した後，⑥フェムトセルネットワークの無線基地局はマクロセルネットワークの無線基地局に対してユーザの受け入れ手続きが完了したことを通知する．

ユーザのハンドオーバに際したユーザ認証処理が完了した後，移動先のフェムトセルネットワークに対して，QoS ポリシーの転送処理を行う．具体的には，⑦マクロセルネットワークの無線基地局はマクロセルネットワークのブローカに対して QoS ポリシーの情報をフェムトセルネットワークへの転送要求を出す．⑧マクロセルネットワークのブローカは QoS ポリシー管理データベースに接続して，必要とされる QoS ポリシーの情報を受け取り，フェムトセルネットワークのブローカに向けて転送処理を行う．フェムトセルネットワークのブローカはすべての QoS ポリシーの情報を正

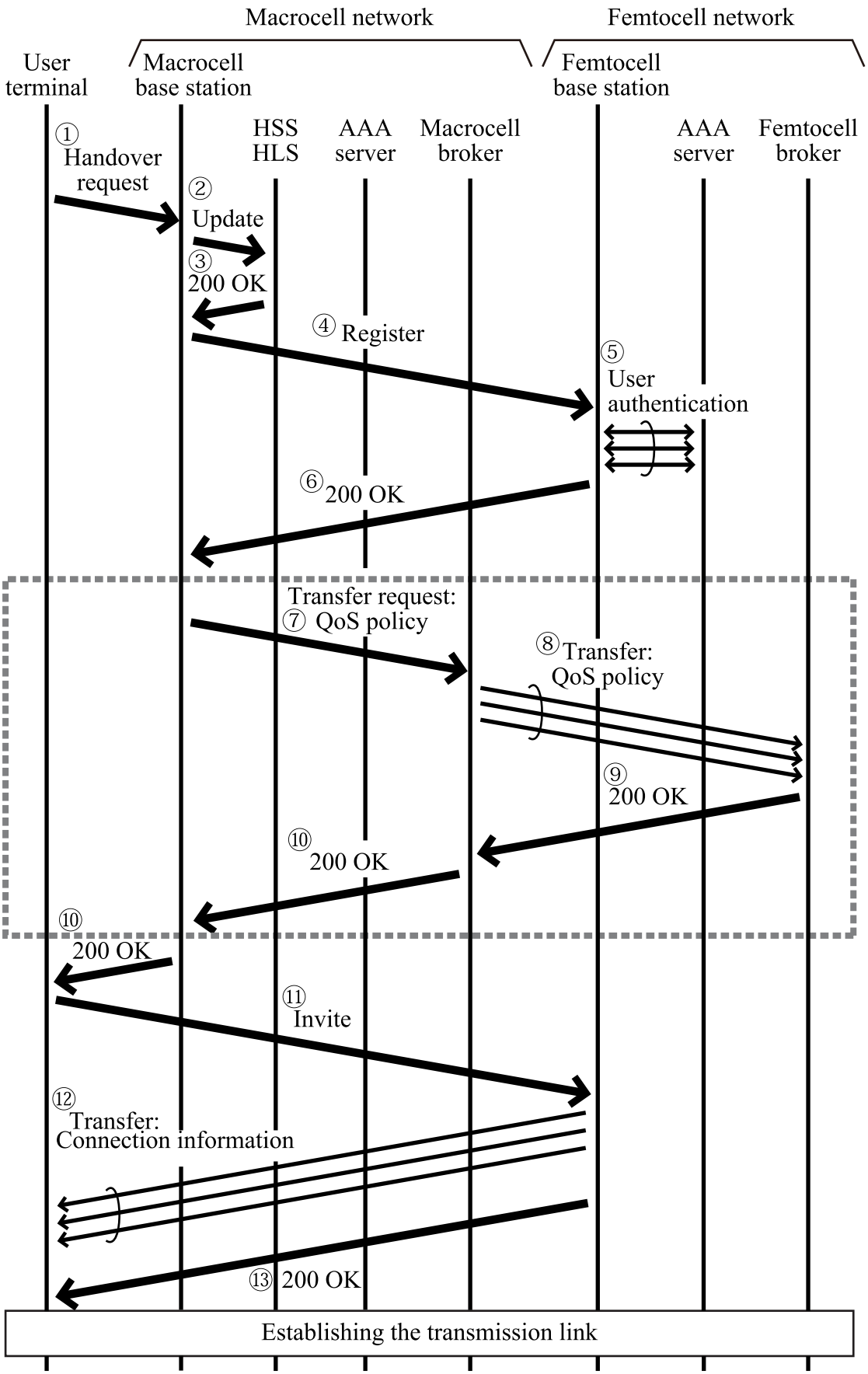


図 4.3　ハンドオーバ時における QoS ポリシーの転送手順

しく受け取り，フェムトセルネットワークの QoS ポリシー管理データベースに保存した後，⑨マクロセルネットワークのブローカに対して QoS ポリシーの転送完了を通知する．その後，⑩マクロセルネットワークの無線基地局を経て，ユーザにハンドオーバに必要な手続きが完了したことを通知する．

最後に，ユーザは移動先のフェムトセルネットワークで通信を行うために必要な情報を取得する．そのために，⑪ユーザは INVITE メッセージを用いて，⑫フェムトセルネットワークの無線基地局から気付アドレス（CoA: Care-of Address）などを取得して，⑬ネットワークの再設定を行う．伝送リンクを再構築して，ネットワークへの再接続設定の実現に関しては 4.3.3 節で述べる．

4.3.3　ネットワークへの再接続設定

同一の無線ネットワークシステム内においてセル間をユーザが移動することに伴う水平型ハンドオーバに関しては，その無線ネットワークシステムで閉じた移動管理の処理を実現することが可能である．一方，ヘテロジニアスネットワークにおいて，異なる無線ネットワーク間をユーザが移動することに伴う垂直型ハンドオーバに関しては，アドレス変更を伴うネットワークへの再接続設定が必要不可欠である．このとき，マクロセルネットワークからフェムトセルネットワークにユーザが移動する場合，セルラシステムで規定されているユーザの移動管理方式と，無線 LAN システムにおける移動管理方式の間での協調動作が必要になる．このネットワークへの再接続を管理するプロトコルには，モバイル IP 方式がある．

モバイル IP 方式では，ユーザ端末にモバイル IP クライアントと呼ばれる移動管理機能が必要になる．具体的には，Foreign Agent（FA）を気付アドレスとして用いる．FA はモバイル IP 環境においてユーザ端末が移動した際の移動先のフェムトセルネットワークに接続する機能で，Home Agent（HA）からのパケットを受け取り，ユーザ端末へそのパケットを転送する役割を持つ．また，HA はユーザ端末が所属するマクロセルネットワークに設置され，ユーザ端末がマクロセルネットワークからハンドオーバされた場合にユーザ端末へパケットを転送する役割を持つ．従って，ユーザ端末宛てのパケットはいったん HA に送られ，HA と FA の間に構築された IP トンネルを経由して，FA からユーザ端末に配信される．HA と FA の IP トンネルは，安全性・信頼性の面から IPsec を用いて実装することは可能である．

4.4 おわりに

本章では，提案手法を導入するヘテロジニアスネットワークについて概観した後，そのモデル化およびネットワーク構成について述べた．また，ヘテロジニアスネットワーク内をユーザが移動する場合に必要とされる，QoS 情報を共有するためのプロトコル設計について提案した．具体的に，提案手法では，SIP を用いたハンドオーバ手順，およびユーザ端末が移動先のネットワークにおいてネットワーク接続を再構築する手法について述べた．

第 5 章
計算機シミュレーションによる評価

本章では，提案するクロスレイヤ設計を解析するために必要な計算機シミュレータの実装に関して，その詳細を述べる．また，数値例として提案手法の評価結果に基づき，クロスレイヤ設計の有効性，および実装した計算機シミュレータによるクロスレイヤ設計手法の解析に対する実現性を示す．

5.1　はじめに

既存の計算機シミュレーション環境は，一般的なレイヤ独立設計に基づく無線ネットワークシステムを解析することが可能であるが，クロスレイヤ設計に基づく無線ネットワークシステムに関しては解析可能なシミュレーション環境は限られている．例えば，広域ネットワークシミュレータ ns2 は，無線伝送信号処理については完全には解析することができない．また，MATLAB は，無線伝送信号処理を解析することに特化したシミュレータであるため，ネットワーク層・トランスポート層のプロトコルスタックを加味した評価ができない．このような状況に対して，複数のレイヤにまたがるような信号処理を必要とするクロスレイヤ設計システムを解析することを目的として，本章では新たな計算機シミュレーション環境を構築する．

図 5.1 に，新たに実装する計算機シミュレーション環境の構成を示す．本シミュレータでは，将来の無線ネットワーク環境に幅広く対応できるようにするために，新たな無線通信モデルを作成する．その無線通信モデルに基づき，クロスレイヤ設計に基づく適応伝送制御を評価するために，C++ 言語を用いてデータリンク層および物理層のプロトコルスタックを実装する．また，トランスポート層およびネットワーク層のプロトコルスタックは，既存の広域ネットワークシミュレータ ns2 を用いて実装する．

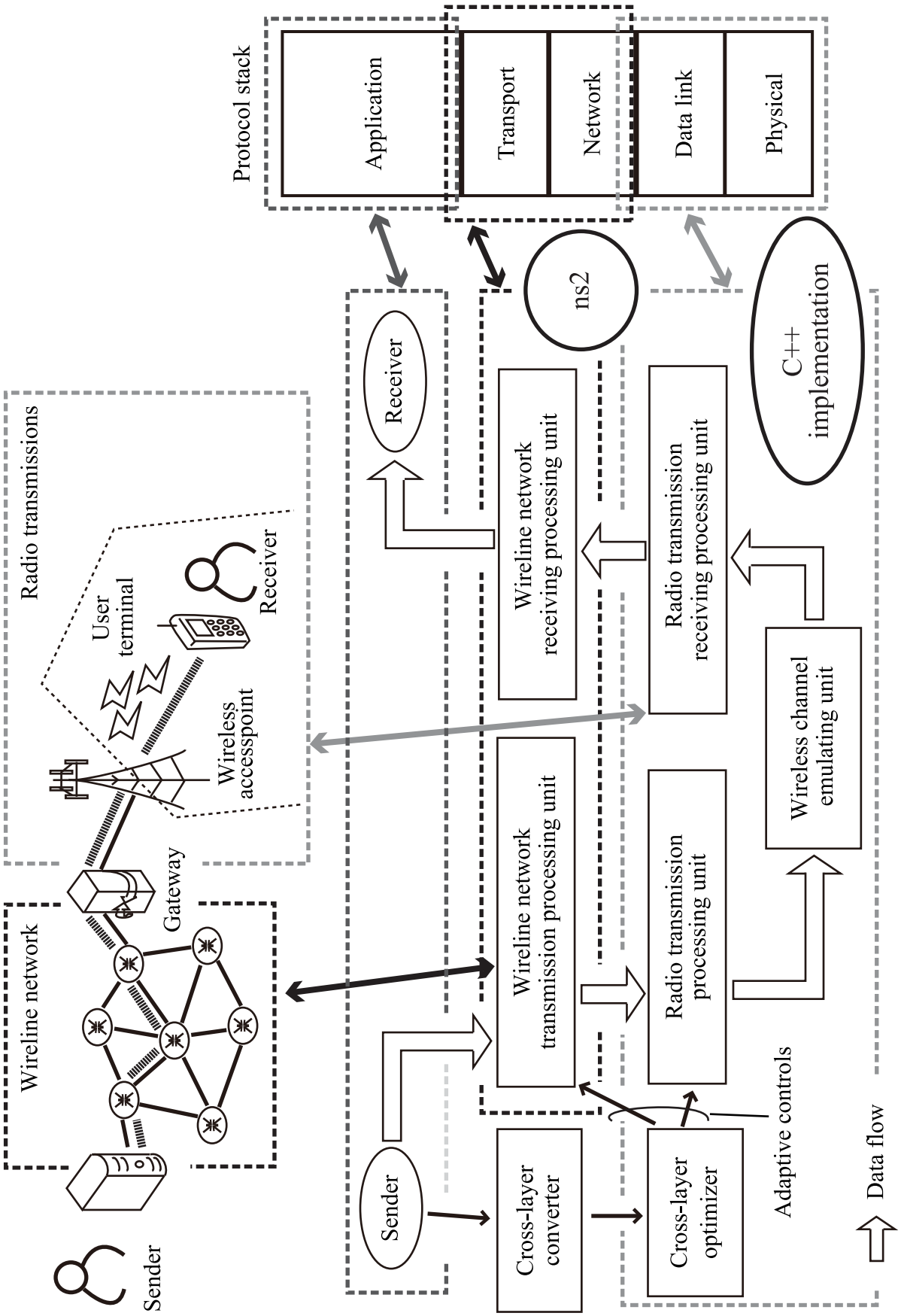


図 5.1　計算機シミュレータの構成

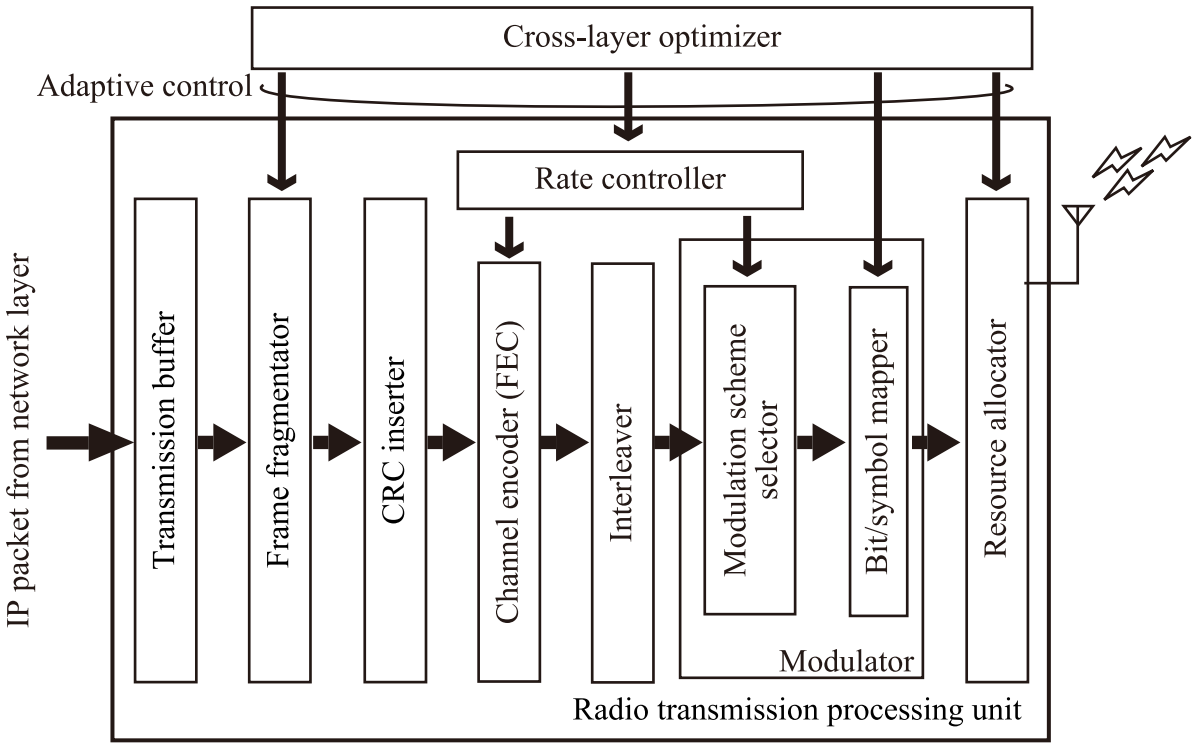


図 5.2　Radio transmission processing unit における信号処理手順

5.2　計算機シミュレータの設計

5.2.1　Radio Transmission Processing Unit

図 5.2 に Radio transmission processing unit における無線伝送信号処理手順を示す．ネットワーク層から受け取った IP パケットは送信バッファに一時的に保存され，無線伝送に必要な信号処理を施す．具体的には，Frame fragmentator を用いてパケットをフレーム分割し，個々の分割フレームには，フレーム誤りを検出するために Cyclic Redundancy Check（CRC）を付加する．また，ビット誤りを訂正するために通信路符号化を行う．また, 無線チャネルにおけるバースト誤りを低減するために，インターリービングを行った後，Modulator にて変調する．

本シミュレータでは，Modulator における Bit/symbol mapper に対してクロスレイヤ適応制御をすることにより，ビット誤り率特性を改善する Dynamic SMD 手法 [72][73] を導入する．Dynamic SMD 手法に関しては，文献 [72][73] において，その基本設計を提案し有効性を検証している．

他方，クロスレイヤ設計に基づく適応パケット長制御および適応レート制御を実現するために，Cross-layer optimizer は Frame fragmentator および Rate controller を適応制御する．具体的には，3 章で述べたクロスレイヤ適応制御手法に基づき，Frame fragmentator は IP パケットをフレーム分割する．ただし，パケットが整数値でフレーム分割できない場合は，ペイロードの余白部分はゼロパディングする．同一のパケットに復元されるフレーム同士を区別するために，分割したフレームのヘッダにはグルーピング情報を付加する．このグルーピング情報はヘッダに理想的に含めることが可能である．また，Rate controller は Channel encoder および Modulation scheme selector を適応制御することにより，AMC に基づく適応レート制御を実現する．本シミュレータでは，変調方式として BPSK，QPSK，16-QAM，通信路符号化方式として畳込み符号（拘束長k = 7）を用いる．そして，変調方式と畳込み符号の符号化率R =1/2，3/4 の組み合わせで表される MCS セットを適応的に切替えることにより適応レート制御を行う．また，畳込み符号における符号化率の調整（レートマッチング）は，パンクチャリングに基づくパンクチャド畳込み符号を用いる．

5.2.2 Radio Transmission Receiving Processing Unit

図 5.3 に Radio transmission receiving processing unit における無線伝送信号処理手順を示す．通信路符号化に対応するための復号には，軟判定ビタビ復号アルゴリズムを用いる [74]．復号に軟判定アルゴリズムを用いる理由は，硬判定アルゴリズムと比較してビット誤り率特性がよく，後述するフレーム再送制御において軟判定情報が必要とされているためである．

フレーム誤りは CRC に基づき判定する．フレームが正しく復元された場合は受信バッファに保存し，分割したすべてのフレームを結合することにより元のパケットを復元する．一方，誤りが検出された場合は送信側に MAC/NACK を返し，送信側は同じフレームを再送する．また，受信信号の復号ビット誤りの影響を低減するために，誤りが検出されたフレームに関しては破棄しないで，その軟判定情報を受信バッファに保存し，再送フレームと合成する．

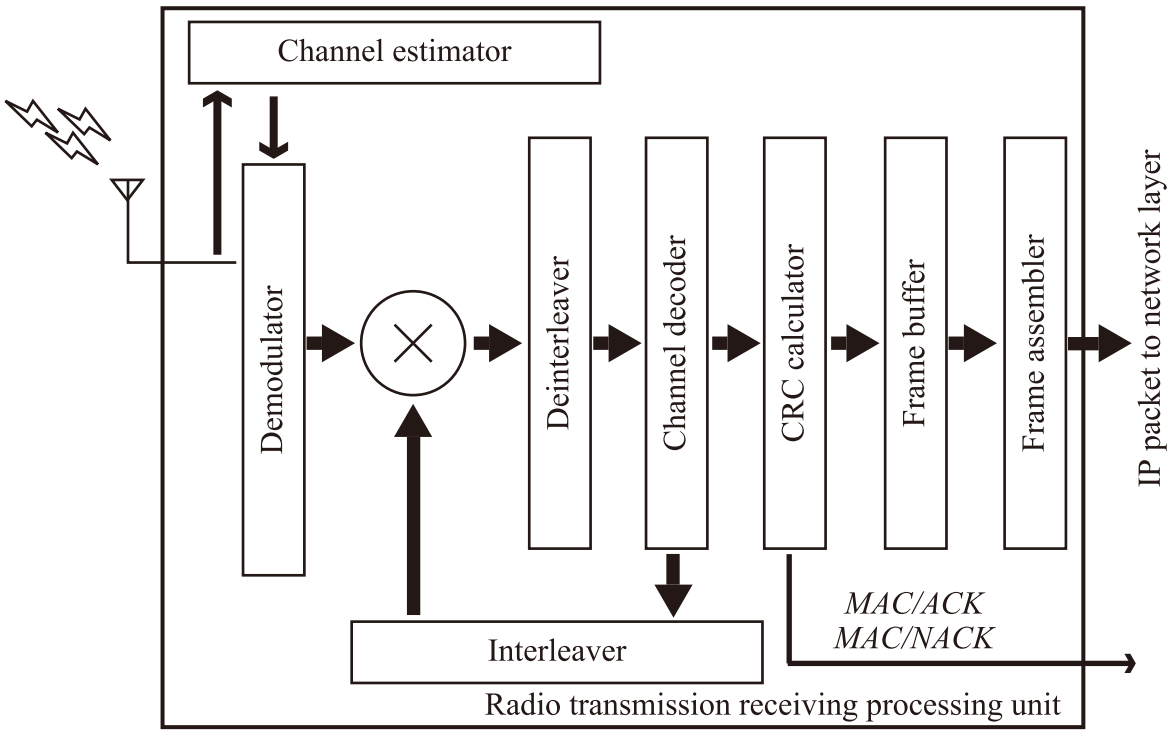


図 5.3　Radio transmission receiving processing unit における信号処理手順

5.2.3　Wireless Channel Emulating Unit

無線チャネルの電波伝搬モデルは，国際的な無線ネットワークシステムの標準化団体 3GPP が提示している標準的なモデルのひとつである Extended Typical Urban（ETU）モデル [75] を実装している．具体的には，9 パスの無線伝搬パスを Jakes モデル [76] に基づき生成して，ETU モデルにおける遅延プロファイルに基づき合成する．また，受信側の復調・検波に必要なチャネル推定に用いる参照信号（パイロット信号）は理想的に利用できる場合を想定して設計している．

5.2.4　ネットワーク設定

実装した計算機シミュレーション環境においては，図 5.1 の上部に示すような無線ネットワーク環境を想定する．このとき，有線伝送区間は，通信リンク帯域を 100 Mbit/s，伝送遅延時間を 5 ms に設定する．また，有線伝送区間に対し DiffServ を導入する．この理由は，有線伝送区間における特性変化の影響を調査するためではなく，提案手法における適応プライオリティ制御を実現するためである．このとき，Cross-layer optimizer は Wireline network transmission processing unit における DiffServ

パラメータをクロスレイヤ適応制御することにより，提案手法における適応プライオリティ制御を可能にする．具体的には，Higher priority に対して DiffServ/Assured Forwarding（AF），Lower priority に対して DiffServ/Best Effort（BE）を割り当てる．ただし，十分なバックログが存在するとき，DiffServ の転送クラスのスケジューリング方式は Round Robin を用いることを想定する．

他方，無線伝送区間は無線チャネル帯域幅が 15 kHz（セルラシステムまたは無線 LAN システムと比較して約 1/100 に相当）の場合を想定して，付録 A に示す C++ 言語で実装した Radio transmission processing unit，Radio transmission receiving processing unit，Wireless channel emulating unit の各プログラムを ns2 に組み込むことにより実現する．

5.3　シミュレーション結果

5.3.1　評価事項

実装した計算機シミュレータを用いて提案手法を解析することにより，クロスレイヤ設計に基づく提案 QoS フレームワークの有効性，計算機シミュレーション環境の実現性に問題がないことを確かめる．とくに，クロスレイヤ設計をシステムに導入する場合，システム特性は改善する一方，システムの複雑化は避けられない．従って，本章の数値例において，クロスレイヤ適応制御を用いることによりネットワーク特性を改善できる点，およびクロスレイヤ設計を導入することによりシステムは複雑化するが，その導入コストを考慮してもシステム特性の改善が期待できる点を示す．

5.3.2　シミュレーション環境

ネットワーク内に流すトラヒックは，ns2 のトラヒックジェネレータを用いて，Constant Bit Rate（CBR）接続によるフローを想定する．また，パケット長は 12,000 bit(ヘッダ長 160 bit)に設定する．ネットワーク特性の評価については，トランスポート層・ネットワーク層のプロトコルスタックは TCP/IP または UDP/IP，データリンク層・物理層は 2 種類のサービス品質が異なるトラヒック（QoS aware または Best Effort）の組み合わせを想定する．

5.3.3　単独のフローに対するネットワーク特性

通信帯域に対して十分にゆとりがある場合において，送信側と受信側で 1 本の TCP または UDP のフローを確立した場合におけるネットワーク特性を評価した．ただし，通信ネットワーク内を流す 1 本のフローは，優先度が高いフロー（QoS aware）を想定する．図 5.4 および 5.5 に，送信情報 1 ビットのエネルギーに対する雑音電力密度比 E_b/N_0 =0–30 dB の範囲において，平均パケット遅延時間および平均スループットを示す．また，従来手法として，クロスレイヤ設計に基づく適応制御を行わない手法を比較対象とした．

図 5.4 において，例えば，E_b/N_0 =20 dB に着目すると，TCP および UDP パケット遅延は 50.0% 改善した．また，E_b/N_0 < 10 dB の範囲では，通信チャネルで混入する雑音に対し十分な受信電力が得られていないため，提案手法および従来手法における TCP/UDP パケット遅延が急激に増大した．一方，図 5.5 において，E_b/N_0 =20 dB に着目すると，TCP および UDP スループットは，各々，30.6%，および 31.3% 改善した．

以上の結果より，提案手法は，ネットワーク特性の改善に対して有効であることが示される．

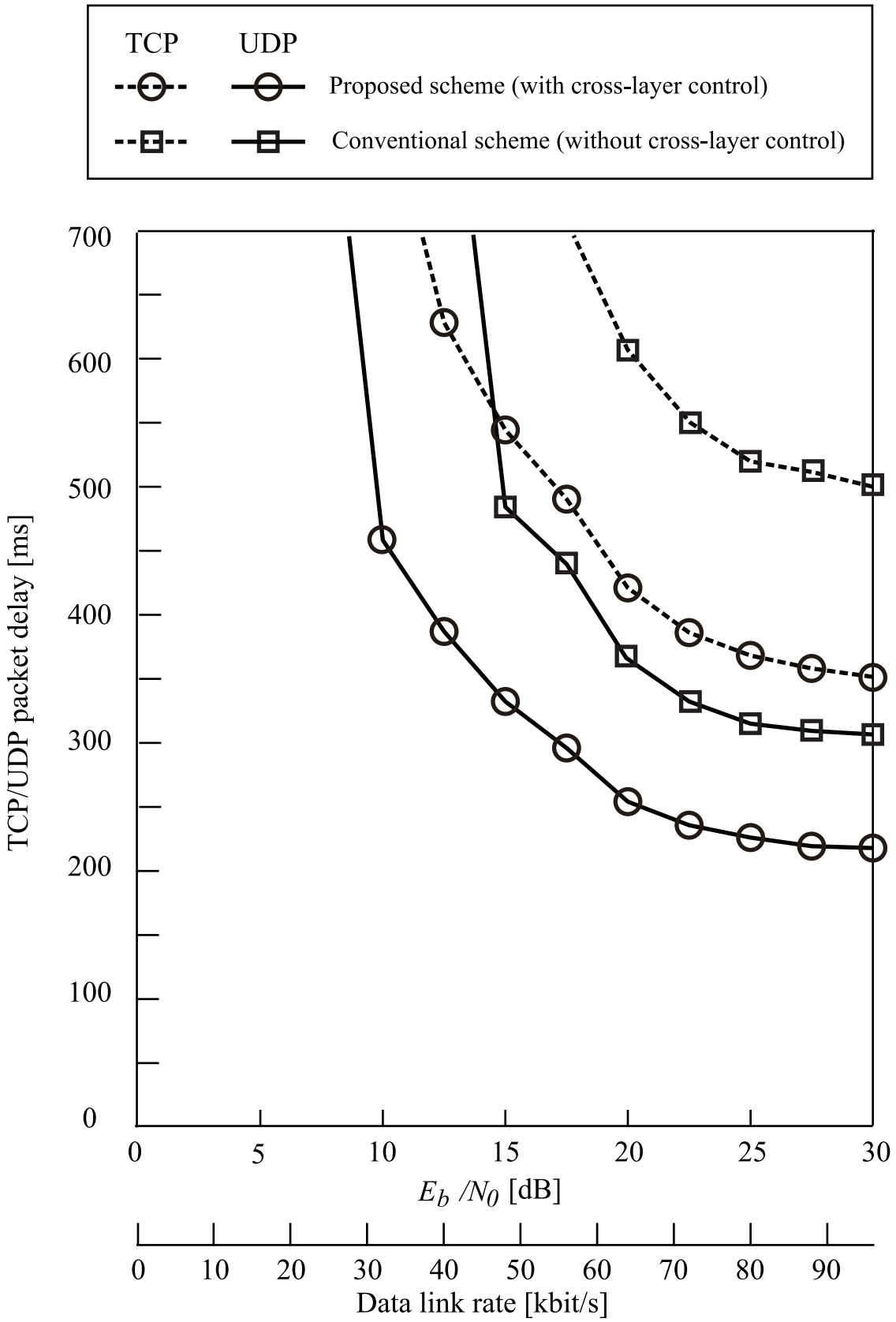


図 5.4　E_b/N_0対平均 TCP/UDP パケット遅延時間

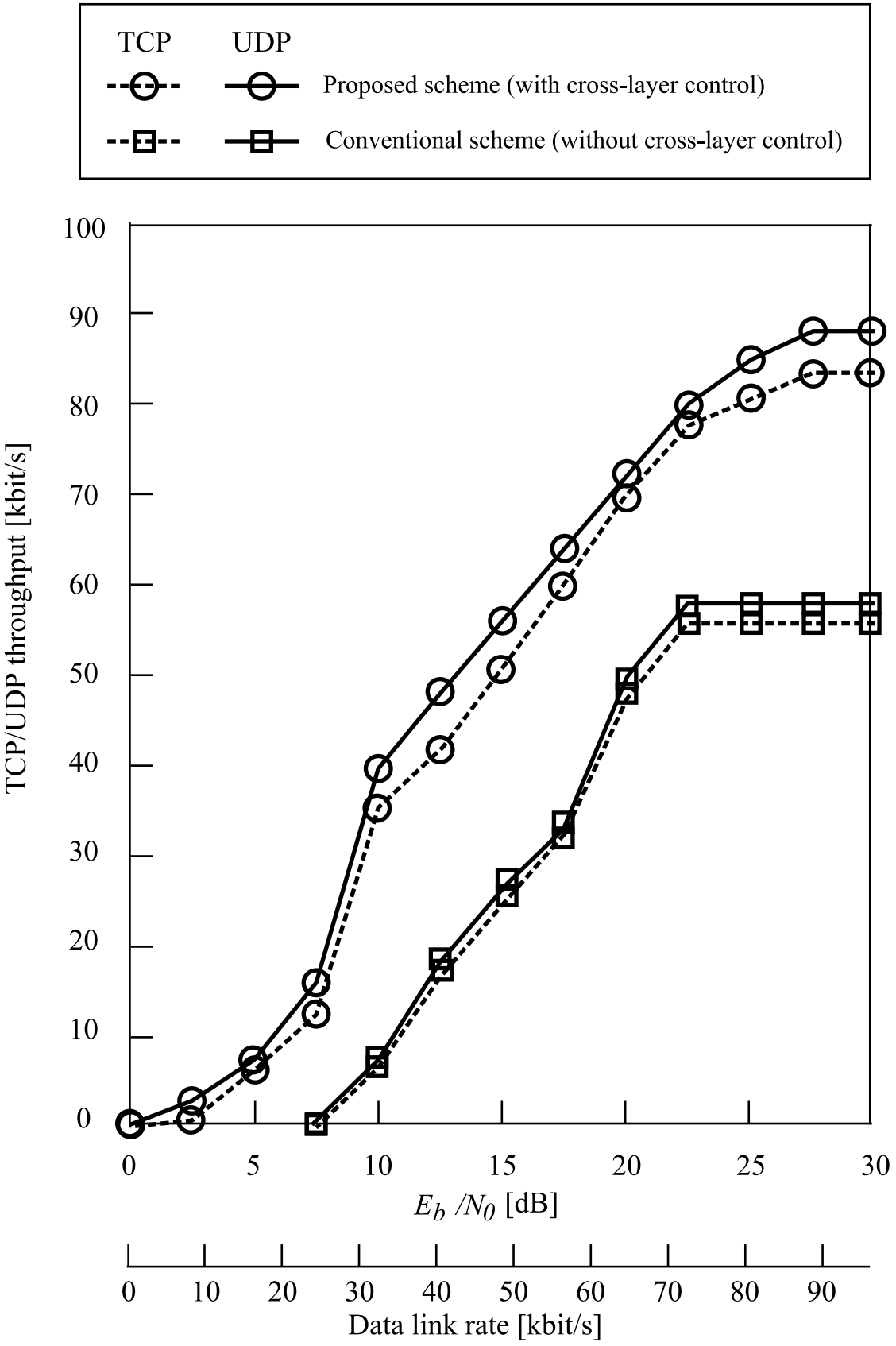


図 5.5　E_b/N_0対平均 TCP/UDP スループット

5.3.4　2 本のフロー混在環境に対するネットワーク特性

本論文では, 将来の無線ネットワークシステムが対象としている, 様々なアプリケーションサービスを単一のネットワークで取り扱う，ヘテロジニアスネットワーク環境を想定している．ヘテロジニアスネットワークでは, 原則として, すべてのネットワークを IP ネットワークで構築するオール IP ネットワークにて実装されることが予測される．一方，近年，無線ネットワークシステムが取り扱うネットワークトラヒックは急増しており，通信ネットワークに高い負荷がかかった状態を想定して，提案手法の有効性を検証することは重要である．このような状況を鑑みて，送信側と受信側で 2 本の TCP または UDP のフローを確立し，それらのフローを単一のネットワーク内で混在させて伝送する場合において，そのネットワーク特性を解析する．ただし，ネットワーク内で混在して流すフローは，優先度が高いフロー（QoS aware）と優先度が低いフロー（Best effort）を各々 1 本ずつ，合計して 2 本とする．

図 5.6 および 5.7 に，無線ネットワークの帯域占有率 ρ = 0.6–1 の範囲において，TCP および UDP に対する平均パケット遅延時間および平均スループットを示す．また，従来手法として，クロスレイヤ制御を行わない手法を比較対象とした．ただし，従来手法は，DiffServ を用いて，クロスレイヤ制御によらない従来のプライマリ制御を適用している．

ρ < 0.6 の範囲では，ネットワークに十分なゆとりがあるため，ネットワーク特性の改善効果がみられなかった．無線ネットワークの帯域占有率 ρ は，通信路容量 C ，ネットワークを流れるトラヒック量 U を用いて式（5.1）で定義する．

$$\rho = U/C \tag{5.1}$$

図 5.6 において，例えば，ρ = 0.8 に着目すると，TCP および UDP パケット遅延は，各々，33.2% および 33.3% 改善した．また，アプリケーションサービスに対する適応無線伝送制御の効果により，ρ に関わらずパケット遅延は一定になった．一方，図 5.7 において，ρ = 0.8 に着目すると，TCP および UDP スループットは，各々，68.7% および 55.2% 改善した．

以上の結果より，提案手法は，2 種類のアプリケーションサービスが混在する状況において，ネットワーク特性の改善に対して有効であることが示される．

5.3.5　クロスレイヤ設計の導入コストとネットワーク特性の改善効果のトレードオフ

クロスレイヤ設計に基づくシステム設計を実現する場合は，その導入コスト（システムの複雑化）とネットワーク性能の改善効果の間にあるトレードオフを考慮する必要がある．提案クロスレイヤ設計手法を導入することによる新たなオーバヘッドに関しては，アプリケーションサービスとクロスレイヤ制御パラメータ間の情報共有に必要となる Cross-layer converter におけるインタラクションが，フロー単位で通信の開始時にのみ生じる．このオーバヘッドとネットワーク特性の改善との間にあるトレードオフに関しては，提案手法を導入する実際の無線ネットワークシステムに依存するため厳密な評価はできていない．しかし，このオーバヘッドは，クロスレイヤ設計を実現するために必要な制御データが増大することに相当する．すなわち，その余計な制御データ量だけネットワークに負荷を与え，結果的に ρ が増大することを意味する．

今回の評価対象となった通信環境においては，図 5.6，5.7 の結果より，遅延時間は $0.6 < \rho < 1$，スループットは $0.75 < \rho < 1$ の範囲で一定値をとり，さらに両者には隔たりがある．一般的には，パケット遅延およびスループットに関して，アプリケーションサービス間のサービス品質の差別化制御（提案手法ではクロスレイヤ適応制御，従来手法ではクロスレイヤ制御を用いない DiffServ 手法）の結果，ρ の値が大きくなる場合であっても通信路容量を超えない限り一定値になる．また，提案方式はボトルネックとなっている無線伝送区間に対して，クロスレイヤ適応制御に基づく無線伝送制御を行っているため，従来方式と比較してネットワーク特性の改善が実現できる．

従って，提案するクロスレイヤ適応制御フレームワークを導入することによるオーバヘッドを加味しても，ネットワーク特性を改善できることが示される．

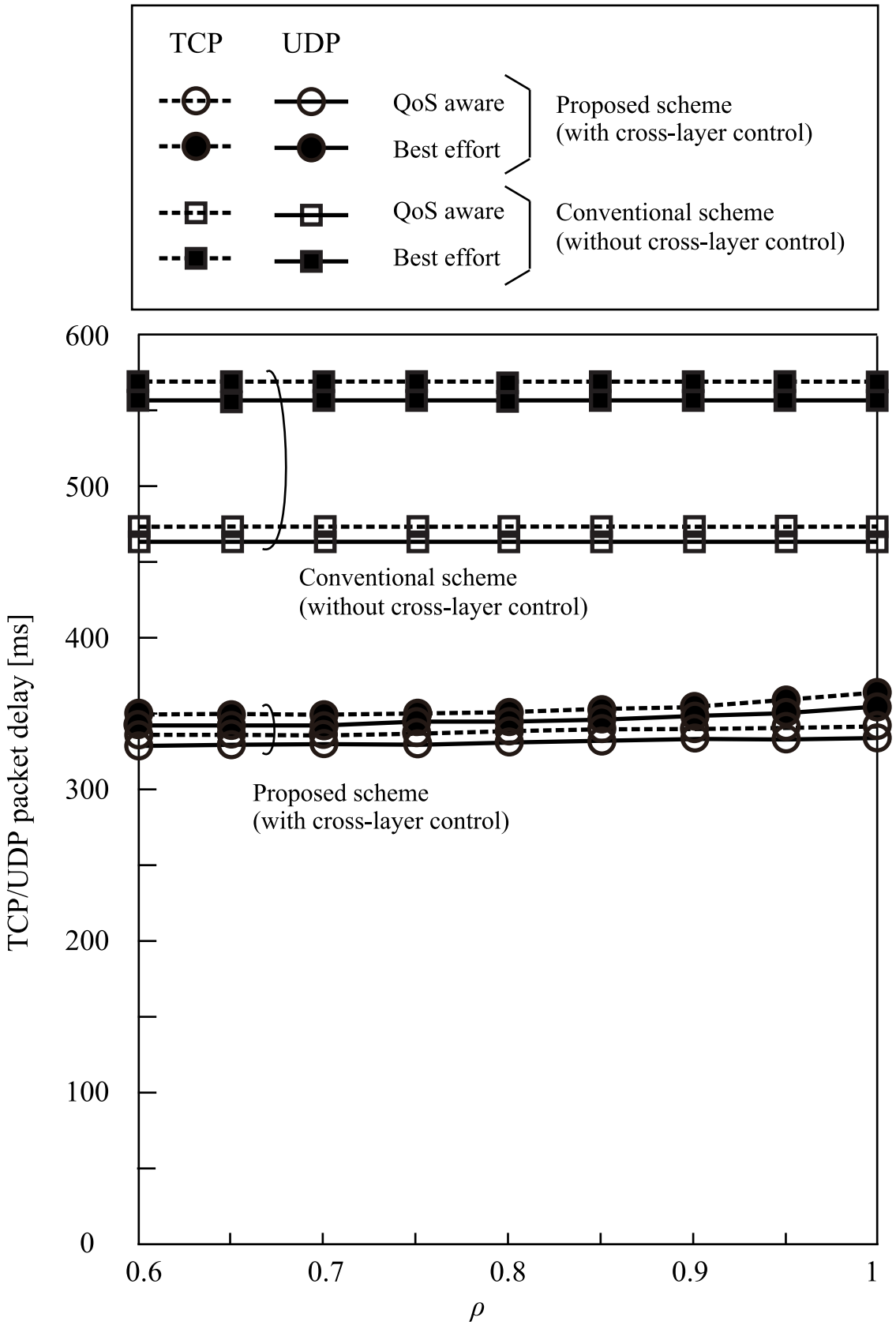


図 5.6　帯域占有率 ρ 対平均 TCP/UDP パケット遅延時間

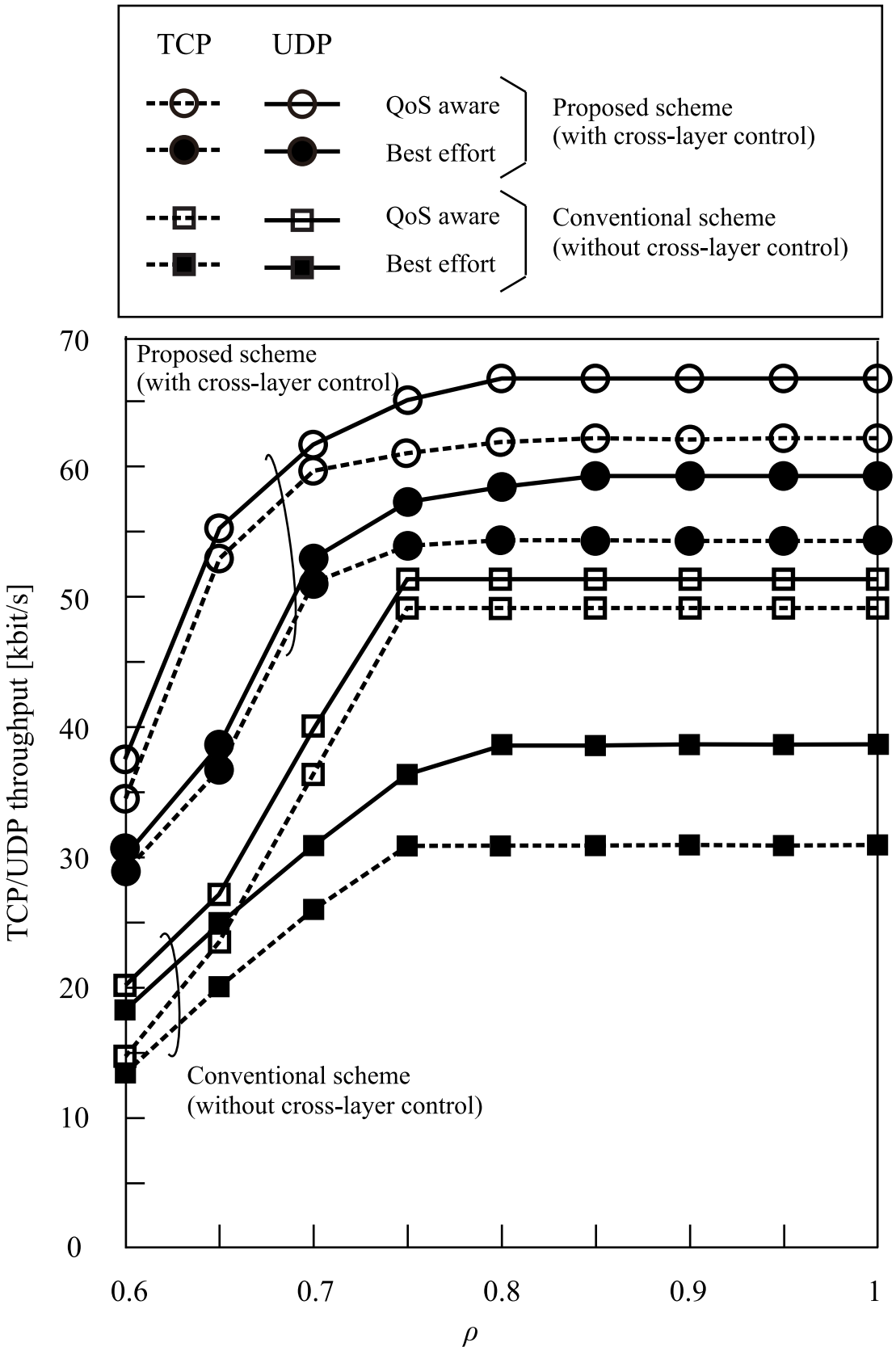


図 5.7　帯域占有率 ρ 対平均 TCP/UDP スループット

5.4　おわりに

本論文では，提案クロスレイヤ設計手法を解析するために，トランスポート層から物理層までのプロトコルスタックに焦点をあてた，新たな計算機シミュレーション環境を実装した．計算機シミュレータを用いた定量的な解析の結果，提案手法に基づくクロスレイヤ適応制御はネットワーク特性の改善に対して有効であることを示した．具体的には，単独のフローを伝送するとき，TCP および UDP に対して，パケット遅延は，ともに，50.0%，スループットは，各々，30.6% および 31.3% 改善した．また，2 本のフローを混在して伝送するとき，TCP および UDP に対して，パケット遅延は，各々，33.2% および 33.3%，スループットは，各々，68.7% および 55.2% 改善した．一方，クロスレイヤ設計を無線ネットワークシステムに導入する際のオーバヘッドを加味しても，ネットワーク特性を改善できることを示した．

第6章
結　言

6.1　本研究のまとめ

無線ネットワークシステムを利用した多彩なモバイルサービスが幅広く普及し，社会全体の通信需要の中で大きな役割を担っている状況において，無線ネットワークシステムは有線ネットワークシステムに並ぶ重要な位置づけがなされている．また，そのモバイルサービスに関しても，大容量なマルチメディアコンテンツを用いた新しいアプリケーションサービスが登場している．今後，ますます無線ネットワークシステムの用途が増大し，様々な分野で利用されることがさらに期待される中，無線ネットワークシステムに対して，ユーザの希望に沿った様々なアプリケーションサービスを提供する必要性が高まっている．

このような状況を鑑み，本研究では，QoS 情報を統一的に取り扱うことを目的とした QoS フレームワークの提案，クロスレイヤ設計に基づく適応制御手法の提案，ヘテロジニアスネットワークに導入するために必要なプロトコル設計，クロスレイヤ設計システムを評価するために必要な計算機シミュレーション環境の構築と有効性を検証した．とくに，今日のほとんどの通信ネットワークシステムで採用されているレイヤ独立設計に基づくプロトコル設計は，無線ネットワークシステムに対して柔軟性に欠ける構造上の課題が指摘されていた．これに対し，本研究の主要な研究テーマであるクロスレイヤ設計を導入することにより，プロトコルレイヤにまたがった垂直統合型の適応制御を行うことが可能である．また，その結果，クロスレイヤ設計はアプリケーションサービスに応じた柔軟かつ大域的な最適化の実現を可能にする．

本研究により，無線ネットワークシステムに対するクロスレイヤ設計に基づく QoS フレームワーク手法を体系化し，その有効性を示すことにより，将来の無線ネットワークシステムの研究分野において，クロスレイヤ設計の応用可能性，および有用性を示した．

6.2　本研究の総括

以下に，本論文の総括を各章ごとにまとめる．

第 2 章では，本論文で取り扱う QoS を定義した後，現在の QoS フレームワーク手法を概観した．既存の QoS を実現する手法は，プライオリティ制御を加味したスケジューリング法，通信ネットワーク資源の予約法，ネットワーク輻輳の抑制法のいずれかの手法を用いて設計されている．有線ネットワークシステムにおいては，これらの手法を駆使して QoS が実現されているが，無線ネットワークシステムにおいては，完成された有線ネットワークシステムの QoS 技術を単に導入しているにすぎない．このような状況において，通信システム全体を考慮した QoS フレームワークを実現するために，本研究ではクロスレイヤ設計に基づく手法を提案した．提案手法は，プロトコルレイヤ間の QoS 情報の定義の差違を吸収するために Cross-layer converter を導入している点が従来手法と大きく異なっている．また，その QoS 情報を用いて Cross-layer optimizer が適切な制御パラメータを設定し，無線伝送信号処理を適応的に制御することにより最適化を図っている．

第 3 章では，提案 QoS フレームワークを用いることにより QoS 情報をプロトコルレイヤ間で共有可能な場合において，Cross-layer optimizer がどのような適応制御を行うべきか提案した．とくに，本研究の中でも重要な位置づけにあるクロスレイヤ適応制御に関して，適応パケット長制御および適応レート制御に焦点をあて，無線 LAN システムに対して導入する場合に必要なプロトコル設計を述べた．提案手法では，動的に変化する CSI だけではなく，QoS 情報も加味して適応制御を実現している点が従来手法と大きく異なっている．適応パケット長制御に関しては，フレーム分割に基づき，その分割フレーム全長を切替えることにより実現する．すなわち，フレーム分割に起因するオーバヘッドの定式化，それを最小化する候補フレームを算出，および最適フレームの決定手法を提案した．一方，適応レート制御に関しては，IEEE 802.11a のすべてのデータ伝送モードに対し，それらを切替える SNR のしきい値を示した．

第 4 章では，提案手法をヘテロジニアスネットワークに導入する場合に必要なプロトコル設計を提案した．すなわち，ヘテロジニアスネットワーク環境のモデル化，および提案手法を導入するために必要なネットワークモデルを提案した．また，ユーザがヘテロジニアスネットワーク内を移動する場合において，QoS 情報を共有するために必要な信号伝送処理に関して，SIP を用いたハンドオーバ手順を示した．

第 5 章では，提案手法を解析するために必要な計算機シミュレーション環境を構築した．とくに，クロスレイヤ設計システムを解析できる計算機シミュレータは限られており，提案手法のようなプロトコルレイヤを統合的に取り扱うような手法を既存のシミュレータでは評価することはできない．このような状況において，ネットワークシミュレータ ns2，および C++ 言語を組み合わせて，新たに計算機シミュレータを実装した．また，計算機シミュレータを用いた定量的な解析の結果，提案手法に基づくクロスレイヤ適応制御はネットワーク特性の改善に対して有効であることを示した．具体的には，単独のフローを伝送するとき，TCP および UDP に対して，パケット遅延は，ともに，50.0%，スループットは，各々，30.6% および 31.3% 改善した．また，2 本のフローを混在して伝送するとき，TCP および UDP に対して，パケット遅延は，各々，33.2% および 33.3%，スループットは，各々，68.7% および 55.2% 改善した．一方，クロスレイヤ設計を無線ネットワークシステムに導入する際のオーバヘッドを加味しても，ネットワーク特性を改善できることを示した．

6.3　今後の課題

提案手法におけるクロスレイヤ設計システムの高度化，種々の無線ネットワークシステムに対して提案手法を導入する場合におけるクロスレイヤ設計の応用，およびクロスレイヤ設計の標準化の観点から今後の課題をまとめる．

提案手法におけるクロスレイヤ設計システムの高度化に関して，クロスレイヤ適応制御手法を拡張する必要がある．具体的には，パケット再送制御を効率化するために，ネットワークコーディング技術のひとつである Superposition Modulation（SM）技術を導入するべきである．このとき，SM 技術におけるビット誤り率低下を引き起こす欠点が生じるため，クロスレイヤ適応制御を用いてこの問題を解決し，かつ最適化を行う必要がある．また，リソース割り当て制御を拡張し，本研究で着目したパケッ

ト遅延時間またはスループットのようなデータ伝送効率だけではなく，ユーザ間の平等性などの他の懸案事項の考慮が必要である．その際には，インセンティブ技術を併用することによるネットワークトラヒック誘導手法を導入したクロスレイヤ設計手法が有用である．

様々な無線ネットワークシステムに対して提案手法を導入する場合における提案手法の応用に関して，ヘテロジニアスネットワークだけではなく，無線分散ネットワークに対する考慮が必要である．とくに，生体通信・災害時緊急通信のような利用シーンのセンサネットワークシステムに対して提案手法を導入する場合，データ伝送効率だけではなく，信頼性の面からも評価しなければならない．また，センサネットワークにおける災害時緊急通信，または Device-to-Device（D2D）通信は，一般的な無線ネットワークシステムが想定しているネットワークトラヒックと比較して，何倍ものネットワークトラヒックを取り扱う必要がある．そのため，高効率なデータ伝送を実現可能にするクロスレイヤ適応制御を考慮した提案手法は有用である．

クロスレイヤ設計の標準化に関して，無線ネットワークシステムは，世界中の通信システムで共通のプロトコルで利用できるようにするために規格化されている．このことは，提案手法におけるクロスレイヤ設計に基づく QoS フレームワークに対しても例外ではない．すなわち，提案手法を既存の通信ネットワークシステムに導入する場合におけるプロトコル上の障壁を取り除かなければならない．とくに，既存の通信ネットワークシステムとの後方互換性を担保することは重要な課題である．また，無線ネットワークシステムにおけるエンドツーエンドでの提案手法の導入を実現する必要がある．すなわち，マルチホップ無線ネットワーク環境を想定し，後方互換性を保証した上で，エンドツーエンドに対するクロスレイヤ設計フレームワークへの拡張が必要である．さらに，その拡張した新たなクロスレイヤ設計フレームワークに対し，ITU，3GPP，IETF などの国際標準化機関において認証されなければならない．

謝　辞

本論文を執筆するにあたり，たいへん多くの方にご協力をいただきました．ここに深く感謝いたします．

本論文をまとめるにあたり，終始あたたかい激励とご鞭撻をいただいた香川大学工学部　生越重章教授に心より感謝申し上げます．生越教授には大学・大学院に在籍中から現在に至るまで，研究活動全般にわたり格別なるご指導とご高配を賜りました．私が博士論文をまとめることができたのは，常に研究計画・推進にあたり懇切なるご指導，ご助言をいただいたからに他なりません．また，学位の修得をあきらめかけようとしたとき，自信をなくしそうなとき，博士課程修了後も研究者として歩むことを選択をしたときなど，研究者の先輩としてだけではなく，人生の先輩としても幾度となく助けられ，様々なことを教えていただきました．心より御礼申し上げます．

本研究を遂行するにあたり，たいへんお忙しい中貴重なお時間を割き，論文の内容に関して有益なご助言をいただいた，香川大学工学部　今井慈郎教授，最所圭三教授，石井光治講師に心より感謝申し上げます．今井教授には計算機システムに関する見地より的確なご助言をいただきました．最所教授にはネットワークシステムに関する見地よりご助言をいただきました．先生方のご助言により，本論文の完成度が高まりました．また，石井講師は，本研究を遂行するにあたり，大学･大学院に在籍時代から，論文投稿の際は共著者としてご指導，ご助言頂きました．また，私に情報理論・符号理論のおもしろさを教えていただきました．本当にありがとうございました．

また，香川大学工学部　淺野裕俊助教，香川大学工学部　生越研究室，石井研究室，浅野研究室の皆様には，日々の研究活動を通じて，多くのご指導やご支援をいただきました．関係諸氏に感謝いたします．

本当に多くのあたたかい，ご指導，ご鞭撻に感謝でいっぱいです．研究・学位論文執筆の上でお世話になったすべての方に感謝いたします．ありがとうございました．

最後に，家族に心から感謝します．この学位論文を見ることなく他界した父，私の健康をいつも気遣い見守ってくれた母に深く深く感謝の意を表します．両親からの期待，支援，励ましがあったからこそ，今の自分が在ると確信しています．

本研究の成果が皆々様のご期待に沿うものかどうか甚だ疑問ではありますが，ここに重ねて厚く謝意を表し，謝辞といたします．

平成２６年３月　森 慎太郎

本研究に関する論文および研究発表一覧

主論文

学術論文

(1) S.Mori, K.Ishii and S.Ogose, “Cross-Layer-Design QoS Policy Management Framework for Wireless Heterogeneous Network,” *J. Signal Process.*, vol.17, no.4, pp.123–126, July 2013.

(2) S.Mori, K.Ishii and S.Ogose, “Cross-Layer Design for Delay Sensitive Packet Length Control based on Fragmentation with Rate Adaptation in Wireless LANs,” *J. Signal Process.*, vol.17, no.6. pp.273–281, Nov. 2013.

国際会議

(1) S.Mori, K.Ishii and S.Ogose, “Cross-Layer Design for Throughput Improvement in Wireless Communications,” *Proc. SICE Annual Conf. 2007*, 1C12-2, Sept. 2007.

(2) S.Mori, K.Ishii and S.Ogose. “Decision Criteria of Adaptive Packet Length for Dynamic SMD with Channel State Information,” *Proc. the 6th IEEE/VTS APWCS*, T06-4, Aug. 2009.

(3) S.Mori, K.Ishii and S. Ogose, “Cross-Layer Designed QoS Mapping Frameworks for Wireless Networks,” *Proc. the 8th IEEE/VTS APWCS*, MP6-8, Aug. 2011.

(4) S.Mori, K.Ishii and S.Ogose, “Cross-Layer Designed Adaptive Packet Length Control for Wireless Networks,” *Proc. 22nd Annual IEEE I. Sym. PIMRC, WS/WDN*, pp.2364–2368, Sept. 2011.

本研究に関連する研究発表

国際会議

(1) S.Mori, K.Ishii and S.Ogose, “Channel Estimation Method for Dynamic SMD,” *Proc. 5th IEEE/VTS APWCS*, T09-2, 5-pages, Seoul, Korea, Aug. 2008.

(2) S.Mori, K.Ishii and S.Ogose, “Cross-Layer Designed QoS Policy Management Framework for Wireless Heterogeneous Network,” *Proc. 2013 RISP I. WS/NCSP*, pp.464–467, Kona, HI, US, Mar. 2013.

(3) S.Mori, K.Ishii and S.Ogose, “Cross-Layer Design for Delay Sensitive Packet Length Control in Wireless LANs,” *Proc. 10th IEEE/VTS APWCS*, A2-1, 5-pages, Seoul, Korea Aug. 2013.

(4) S.Mori, K.Ishii and S.Ogose, “A Study on Cross-Layer-Design Packet Retransmission Control Based on Superposition Modulation,” *Proc. 2014 RISP I. WS/NCSP*, Honolulu, HI, US, Feb.–Mar. 2014. [Accepted]

学会口頭発表

(1) 森慎太郎, 石井光治, 生越重章, “次世代無線ネットワークにおけるTCP性能を改善するためのクロスレイヤ設計,” 信学技報, vol.107, no.224, pp.35–39, Sept. 2007.

(2) 森慎太郎, 石井光治, 生越 重章, “Cross-Layer Design for Throughput Improvement in Wireless Communications,” 電気関係学会四国支部連合大会, 17-22, Sept. 2007.

(3) 森慎太郎, 石井光治, 生越重章, “Dynamic SMDのためのチャネル情報推定法,” 信学技報, vol.107, no.483, pp.53–57, Feb. 2008.

(4)　森慎太郎, 石井光治, 生越重章, “無線通信システムにおけるチャネル情報を用いた適応パケット長決定法,” 信学技報, vol.108, no.426, pp.49–53, Feb. 2009.

(5)　森慎太郎, 石井光治, 生越重章, “Dynamic SMDにおける適応パケット長決定法,” 信学総大2009, B-5-73, Mar. 2009.

(6)　森慎太郎, 石井光治, 生越重章, “M4GSimulatorを用いたSMDの伝送品質評価,” 電気関係学会四国支部連合大会, 12-25, Sept. 2009.

(7)　森慎太郎, 石井光治, 生越重章, “畳込み符号を用いたDynamic SMDにおける最適フレーム長,” 信学技報, vol.109, no.441, pp.77–81 , Mar. 2010.

(8)　森慎太郎, 石井光治, 生越重章, “クロスレイヤ設計によるQoSマッピング法,” 信学総大2011, B-15-8, Mar. 2011.

(9)　森慎太郎, 石井光治, 生越重章, “無線通信システムにおけるクロスレイヤ設計によるQoSマッピング法,” 信学技報, vol.111, no.9, pp.13–18, Apr. 2011.

(10)　森慎太郎, 石井光治, 生越重章, “無線通信システムにおけるクロスレイヤ設計に基づく適応パケット長制御法,” 信学技報, vol.111, no.68, pp.77–82, May 2011.

(11)　森慎太郎, 石井光治, 生越重章, “無線通信システムにおけるクロスレイヤ設計に基づくQoS保証法,” 電気関係学会四国支部連合大会, 12-14, Sept. 2011.

(12)　森慎太郎, 石井光治, 生越重章, “DiffServネットワークにおけるクロスレイヤ設計に基づくQoSマッピング法,” 信学技報, vol.111, no.245, pp.19–24, Oct. 2011.

(13)　森慎太郎, 石井光治, 生越重章, “無線クロスレイヤシステムにおけるQoS情報管理法, 信学技報, vol.111, no.469, pp.281–286, Mar. 2012.

(14)　森慎太郎, 生越重章, “災害時における移動基地局の配置法,” 電気関係学会四国支部連合大会, 12-20, Sept. 2012.

参考文献

[1] Cisco Visual Networking Index, Global Mobile Data Traffic Forecast Update, 2010–2015, Feb. 2011.

[2] M.Dohler, R.Heath, A.Lozano, C.Papadias and R.Valenzuela, "Is the PHY Layer Dead?," *IEEE Commun. Mag.*, vol.49, no.4, pp.159–165, Apr. 2011.

[3] ITU: http://www.itu.int/ .

[4] ITU-R, International Mobile Telecommunications-2000 (IMT-2000), Recommendation ITU-R M.687-2, Feb. 1997.

[5] ITU-R, Detailed Specifications of the Radio Interfaces of IMT-2000, Recommendation ITU-R M1457-9, May 2010.

[6] 3GPP: http://www.3gpp.org/ .

[7] E.Dahlman, S.Parkvall, J.Skold and P.Beming: *3G Evolution, HSPA and LTE for Mobile Broadband*, Academic Press, USA, July 2007.

[8] ITU-R, Document IMT-ADV/1, "Background on IMT-Advanced," Mar. 2008.

[9] 3GPP, RP-090137, "Proposed SID on LTE-Advanced," Mar. 2008.

[10] 3GPP, TS36.814(v9.0.0), "Further Advancements for E-UTRA Physical Layer Aspects," Mar. 2010.

[11] IEEE 802.16 Working Group on Bloadband Wireless Access Standards: http://www.ieee802.org/16/ .

[12] ITU-R, Working Document Toward A Preliminary Draft New Recommendation ITU-R M.[IMT.RESPEC], ITU-R WP5D, Dec. 2011.

[13] E.Dahlman, S.Parkvall and J.Skold: *4G: LTE/LTE-Advanced for Mobile Broadband*, Academic Press, USA, May 2011.

[14] 3GPP, RP-121418, "Proposed SI: Scenarios and Requirements of LTE Small Cell Enhancements," Sept. 2012.

[15] A.Meddeb, "Internet QoS: Pieces of the Puzzle," *IEEE Commun. Mag.*, vol.48, no.1, pp.86–94, Jan. 2010.

[16] R.Stankiewicz, P.Cholada and A.Jajszczyk, "QoX: What is It Really?," *IEEE Commun. Mag.*, vol.49, no.4, pp.148–158, Apr. 2011.

[17] V.Kawadia and P.R.Kumar, "A Cuationary Perspective on Cross-Layer Design," *IEEE Wireless Commun.*, vol.12, no.1, pp.3–11, Feb. 2005.

[18] V.Srivastava and M.Motani, "Cross-Layer Design: A Survey and Road Ahead,"

IEEE Commun. Mag., vol.43, no.12, pp.112–119, Dec. 2005.

[19] R.Ferrus, L.Alonso, A.Umbert, X.Reves, J.Rerez-Romero and F.Casadevall, "Cross-Layer Scheduling Strategy for UMTS Downlink Enhancement," *IEEE Commun. Mag.*, vol.43, no.6, pp.24–28, June 2005.

[20] H.Jiang and W.Zhuang, "Cross-Layer Resource Allocation for Integrated Voice/ Data Traffic in Wireless Cellular Networks," *IEEE Trans. Wireless Commun.*, vol.5, no.2, pp.457–468, Feb. 2006.

[21] M.Assaad and D.Zeghlache, "Cross-Layer Design in HSDPA System to Reduce the TCP Effect," *IEEE J. Sel. Areas in Commun.*, vol.23, no.3, pp.614–625, Mar. 2006.

[22] P.Zhang and S.Jordan, "Cross-Layer Dynamic Resource Allocation with Targeted Throughput for WCDMA Data," *IEEE Trans. Wireless Commun.*, vol.7, no.12, pp.4896–4906, Dec. 2008.

[23] H.Luo, S.Ci, D.Wu, J.Wu and H.Tang, "Quality-Driven Cross-Layer Optimized Video Delivery over LTE," *IEEE Commun. Mag.*, vol.48, no.2, pp.102–109, Feb. 2010.

[24] J.Hwang and S.Kim, "A Cross-Layer Optimization of IEEE 802.11 MAC for Wireless Multihop Networks," *IEEE Commun. Letters*, vol.10, no.7, pp.531–533, July 2006.

[25] Y.Cheng, X.Ling, W.Song, L.Cai, W.Zhuang and X.Shen, "A Cross-Layer Approach for WLAN Voice Capacity Planning," *IEEE J. Sel. Areas in Commun.*, vol.25, no.4, pp.678–688, May 2007.

[26] J.Alonso-Zarate, C.Verikoukis, E.Kartsakli, A.Cateura and L.Alonso, "A Near-Optimum Cross-Layered Distributed Queuing Protocol for Wireless LAN," *IEEE Wireless Commun.*, vol.15, no.1, pp.48–55. Feb. 2008.

[27] J.Wang, M.Venkatachalam and Y.Fang, "System Architecture and Cross-Layer Optimization of Video Broadcast over WiMAX," *IEEE J. Sel. Areas in Commun.*, vol.25, no.4, pp.712–721, May 2007.

[28] J.She, X.Yu, P.Ho and E.Yang, "A Cross-Layer Design Framework for Robust IPTV Services Over IEEE 802.16 Networks," *IEEE J. Sel. Areas in Commun.*, vol.27, no.2, pp.235–245, Feb. 2009.

[29] N.Celandroni, F.Davoli, E.Ferro and A.Gotta, "Long-Lived TCP Connections via Satellite: Cross-Layer Bandwidth Allocation, Pricing, and Adaptive Control," *IEEE/ACM Trans. Networking*, vol.14, no.5, pp.1019–1030, Oct. 2006.

[30] H.Du, L.Fan, U.Mudugamuwa and B.G.Evans, "A Cross-Layer Packet Scheduling Scheme for Multimedia Broadcasting via Satellite Digital Multimedia Broadcasting System," *IEEE Commun. Mag.*, vol.45, no.8, pp.94–101, Aug. 2007.

[31] M.Castro and D.Fernandez, “VoIP Cross-Layer Load Control for Hybrid Satellite-WiMAX Networks,” *IEEE Wireless Commun.*, vol.15, no.3, pp.32–39, June 2008.

[32] K.Yang and X.Wang, “Cross-Layer Network Planning for Multi-Radio Multi-Channel Cognitive Wireless Networks,” *IEEE Trans. Commun.*, vol.56, no.10, pp.1705–1714, Oct. 2008.

[33] A.Ghosh and W.Hamouda, “Cross-Layer Antenna Selection and Channel Allocation for MIMO Cognitive Radios,” *IEEE Trans. Wireless Commun.*, vol.10, no.11, pp.3666–3674, Nov. 2011.

[34] E.Setton, T.Yoo, X.Zhu, A.Goldsmith and B.Girod, “Cross-Layer Design of Ad-Hoc Networks for Real-Time Video Streaming,” *IEEE Wireless Commun.*, vol.12, no.4, pp.59–65, Aug. 2005.

[35] A.Rad and V.Wong, “Cross-Layer Fair Bandwidth Sharing for Multi-Channel Wireless Mesh Networks,” *IEEE Trans. Wireless Commun.*, vol.7, no.9, pp.3436–3445, Sept. 2008.

[36] S.Paris, C.Nita-Rotaru, F.Martignon and A.Capone, “Cross-Layer Metrics for Reliable Routing in Wireless Mesh Networks,” *IEEE/ACM Trans. Networking*, vol.21, no.3, pp.1003–1016, June 2013.

[37] T.Melodia and I.Akyildiz, “Cross-Layer QoS-Aware Communications for Ultra Wide Band Wireless Multimedia Sensor Networks,” *IEEE J. Sel. Areas in Commun.*, vol.28, no.5, pp.653–663, June 2010.

[38] H.Wang, D.Peng, W.Wang, H.Sharif and H.Chen, “Cross-Layer Routing Optimization in Multirate Wireless Sensor Networks for Distributed Source Coding based Applications,” *IEEE Trans. Wireless Commun.*, vol.7, no.10, pp.3999–4009, Oct. 2008.

[39] H.Wang, Y.Yang, M.Ma, H.He and X.Wang, “Network Lifetime Maximization with Cross-Layer Design in Wireless Sensor Networks,” *IEEE Trans. Wireless Commun.*, vol.7, no.10, pp.3579–3768, Oct. 2008.

[40] M.D.Francesco, G.Anastasi, M.Conti, S.K.Das and V.Neri, “Reliability and Energy-Efficiency in IEEE 802.15.4/ZigBee Sensor Networks: An Adaptive and Cross-Layer Approach,” *IEEE J. Sel. Areas in Commun.*, vol.29, no.8, pp.1508–1524, Sept. 2011.

[41] B.Jarupan and E.Ekici, “Location- and Delay-Aware Cross-Layer Communication in V2I Multihop Vehicular Networks,” *IEEE Commun. Mag.*, vol.47, no.11, pp.112–118, Nov. 2009.

[42] Z.Ding and K.Leung, “Cross-Layer Routing Using Cooperative Transmission in Vehicular Ad-Hoc Networks,” *IEEE J. Sel. Areas in Commun.*, vol.29, no.3,

pp.571–581, Mar. 2011.

[43] H.Shan, H.T.Cheng and W.Zhuang, "Cross-Layer Cooperative MAC Protocol in Distributed Wireless Networks," *IEEE Trans. Wireless Commun.*, vol.10, no.8, pp.2603–2615, Aug. 2011.

[44] D.Pompili and F.Akyildiz, "A Multimedia Cross-Layer Protocol for Underwater Acoustic Sensor Networks," *IEEE Trans. Wireless Commun.*, vol.9, no.9, pp.2924–2933, Sept. 2010.

[45] S.Singth, F.Ziliotto, U.Madhow, E.M.Belding and M.Rodwell, "Blockage and Directive in 60 GHz Wireless Personal Area Networks: From Cross-Layer Model to Multihop MAC Design," *IEEE J. Sel. Areas in Commun.*, vol.28, no.8, pp.1400–1413, Oct. 2009.

[46] Y.Pointurier, M.Brandt-Pearce, S.Subramaniam and B.Xu, "Cross-Layer Adaptive Routing and Wavelength Assignment in All-Optical Networks," *IEEE J. Sel. Areas in Commun.*, vol.26, no.6, pp.32–44, Aug. 2008.

[47] K.Lee, E.Modiano and H.Lee, "Cross-Layer Survivability in WDM-Based Networks," *IEEE/ACM Trans. Networking*, vol.19, no.4, pp.1000–1013, Aug. 2011.

[48] F.Z.Yousaf, M.Liebsch, A.Maeder and S.Schmid, "Mobile CDN Enhancements for QoE-Implement Content Delivery in Mobile Operator Networks," *IEEE Network*, vol.27, no.2, pp.14–21, Apr. 2013.

[49] K.Johnsson and D.Cox, "An Adaptive Cross-layer Scheduler for Improved QoS Support of Multiclass Data Services on Wireless Systems," *IEEE J. Sel. Areas in Commun.*, vol.23, no.2, pp.334–343, Feb. 2005.

[50] X.Lin and N.B.Shroff, "The Impact of Inperfect Scheduling on Cross-Layer Congestion Control in Wireless Networks," *IEEE/ACM Trans. Networking*, vol.14, no.2, pp.302–315, Apr. 2006.

[51] A.Zhou, M.Liu, Z.Li and E.Dutkiewicz, "Cross-Layer Design for Proportional Delay Differentiation and Network Utility Maximization in Multi-Hop Wireless Networks," *IEEE Trans. Wireless Commun.*, vol.11, no.4, pp.1446–1455, Apr. 2012.

[52] C.Lin, Y.Liu and M.Tao, "Cross-Layer Optimization of Two-Way Relaying for Statistical QoS Guarantees," *IEEE. J. Sel. Areas in Commun.*, vol.31, no.8, pp.1583–1596, Aug. 2013.

[53] D.Hui, V.Lau and W.Lam, "Cross-Layer Design for OFDMA Wireless Systems with Heterogeneous Delay Requirements," *IEEE Trans. Wireless Commun.*, vol.6, no.8, pp.2872–2880, Aug. 2007.

[54] J.Tang and X.Zhang, “Cross-Layer-Model Based Adaptive Resource Allocation for Statistical QoS Guarantees in Mobile Wireless Networks,” *IEEE Trans. Wireless Commun.*, vol.7, no.6, pp.2318–2328, June 2008.

[55] H.Lin, T.Wu and C.Huang, “Cross Layer Adaptation with QoS Guarantees for Wireless Scalable Video Streaming,” *IEEE Commun. Letters*, vol.16, no.9, pp.1349–1352, Sept. 2012.

[56] A.Ksentini, M.Naimi and A.Gueroui, “Toward an Improvement of H.264 Video Transmission over IEEE 802.11e through a Cross-Layer Architecture,” *IEEE Commun. Mag.*, vol.44, no.1, pp.107–114, Jan. 2006.

[57] B.Stroustrup: *The C++ Programming Language*, Addison-Wesley, USA, June 1997.

[58] NS2: http://www.isi.edu/nsnam/ns/ .

[59] ITU-T, Recommendation P.10/G100

[60] DiffServ, Differentiated Services,*IETF RFC2475*, Dec. 1998.

[61] A.Ravichandran, M.Tacca, M.Welzl and A.Fumagalli, “LN-MAC: A Cross-Layer Explicit Loss Notification Solucation for TCP over IEEE 802.11,” *IEEE GlobeCom 2008*, pp.1–5, New Orleans, LA, USA, Nov.–Dec. 2008.

[62] H.Byun and J.Lim, “On A Fair Congestion Control Scheme for TCP Vegas,” *IEEE Commun. Letters*, vol.9, no.2, pp.190–192, Feb. 2005.

[63] J.D.Day and H.Zimmermann, “The OSI Reference Model,” *Proceedings of the IEEE*, vol.71, no.12, pp.1334–1340, Dec. 1983.

[64] S.Lin, D.Costello and M.Miller, “Automatic Repeat Request Error Control Schemes,” *IEEE Commun. Mag.*, vol.22, no.12, pp.5–17, Dec. 1984.

[65] S.Sampei, S.Komaki and N.Morinaga, “Adaptive Modulation/TDMA Scheme for Large Capacity Personal Multi-media Communication Systems,” *IEICE Trans. Commun.*, vol.E77-B, no.9, pp.1096–1103, Sept. 1994.

[66] C.Fraleigh, S.Moon, B.Lyles, C.Cotton, M.Khan, D.Moll, R.Rockell, T.Seely and C.Diot, “Packet-level Traffic Measurements from the Sprint IP Backbone, ” *IEEE Network*, vol.17, no.6, pp.6–16, Nov.–Dec. 2003.

[67] The Cooperative Association for Internet Data Analysis, “Packet Size Distribution Comparison between Internet Links in 1998 and 2008,” Dec. 2008.

[68] MATLAB: http://www.mathworks.com/ .

[69] J.G.Proakis:*Digital Communications*, McGraw-Hill, U.S.A., Dec. 2000.

[70] J.Hagenauer, “Rate-Compatible Punctured Convolutional Codes (RCPC Codes) and their Applications,” *IEEE Trans. Commun.*, vol.36, no.4, pp.389–400, Apr. 1988.

[71] S.Choudhury and J.D.Gibson, "Payload Length and Rate Adaptation for Multimedia Communications in Wireless LANs," *IEEE J. Sel. Areas in Commun.*, vol.25, no.4, pp.796–807, May 2007.

[72] S.Mori, K.Ishii and S.Ogose, "Channel Estimation Method for Dynamic SMD," *IEEE APWCS 2008*, T09-2, Aug. 2008.

[73] S.Mori, K.Ishii and S.Ogose, "Decision Criteria of Adaptive Packet Length for Dynamic SMD with Channel State Information," *IEEE APWCS 2009*, T06-4, Aug. 2009.

[74] S.Lin, and D.J.Costello: *Error Control Coding*, pp.453–604, Pearson Education, USA, May 2004.

[75] 3GPP, *TS36.104*, V8.2.0, Annex B, Sept. 2008.

[76] W.C.Jakes, *Microwave Mobile Communications*, IEEE Press, May 1994.

付録 A

プログラムのソースコード

A.1　基本信号処理

A.1.1 I/Q 信号

```
0001:  //  iq.hpp:
0002:  //    Representation of I/Q signal constellation and its vector/matrix.
0003:  ////////////////////////////////////////////////////////////////////////
0004:
0005:  #pragma once
0006:  #include <complex>
0007:  #include <vector>
0008:
0009:  namespace iq {
0010:    //  Definitions of I/Q signal constellation template class.
0011:    class CConstellation
0012:      : public std::complex < float >
0013:    {
0014:    public:
0015:      CConstellation ()
0016:        : std::complex < float > ( 0.0f, 0.0f ) { }
0017:      CConstellation ( float real, float imag )
0018:        : std::complex < float > ( real, imag ) { }
0019:      explicit CConstellation ( float real )
0020:        : std::complex < float > ( real, 0.0f ) { }
0021:      CConstellation ( const CConstellation& that )
0022:        : std::complex < float > ( that.real(), that.imag() ) { }
0023:      virtual ~CConstellation () { }
0024:      CConstellation& operator= ( const CConstellation& that )
0025:      {
0026:        real ( that.real() );
0027:        imag ( that.imag() );
0028:        return *this;
0029:      }
0030:      CConstellation& operator^= ( float rhs )
0031:      {
0032:        real ( rhs * abs(*this) * cos ( arg(*this) ) );
0033:        imag ( rhs * abs(*this) * sin ( arg(*this) ) );
0034:        return *this;
0035:      }
0036:      CConstellation& operator^= ( const CConstellation& rhs )
0037:      {
0038:        float r = abs(*this) * abs(rhs);
0039:        float t = arg(*this);
0040:        real ( r*sin(t) );
0041:        imag ( r*cos(t) );
0042:        return *this;
0043:      }
0044:      CConstellation& operator%= ( float rhs )
0045:      {
0046:        real ( rhs / abs(*this) * cos ( arg(*this) ) );
0047:        imag ( rhs / abs(*this) * sin ( arg(*this) ) );
0048:        return *this;
0049:      }
0050:      CConstellation& operator%= ( const CConstellation& rhs )
0051:      {
0052:        float r = abs(*this) / abs(rhs);
0053:        float t = arg(*this);
0054:        real ( r*sin(t) );
0055:        imag ( r*cos(t) );
0056:        return *this;
0057:      }
0058:      CConstellation& operator+= ( const CConstellation& rhs )
0059:      {
0060:        real ( real() + rhs.real() );
0061:        imag ( imag() + rhs.imag() );
0062:        return *this;
0063:      }
0064:      CConstellation& operator-= ( const CConstellation& rhs )
0065:      {
0066:        real ( real() - rhs.real() );
0067:        imag ( imag() - rhs.imag() );
0068:        return *this;
0069:      }
0070:      //  Square Euclidean distance between I/Q signal constellations.
0071:      float distance ( const CConstellation& rhs ) const
0072:      {
0073:        float real2 = ( real() - rhs.real() ) * ( real() - rhs.real() );
0074:        float imag2 = ( imag() - rhs.imag() ) * ( imag() - rhs.imag() );
0075:        return real2 + imag2;
0076:      }
0077:      void clear ()
0078:      {
0079:        real ( 0.0f );
0080:        imag ( 0.0f );
0081:      }
0082:    };
0083:
0084:    //  Multiplication for the envelop of I/Q signal constellations with binary arithmetic operator.
0085:    CConstellation operator+ ( const CConstellation& lhs, const CConstellation& rhs )
0086:    {
0087:      CConstellation ret;
0088:      ret.real ( lhs.real() + rhs.real() );
0089:      ret.imag ( lhs.imag() + rhs.imag() );
0090:      return ret;
0091:    }
0092:    CConstellation operator- ( const CConstellation& lhs, const CConstellation& rhs )
0093:    {
0094:      CConstellation ret;
0095:      ret.real ( lhs.real() - rhs.real() );
0096:      ret.imag ( lhs.imag() - rhs.imag() );
0097:      return ret;
0098:    }
0099:    CConstellation operator* ( const CConstellation& lhs, const CConstellation& rhs )
0100:    {
0101:      CConstellation ret;
0102:      ret.real ( lhs.real() * rhs.real() - lhs.imag() * rhs.imag() );
0103:      ret.imag ( lhs.imag() * rhs.real() + lhs.real() * rhs.imag() );
0104:      return ret;
0105:    }
0106:    CConstellation operator/ ( const CConstellation& lhs, const CConstellation& rhs )
```

```
0107:     {
0108:       CConstellation ret;
0109:       float domi = rhs.real() * rhs.real() + rhs.imag() * rhs.imag();
0110:       ret.real ( (lhs.real()*rhs.real() + lhs.imag()*rhs.imag()) / domi );
0111:       ret.imag ( (rhs.imag()*lhs.real() - lhs.real()*rhs.imag()) / domi );
0112:       return ret;
0113:     }
0114:     CConstellation operator^ ( const CConstellation& lhs, float rhs )
0115:     {
0116:       float real = rhs * abs(lhs) * cos ( arg(lhs) );
0117:       float imag = rhs * abs(lhs) * sin ( arg(lhs) );
0118:       return CConstellation ( real, imag );
0119:     }
0120:     CConstellation operator^ ( const CConstellation& lhs, const CConstellation& rhs )
0121:     {
0122:       float r = abs(lhs) + abs(rhs);
0123:       float t = arg(lhs);
0124:       return CConstellation ( r*sin(t), r*cos(t) );
0125:     }
0126:     CConstellation operator% ( const CConstellation& lhs, const CConstellation& rhs )
0127:     {
0128:       float r = abs(lhs) - abs(rhs);
0129:       float t = arg(lhs);
0130:       return CConstellation ( r*sin(t), r*cos(t) );
0131:     }
0132:
0133:     //  Definitions of the CConstellation sequence.
0134:     template < size_t SIZE >
0135:     class CConstellationSequence
0136:     {
0137:     public:
0138:       static const size_t size = SIZE;
0139:       typedef CConstellation constellation_type;
0140:       typedef std::vector < constellation_type > sequence_type;
0141:       typedef sequence_type::iterator sequence_iterator;
0142:       typedef sequence_type::const_iterator const_sequence_iterator;
0143:     private:
0144:       //  I've implemented the constellation sequence by using container.
0145:       sequence_type sequence_;
0146:     public:
0147:       CConstellationSequence ()
0148:         : sequence_ ( size, CConstellation ( 0.0f, 0.0f ) ) { }
0149:       explicit CConstellationSequence ( const CConstellation& rhs )
0150:       {
0151:         using namespace boost::lambda;
0152:         for_each ( sequence_.begin(), sequence_.end(), _1 = rhs );
0153:       }
0154:       CConstellationSequence ( const CConstellationSequence < SIZE >& that )
0155:       {
0156:         sequence_.assign ( that.sequence_.begin(), that.sequence_.end() );
0157:       }
0158:       virtual ~CConstellationSequence () { }
0159:       CConstellationSequence < SIZE > operator= ( const CConstellationSequence < SIZE >& that )
0160:       {
0161:         sequence_.assign ( that.sequence_.begin(), that.sequence_.end() );
0162:         return *this;
0163:       }
0164:       CConstellationSequence < SIZE > operator= ( const CConstellation& rhs )
0165:       {
0166:         using namespace boost::lambda;
0167:         for_each ( sequence_.begin(), sequence_.end(), _1 = rhs );
0168:         return *this;
0169:       }
0170:
0171:       //  For expression template with two terms operator
0172:       template < typename T > CConstellationSequence < SIZE > operator= ( const T& rhs )
0173:       {
0174:         for ( size_t i = 0; i < SIZE; ++i )
0175:           sequence_[i] = rhs[i];
0176:         return *this;
0177:       }
0178:       CConstellationSequence < SIZE >& operator += ( const CConstellationSequence < SIZE >& rhs )
0179:       {
0180:         for ( size_t i = 0; i < SIZE; ++i )
0181:           sequence_[i] += rhs.sequence_[i];
0182:         return *this;
0183:       }
0184:       CConstellationSequence < SIZE >& operator+= ( const CConstellation& rhs )
0185:       {
0186:         for ( sequence_iterator i = sequence_.begin(); i != sequence_.end(); ++i )
0187:           *i += rhs;
0188:         return *this;
0189:       }
0190:       CConstellationSequence < SIZE >& operator-= ( const CConstellationSequence < SIZE >& rhs )
0191:       {
0192:         for ( size_t i = 0; i < SIZE; ++i )
0193:           sequence_[i] -= rhs.sequence_[i];
0194:         return *this;
0195:       }
0196:       CConstellationSequence < SIZE >& operator-= ( const CConstellation& rhs )
0197:       {
0198:         for ( sequence_iterator i = sequence_.begin(); i != sequence_.end(); ++i )
0199:           *i -= rhs;
0200:         return *this;
0201:       }
0202:       CConstellationSequence < SIZE >& operator *= ( const CConstellationSequence < SIZE >& rhs )
0203:       {
0204:         for ( size_t i = 0; i < SIZE; ++i )
0205:           sequence_[i] *= rhs.sequence_[i];
0206:         return *this;
0207:       }
0208:       CConstellationSequence < SIZE >& operator*= ( const CConstellation& rhs )
0209:       {
0210:         for ( sequence_iterator i = sequence_.begin(); i != sequence_.end(); ++i )
0211:           *i *= rhs;
0212:         return *this;
0213:       }
0214:       CConstellationSequence < SIZE >& operator /= ( const CConstellationSequence < SIZE >& rhs )
0215:       {
0216:         for ( size_t i = 0; i < SIZE; ++i )
0217:           sequence_[i] /= rhs.sequence_[i];
0218:         return *this;
0219:       }
0220:       CConstellationSequence < SIZE >& operator/= ( const CConstellation& rhs )
0221:       {
0222:         for ( sequence_iterator i = sequence_.begin(); i != sequence_.end(); ++i )
0223:           *i /= rhs;
0224:         return *this;
0225:       }
0226:       CConstellationSequence < SIZE >& operator^= ( const CConstellationSequence < SIZE >& rhs )
0227:       {
0228:         for ( size_t i = 0; i < SIZE; ++i )
0229:           sequence_[i] ^= rhs.sequence_[i];
0230:         return *this;
0231:       }
0232:       CConstellationSequence < SIZE >& operator^= ( const CConstellation& rhs )
0233:       {
0234:         for ( sequence_iterator i = sequence_.begin(); i != sequence_.end(); ++i )
0235:           *i ^= rhs;
0236:         return *this;
0237:       }
0238:       CConstellationSequence < SIZE >& operator%= ( const CConstellationSequence < SIZE >& rhs )
0239:       {
0240:         for ( size_t i = 0; i < SIZE; ++i )
0241:           sequence_[i] %= rhs.sequence_[i];
0242:         return *this;
0243:       }
0244:       CConstellationSequence < SIZE >& operator%= ( const CConstellation& rhs )
0245:       {
0246:         for ( sequence_iterator i = sequence_.begin(); i != sequence_.end(); ++i )
0247:           *i %= rhs;
0248:         return *this;
0249:       }
0250:       CConstellation& operator[] ( size_t index )
0251:       {
0252:         return sequence_[index];
```

```
0253:       }
0254:       CConstellation operator[] ( size_t index ) const
0255:       {
0256:         return sequence_[index];
0257:       }
0258:       CConstellation at ( size_t index ) const
0259:       {
0260:         return sequence_[index];
0261:       }
0262:       void clear ()
0263:       {
0264:         for ( sequence_iterator i = sequence_.begin(); i != sequence_.end(); ++i )
0265:           i->clear();
0266:       }
0267:     };
0268:
0269:     //  I've implemented the expression template with two terms operators
0270:     template < typename LeftType, typename OperatorType, typename RightType >
0271:     class constellation_sequence_expression
0272:     {
0273:     private:
0274:       const LeftType& lhs_;
0275:       const RightType& rhs_;
0276:     public:
0277:       constellation_sequence_expression ( const LeftType& lhs, const RightType& rhs )
0278:         : lhs_ ( lhs ), rhs_ ( rhs ) { }
0279:       CConsteIlation operator[] ( size_t index ) const
0280:       {
0281:         return OperatorType::apply ( lhs_[index], rhs_[index] );
0282:       }
0283:     };
0284:     struct plus
0285:     {
0286:       static CConstellation apply ( const CConstellation& lhs, const CConstellation& rhs )
0287:       {
0288:         return lhs + rhs;
0289:       }
0290:     };
0291:     struct minus
0292:     {
0293:       static CConstellation apply ( const CConstellation& lhs, const CConstellation& rhs )
0294:       {
0295:         return lhs - rhs;
0296:       }
0297:     };
0298:     struct multiply
0299:     {
0300:       static CConstellation apply ( const CConstellation& lhs, const CConstellation& rhs )
0301:       {
0302:         return lhs * rhs;
0303:       }
0304:     };
0305:     struct division
0306:     {
0307:       static CConstellation apply ( const CConstellation& lhs, const CConstellation& rhs )
0308:       {
0309:         return lhs / rhs;
0310:       }
0311:     };
0312:     struct envelope
0313:     {
0314:       static CConstellation apply ( const CConstellation& lhs, const CConstellation& rhs )
0315:       {
0316:         return lhs ^ rhs;
0317:       }
0318:     };
0319:
0320:     //  I've implemented the two terms operators
0321:     template < typename LeftType, typename RightType >
0322:     constellation_sequence_expression < LeftType, plus, RightType >
0323:     operator+ ( const LeftType& lhs, const RightType& rhs )
0324:     {
0325:       return constellation_sequence_expression < LeftType, plus, RightType > ( lhs, rhs );
0326:     }
0327:     template < typename LeftType, typename RightType >
0328:     constellation_sequence_expression < LeftType, minus, RightType >
0329:     operator- ( const LeftType& lhs, const RightType& rhs )
0330:     {
0331:       return constellation_sequence_expression < LeftType, minus, RightType > ( lhs, rhs );
0332:     }
0333:     template < typename LeftType, typename RightType >
0334:     constellation_sequence_expression < LeftType, multiply, RightType >
0335:     operator* ( const LeftType& lhs, const RightType& rhs )
0336:     {
0337:       return constellation_sequence_expression < LeftType, multiply, RightType > ( lhs, rhs );
0338:     }
0339:     template < typename LeftType, typename RightType >
0340:     constellation_sequence_expression < LeftType, division, RightType >
0341:     operator/ ( const LeftType& lhs, const RightType& rhs )
0342:     {
0343:       return constellation_sequence_expression < LeftType, division, RightType > ( lhs, rhs );
0344:     }
0345:     template < typename LeftType, typename RightType >
0346:     constellation_sequence_expression < LeftType, division, RightType >
0347:     operator^ ( const LeftType& lhs, const RightType& rhs )
0348:     {
0349:       return constellation_sequence_expression < LeftType, envelope, RightType > ( lhs, rhs );
0350:     }
0351:
0352:   }  //  End of the namespace iq
```

A.1.2 ビット信号

```
0001:   //  bits.hpp:
0002:   //    Representation of bit operation and its vector/matrix.
0003:   ////////////////////////////////////////////////////////////////////////
0004:
0005:   #pragma once
0006:   #include <vector>
0007:
0008:   namespace bits {
0009:     //  Definitions of the bits structure.
0010:     template < size_t LENGTH, typename BitsType = unsigned long >
0011:     struct bits { };
0012:
0013:     template < size_t LENGTH >
0014:     struct bits < LENGTH, unsigned long >
0015:     {
0016:       static const size_t block_size = 32;
0017:       typedef unsigned long bits_type;
0018:       static const size_t length = LENGTH;
0019:       static const size_t size = block_size * length;
0020:       std::vector < bits_type > bits_;
0021:       bits ( )
0022:         : bits_(LENGTH, 0UL) { }
0023:       bits < LENGTH, unsigned long > operator= ( const bits < LENGTH, unsigned long >& that )
0024:       {
0025:         for ( size_t i = 0; i < block_size; ++i )
0026:           bits_[i] = that.bits_[i];
0027:       }
0028:       bits_type operator() ( size_t block, size_t index ) const
0029:       {
0030:         return ( bits_[block] >> ( block_size - index - 1 ) ) & 0x1UL;
0031:       }
0032:       bits_type operator() ( size_t index ) const
0033:       {
0034:         size_t blk = index / block_size;
0035:         size_t idx = index % block_size;
0036:         return ( bits_[blk] >> ( block_size - idx - 1 ) ) & 0x1UL;
0037:       }
0038:       void operator() ( size_t block, size_t index, const bits_type& value )
0039:       {
```

```
0040:       bits_[block] |= ( value << ( block_size - index - 1 ) );
0041:     }
0042:     void operator() ( size_t index, const bits_type& value )
0043:     {
0044:       size_t blk = index / block_size;
0045:       size_t idx = index % block_size;
0046:       bits_[blk] |= ( value << ( block_size - idx - 1 ) );
0047:     }
0048:     void clear ( )
0049:     {
0050:       for ( size_t i = 0; i < LENGTH; ++i )
0051:         bits_[i] = 0U;
0052:     }
0053:   };
0054:
0055:   template < size_t FRAME_SIZE, size_t INFORMATION_FRAME_SIZE, size_t IP_PACKET_SIZE >
0056:   struct frame_types
0057:   {
0058:     typedef bits < FRAME_SIZE > frame_type;
0059:     typedef bits < INFORMATION_FRAME_SIZE > iframe_type;
0060:     static const size_t division_number = IP_PACKET_SIZE / INFORMATION_FRAME_SIZE;
0061:   };
0062:
0063: }  //  End of the namespace bits
```

A.1.3 タイマー信号

```
0001: //  clock.hpp:
0002: //    Definitions of the clock class.
0003: ////////////////////////////////////////////////////////////////////
0004:
0005: #pragma once
0006:
0007: //  Definitions of the clock classm, which is expressed as the time parameters
0008: template < typename ResolutionType = float >
0009: class CClock
0010: {
0011: public:
0012:   typedef ResolutionType resolution_type;
0013: private:
0014:   //  I assume that the resolution value is expressed as micro second.
0015:   ResolutionType resolution_;
0016:   unsigned long low_;
0017:   unsigned long high_;
0018: public:
0019:   CClock()
0020:     : resolution_( static_cast<ResolutionType>(0) ), low_( 0UL ), high_( 0UL ) { }
0021:   CClock ( const CClock& clk )
0022:     : resolution_( clk.resolution_ ), low_( clk.low_ ), high_( clk.high_ ) { }
0023:   explicit CClock ( const ResolutionType& resolution )
0024:     : resolution_( resolution ), low_( 0UL ), high_( 0UL ) { }
0025:   virtual ~CClock() { }
0026:   CClock& operator= ( const CClock& that )
0027:   {
0028:     resolution_ = that.resolution_;
0029:     high_ = that.high_;
0030:     low_ = that.low_;
0031:     return *this;
0032:   }
0033:   CClock& operator++ ()
0034:   {
0035:     if ( low_ == 0xFFFFFFFFUL )
0036:     {
0037:       ++high_;
0038:       low_ = 0x00000000UL;
0039:     }
0040:     else
0041:       ++low_;
0042:     return *this;
0043:   }
0044:   CClock operator++ ( int )
0045:   {
0046:     CClock tmp = *this;
0047:     ++(*this);
0048:     return tmp;
0049:   }
0050:   CClock& operator-- ()
0051:   {
0052:     if ( low_ == 0x00000000UL )
0053:     {
0054:       --high_;
0055:       low_ = 0xFFFFFFFFUL;
0056:     }
0057:     else
0058:       --low_;
0059:     return *this;
0060:   }
0061:   CClock operator-- ( int )
0062:   {
0063:     CClock tmp = *this;
0064:     --(*this);
0065:     return tmp;
0066:   }
0067:   //  This function replays the time [second].
0068:   ResolutionType operator () ()
0069:   {
0070:     return ( ( 0x100000000 * resolution_ ) * high_ + resolution_ * low_ ) * 1e-6f;
0071:   }
0072:   ResolutionType resolution () const
0073:   {
0074:     return resolution_;
0075:   }
0076: };
0077:
0078: //  Typedefs of the clock class.
0079: typedef CClock < float > CLOCK, *LPCLOCK;
```

A.1.4 乱数生成器

```
0001: //  random.hpp:
0002: //    In order to simplify the boost::random, I've implemented the wrapped classes.
0003: ////////////////////////////////////////////////////////////////////
0004:
0005: #pragma once
0006: #include <boost/random.hpp>
0007:
0008: namespace random {
0009:   //  Generator of the random values based on the normal distribution.
0010:   template < typename RealType, typename Engine >
0011:   class Gaussian
0012:   {
0013:   private:
0014:     boost::variate_generator < Engine, boost::normal_distribution < RealType > > generator_;
0015:     template < typename T > struct seed_traits
0016:     {
0017:       typedef typename T::result_type seed_type;
0018:     };
0019:   public:
0020:     typedef boost::variate_generator <Engine,boost::normal_distribution<RealType> > generator_type;
0021:     typedef typename generator_type::engine_value_type engine_value_type;
0022:     typedef typename generator_type::engine_type engine_type;
0023:     typedef typename generator_type::distribution_type distribution_type;
0024:     typedef typename generator_type::result_type result_type;
0025:     typedef typename seed_traits < Engine >::seed_type seed_type;
0026:     explicit Gaussian ( seed_type seed,
0027:       const result_type& mean = result_type(0), const result_type& dst = result_type(1) )
0028:       : generator_( Engine ( seed ), boost::normal_distribution<RealType>( mean, dst ) ) { }
0029:     void seed ( seed_type value )
0030:     {
```

```
0031:         generator_.engine().seed ( value );
0032:       }
0033:       result_type mean ( ) const
0034:       {
0035:         return generator_.distribution().mean ( );
0036:       }
0037:       result_type dst ( ) const
0038:       {
0039:         return generator_.distribution().sigma ( );
0040:       }
0041:       result_type operator ( ) ( )
0042:       {
0043:         return generator_();
0044:       }
0045:       engine_value_type& engine ( )
0046:       {
0047:         return generator_.engine();
0048:       }
0049:       const engine_value_type& engine ( ) const
0050:       {
0051:         return generator_.engine();
0052:       }
0053:       distribution_type& distribution ( )
0054:       {
0055:         return generator_.distribution();
0056:       }
0057:       const distribution_type& distribution ( ) const
0058:       {
0059:         return generator_.distribution();
0060:       }
0061:       generator_type& generator ( )
0062:       {
0063:         return generator_;
0064:       }
0065:       const generator_type& generator ( ) const
0066:       {
0067:         return generator_;
0068:       }
0069:     };
0070:
0071:     //  Typedefs of the Gaussian class.
0072:     typedef Gaussian < float, boost::mt19937 > GAUSSIAN_GENERATOR, *LPGAUSSIAN_GENERATOR;
0073:
0074:     //  Generator of the random values based on the uniform random distribution.
0075:     template < typename RealType, typename Engine >
0076:     class Uniform
0077:     {
0078:     private:
0079:       template < typename T >
0080:       struct seed_traits
0081:       {
0082:         typedef typename T::result_type seed_type;
0083:       };
0084:       boost::variate_generator < Engine, boost::uniform_real<RealType> > generator_;
0085:     public:
0086:       typedef boost::variate_generator < Engine, boost::uniform_real<RealType> > generator_type;
0087:       typedef typename generator_type::engine_value_type engine_value_type;
0088:       typedef typename generator_type::engine_type engine_type;
0089:       typedef typename generator_type::distribution_type distribution_type;
0090:       typedef typename generator_type::result_type result_type;
0091:       typedef typename seed_traits < Engine >::seed_type seed_type;
0092:       explicit Uniform ( seed_type seed,
0093:         const result_type& begin = result_type(0), const result_type& end = result_type(1) )
0094:         : generator_ ( Engine(seed), boost::uniform_real < RealType >(begin, end) ) { }
0095:       void seed ( seed_type value )
0096:       {
0097:         generator_.engine().seed ( value );
0098:       }
0099:       result_type begin ( ) const
0100:       {
0101:         return generator_.distribution().begin();
0102:       }
0103:       result_type end ( ) const
0104:       {
0105:         return generator_.distribution().end();
0106:       }
0107:       result_type operator ( ) ( )
0108:       {
0109:         return generator_();
0110:       }
0111:       engine_value_type& engine ( )
0112:       {
0113:         return generator_.engine()
0114:       };
0115:       const engine_value_type& engine ( ) const
0116:       {
0117:         return generator_.engine();
0118:       }
0119:       distribution_type& distribution ( )
0120:       {
0121:         return generator_.distribution();
0122:       }
0123:       const distribution_type& distribution ( ) const
0124:       {
0125:         return generator_.distribution();
0126:       }
0127:       generator_type& generator ( )
0128:       {
0129:         return generator_;
0130:       }
0131:       const generator_type& generator ( ) const
0132:       {
0133:         return generator_;
0134:       }
0135:     };
0136:
0137:     //  Typedefs of the uniform class.
0138:     typedef Uniform < float, boost::mt19937 > UNIFORM_GENERATOR, *LPUNIFORM_GENERATOR;
0139:     //  In this function, we can obtain the random 170 sequence which is unsigned long type.
0140:     unsigned long gen_random_sequence ( UNIFORM_GENERATOR& ugen )
0141:     {
0142:       unsigned long ret = 0x0UL;
0143:       for ( size_t i = 0; i < 32; ++i, ret <<= 1 )
0144:         ret |= (ugen() > 0.5 ) ? 0x1UL : 0x0UL;
0145:       return ret;
0146:     }
0147:
0148:   }  //  End of the namespace random
```

A.2 通信路符号化信号処理

A.2.1 シフトレジスタ

```
0001:   //  shift_register.hpp:
0002:   //   Definitions of the shift registers class.
0003:   ////////////////////////////////////////////////////////////////////
0004:
0005:   #pragma once
0006:   #include "bits.hpp"
0007:   #include <vector>
0008:   #include <ostream>
0009:   #include <boost/lambda/lambda.hpp>
0010:   #include <boost/numeric/ublas/matrix.hpp>
0011:
0012:   namespace ecc {
0013:     //  Definitions of shift registers class, which is utilized for the convolutional and turbo codes
0014:     class shift_register
0015:     {
```

```
0016:     private:
0017:       unsigned long memory_;
0018:       size_t memory_size_;
0019:     public:
0020:       shift_register ( )
0021:         : memory_(0UL), memory_size_(0) { }
0022:       explicit shift_register ( size_t memory_size )
0023:         : memory_(0UL), memory_size_(memory_size) { }
0024:       shift_register ( unsigned long memory, size_t memory_size )
0025:         : memory_(memory), memory_size_(memory_size) { }
0026:       shift_register ( const shift_register& that )
0027:         : memory_(that.memory_), memory_size_(that.memory_size_) { }
0028:       virtual ~shift_register ( ) { }
0029:       shift_register& operator= ( unsigned long rhs )
0030:       {
0031:         memory_ = rhs;
0032:         return *this;
0033:       }
0034:       shift_register& operator= ( const shift_register& that )
0035:       {
0036:         memory_ = that.memory_;
0037:         memory_size_ = that.memory_size_;
0038:         return *this;
0039:       }
0040:       unsigned long operator() ( )
0041:       {
0042:         return memory_;
0043:       }
0044:       void operator() ( unsigned long& rhs )
0045:       {
0046:         rhs = memory_;
0047:       }
0048:       unsigned long operator>> ( unsigned long input )
0049:       {
0050:         memory_ |= ( input << memory_size_ );
0051:         unsigned long ret = memory_ & 0x1UL;
0052:         memory_ >>= 1;
0053:         return ret;
0054:       }
0055:       unsigned long operator[] ( size_t index ) const
0056:       {
0057:         return ( memory_ >> ( memory_size_ - index - 1 ) ) & 0x1UL;
0058:       }
0059:       unsigned long at ( size_t index ) const
0060:       {
0061:         return ( memory_ >> ( memory_size_ - index - 1 ) ) & 0x1UL;
0062:       }
0063:       void clear ( )
0064:       {
0065:         memory_ = 0UL;
0066:       }
0067:       Void setStatus ( unsigned long stat )
0068:       {
0069:         memory_ = stat;
0070:       }
0071:       unsigned long getStatus ( ) const
0072:       {
0073:         return memory_;
0074:       }
0075:     };
0076:
0077:   }  //  End of the namespace ecc.
```

A.2.2 畳込み符号化

```
0001:   //  convolutional.hpp:
0002:   //    Definitions of convolutional code classes.
0003:   ////////////////////////////////////////////////////////////////////////
0004:
0005:   #pragma once
0006:   #include "shift_register.hpp"
0007:   #include "bits.hpp"
0008:   #include <boost/preprocessor.hpp>
0009:   #include <vector>
0010:
0011:   namespace ecc { namespace convolutional {
0012:     //  Connections of the shift registers in the convolutional codes.
0013:     class convolutional_connection
0014:     {
0015:     private:
0016:       std::vector < int > position_;
0017:     public:
0018:       convolutional_connection () { }
0019:       virtual ~convolutional_connection () { }
0020:       void initialize ( const std::vector < int >& rhs )
0021:       {
0022:         position_.assign ( rhs.begin(), rhs.end() );
0023:       }
0024:       //  Initialize the convolutional_connection class by using octal digit expression.
0025:       void initialize ( unsigned long rhs, size_t length )
0026:       {
0027:         std::vector<int> init;
0028:         for ( size_t i = 0; i < length - 1; ++i )
0029:           if ( ( rhs >> i ) & 0x1UL )
0030:             init.push_back ( i );
0031:         initialize ( init );
0032:       }
0033:       unsigned long operator() ( const shift_register& sr, unsigned long input ) const
0034:       {
0035:         unsigned long ret = input;
0036:         for ( size_t i = 0; i < position_.size(); ++i )
0037:           ret ^= sr[position_[i]];
0038:         return ret;
0039:       }
0040:     };
0041:
0042:     class basic_convolutional_code_encoder
0043:     {
0044:     public:
0045:       typedef bits::bits < 1 > value_type;
0046:     protected:
0047:       shift_register shift_register_;
0048:       explicit basic_convolutional_code_encoder ( size_t memories_size )
0049:         : shift_register_(memories_size) { }
0050:     public:
0051:       virtual ~basic_convolutional_code_encoder() { }
0052:       void clear()
0053:       {
0054:         shift_register_.clear();
0055:       }
0056:
0057:       void
0058:       setStatus ( unsigned long rhs )
0059:       {
0060:         shift_register_.setStatus(rhs);
0061:       }
0062:
0063:       unsigned long
0064:       getStatus ( ) const
0065:       {
0066:         return shift_register_.getStatus();
0067:       }
0068:     };
0069:
0070:     //  Optimum convolutional code encoder with rate R=1/2, 1/3, 1/4 and constrain length K=3-14
0071:     class basic_r1n_convolutional_code_encoder
0072:       : public basic_convolutional_code_encoder
0073:     {
0074:     protected:
0075:       std::vector < convolutional_connection > generator_equations_;
0076:     public:
0077:       basic_r1n_convolutional_code_encoder ( size_t memories_size, size_t generator_equations_number )
```

```
0078:         : generator_equations_(generator_equations_number),
0079:             basic_convolutional_code_encoder ( memories_size ) { }
0080:       virtual ~basic_r1n_convolutional_code_encoder () { }
0081:       template < typename SequenceType1, typename SequenceType2 >
0082:       void encoding ( const SequenceType1& s1, SequenceType2& s2 )
0083:       {
0084:         const size_t eq = generator_equations_.size();
0085:         const size_t r = SequenceType2::length / SequenceType1::length;
0086:         s2.clear();
0087:         for ( size_t i = 0; i < SequenceType1::length; ++i )
0088:           for ( size_t j = 0; j < SequenceType1::block_size; ++j )
0089:           {
0090:             for ( size_t k = 0; k < eq; ++k )
0091:               s2( (j*eq+k)/SequenceType2::block_size+r*i, (j*eq+k)%SequenceType2::block_size,
0092:                 generator_equations_[k] ( shift_register_, s1(i,j) ) );
0093:             shift_register_ >> s1(i,j);
0094:           }
0095:       }
0096:       template < typename SequenceType1, typename SequenceType2 >
0097:       void encoding ( const SequenceType1& s1, SequenceType2& s2, size_t length )
0098:       {
0099:         const size_t eq = generator_equations_.size();
0100:         const size_t r = SequenceType2::length / SequenceType1::length;
0101:         size_t len = 0;
0102:         s2.clear();
0103:         for ( size_t i = 0; i < SequenceType1::length; ++i )
0104:           for ( size_t j = 0; j < SequenceType1::block_size; ++j )
0105:           {
0106:             for ( size_t k = 0; k < eq; ++k )
0107:               s2 ( (j*eq+k)/SequenceType2::block_size+r*i, (j*eq+k)%SequenceType2::block_size,
0108:                   generator_equations_[k] ( shift_register_, s1(i,j) ) );
0109:             if ( len++ >= length )
0110:               return;
0111:             shift_register_ >> s1(i,j);
0112:           }
0113:       }
0114:     };
0115:
0116:     class basic_r14_convolutional_code_encoder
0117:       : public basic_r1n_convolutional_code_encoder
0118:     {
0119:     public:
0120:       static const size_t inverse_rate = 4;
0121:       static const size_t input = 2;
0122:       static const size_t input_bit = 1;
0123:       static const size_t output = 16;
0124:       static const size_t output_bit = 4;
0125:       static const size_t smat2 = output;
0126:       explicit basic_r14_convolutional_code_encoder ( size_t memories_size )
0127:         : basic_r1n_convolutional_code_encoder ( memories_size, 4 ) {}
0128:       virtual ~basic_r14_convolutional_code_encoder ( ) {}
0129:     };
0130:
0131:     class basic_r13_convolutional_code_encoder
0132:       : public basic_r1n_convolutional_code_encoder
0133:     {
0134:     public:
0135:       static const size_t inverse_rate = 3;
0136:       static const size_t input = 2;
0137:       static const size_t input_bit = 1;
0138:       static const size_t output = 8;
0139:       static const size_t output_bit = 3;
0140:       static const size_t smat2 = output;
0141:       explicit basic_r13_convolutional_code_encoder ( size_t memories_size )
0142:         : basic_r1n_convolutional_code_encoder ( memories_size, 3 ) {}
0143:       virtual ~basic_r13_convolutional_code_encoder ( ) {}
0144:     };
0145:
0146:     class basic_r12_convolutional_code_encoder
0147:       : public basic_r1n_convolutional_code_encoder
0148:     {
0149:     public:
0150:       static const size_t inverse_rate = 2;
0151:       static const size_t input = 2;
0152:       static const size_t input_bit = 1;
0153:       static const size_t output = 4;
0154:       static const size_t output_bit = 2;
0155:       static const size_t smat2 = output;
0156:       explicit basic_r12_convolutional_code_encoder ( size_t memories_size )
0157:         : basic_r1n_convolutional_code_encoder ( memories_size, 2 ) {}
0158:       virtual ~basic_r12_convolutional_code_encoder ( ) {}
0159:     };
0160:
0161:     //  We generate the convolutional code encoder systematically by using Boost::Preprocesse.
0162:     //  We define the Boost::Preprocessor's macros to make the token.
0163:     #define CLASS_NAME( r, m,   )  \
0164:       BOOST_PP_CAT( class CR, BOOST_PP_CAT( r, BOOST_PP_CAT( M, BOOST_PP_CAT( m,  \
0165:       BOOST_PP_CAT( OptimumConvolutionalCodeEncoder, BOOST_PP_CAT( : , _))))))
0166:     #define BASE_CLASS_NAME( r,   )  \
0167:       BOOST_PP_CAT( public basic_r, BOOST_PP_CAT( r,  \
0168:       BOOST_PP_CAT(  convolutional_code_encoder, BOOST_PP_CAT( { public:, _ ))))
0169:     #define PARAMETERS_STATUS( m,   )  \
0170:       BOOST_PP_CAT( static const size_t status=,  \
0171:       BOOST_PP_CAT( BOOST_PP_ARRAY_ELEM( BOOST_PP_SUB(m,2), STATUS_NUMBER),  \
0172:       BOOST_PP_CAT( ;,  ,))
0173:     #define STATUS_NUMBER  \
0174:       (12, (4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096, 8192 ) )
0175:     #define PARAMETERS_STATUS_BIT( m,   )  \
0176:       BOOST_PP_CAT( static const size_t status_bit=, BOOST_PP_CAT ( m,  \
0177:       BOOST_PP_CAT( ;,  ,))
0178:     #define PARAMETERS_SMAT_1(   )  \
0179:       BOOST_PP_CAT( static const size_t smat1=status;, _ )
0180:     #define CONSTRACTOR( r, m,   )  \
0181:       BOOST_PP_CAT( CR, BOOST_PP_CAT( r, BOOST_PP_CAT( M,  \
0182:       BOOST_PP_CAT( m, BOOST_PP_CAT( OptimumConvolutionalCodeEncoder(),  \
0183:       BOOST_PP_CAT( : basic_r, BOOST_PP_CAT( r,  \
0184:       BOOST_PP_CAT(  convolutional_code_encoder BOOST_PP_LPAREN(),  \
0185:       BOOST_PP_CAT( m, BOOST_PP_CAT( BOOST_PP_RPAREN() {}, _ )))))))))
0186:     #define DESTRUCTOR( r, m,   )  \
0187:       BOOST_PP_CAT( virtual ~CR, BOOST_PP_CAT( r, BOOST_PP_CAT( M,  \
0188:       BOOST_PP_CAT( m, BOOST_PP_CAT( OptimumConvolutionalCodeEncoder() {},  _ )))))
0189:     #define PRE_INITIALIZE(   )  \
0190:       BOOST_PP_CAT( void initialize () {, _ )
0191:     #define EQUATIONS( i, g, m,   )  \
0192:       BOOST_PP_CAT( generator_equations_[, BOOST_PP_CAT( i,  \
0193:       BOOST_PP_CAT( ].initialize(g, BOOST_PP_ADD(m,1));, _ )))
0194:     #define POST_INITIALIZE( _ )  \
0195:       BOOST_PP_CAT( },  )
0196:     #define GENERATE_CR14_OPTIMUM_CONVOLUTIONAL_CODE_ENCODER(m, g0, g1, g2, g3)  \
0197:       CLASS_NAME( 14, m, BASE_CLASS_NAME(14, PARAMETERS_STATUS( m,  \
0198:       PARAMETERS_STATUS_BIT( m, PARAMETERS_SMAT_1( CONSTRACTOR( 14, m,  \
0199:       DESTRUCTOR( 14, m, PRE_INITIALIZE( EQUATIONS( 0, g0, m,  \
0200:       EQUATIONS( 1, g1, m, EQUATIONS( 2, g2, m, EQUATIONS( 3, g3, m,  \
0201:       POST_INITIALIZE( }; )))))))))))))
0202:     #define GENERATE_CR13_OPTIMUM_CONVOLUTIONAL_CODE_ENCODER(m, g0, g1, g2)  \
0203:       CLASS_NAME( 13, m, BASE_CLASS_NAME(13, PARAMETERS_STATUS( m,  \
0204:       PARAMETERS_STATUS_BIT( m, PARAMETERS_SMAT_1( CONSTRACTOR( 13, m,  \
0205:       DESTRUCTOR( 13, m, PRE_INITIALIZE( EQUATIONS( 0, g0, m,  \
0206:       EQUATIONS( 1, g1, m, EQUATIONS( 2, g2, m, POST_INITIALIZE( }; )))))))))))
0207:     #define GENERATE_CR12_OPTIMUM_CONVOLUTIONAL_CODE_ENCODER(m, g0, g1)  \
0208:       CLASS_NAME( 12, m, BASE_CLASS_NAME(12, PARAMETERS_STATUS( m,  \
0209:       PARAMETERS_STATUS_BIT( m, PARAMETERS_SMAT_1( CONSTRACTOR( 12, m,  \
0210:       DESTRUCTOR( 12, m, PRE_INITIALIZE( EQUATIONS( 0, g0, m,  \
0211:       EQUATIONS( 1, g1, m, POST_INITIALIZE( }; )))))))))))
0212:
0213:     //  I define the convolutional code encoder, e.g., class CR14M2OptimumConvolutionalCodeEncoder
0214:     GENERATE_CR14_OPTIMUM_CONVOLUTIONAL_CODE_ENCODER( 2, 05, 05, 07, 07 )
0215:     GENERATE_CR14_OPTIMUM_CONVOLUTIONAL_CODE_ENCODER( 3, 013, 013, 015, 017 )
0216:     GENERATE_CR14_OPTIMUM_CONVOLUTIONAL_CODE_ENCODER( 4, 025, 027, 033, 037 )
0217:     GENERATE_CR14_OPTIMUM_CONVOLUTIONAL_CODE_ENCODER( 5, 045, 053, 067, 077 )
0218:     GENERATE_CR14_OPTIMUM_CONVOLUTIONAL_CODE_ENCODER( 6, 0117, 0127, 0155, 0171 )
0219:     GENERATE_CR14_OPTIMUM_CONVOLUTIONAL_CODE_ENCODER( 7, 0257, 0311, 0337, 0355 )
0220:     GENERATE_CR14_OPTIMUM_CONVOLUTIONAL_CODE_ENCODER( 8, 0533, 0575, 0647, 0711 )
0221:     GENERATE_CR14_OPTIMUM_CONVOLUTIONAL_CODE_ENCODER( 9, 01173, 01325, 01467, 01751 )
0222:     GENERATE_CR14_OPTIMUM_CONVOLUTIONAL_CODE_ENCODER( 10, 02327, 02353, 02671, 03175 )
0223:     GENERATE_CR14_OPTIMUM_CONVOLUTIONAL_CODE_ENCODER( 11, 04767, 05723, 06265, 07455 )
```

```
0224:    GENERATE_CR14_OPTIMUM_CONVOLUTIONAL_CODE_ENCODER( 12, 011145, 012477, 015537, 016727 )
0225:    GENERATE_CR14_OPTIMUM_CONVOLUTIONAL_CODE_ENCODER( 13, 021113, 023175, 035527, 035537 )
0226:
0227:    GENERATE_CR13_OPTIMUM_CONVOLUTIONAL_CODE_ENCODER( 2, 05, 07, 07 )
0228:    GENERATE_CR13_OPTIMUM_CONVOLUTIONAL_CODE_ENCODER( 3, 013, 015, 017 )
0229:    GENERATE_CR13_OPTIMUM_CONVOLUTIONAL_CODE_ENCODER( 4, 025, 033, 037 )
0230:    GENERATE_CR13_OPTIMUM_CONVOLUTIONAL_CODE_ENCODER( 5, 047, 053, 075 )
0231:    GENERATE_CR13_OPTIMUM_CONVOLUTIONAL_CODE_ENCODER( 6, 0117, 0127, 0155 )
0232:    GENERATE_CR13_OPTIMUM_CONVOLUTIONAL_CODE_ENCODER( 7, 0225, 0331, 0367 )
0233:    GENERATE_CR13_OPTIMUM_CONVOLUTIONAL_CODE_ENCODER( 8, 0575, 0623, 0727 )
0234:    GENERATE_CR13_OPTIMUM_CONVOLUTIONAL_CODE_ENCODER( 9, 01167, 01375, 01545 )
0235:    GENERATE_CR13_OPTIMUM_CONVOLUTIONAL_CODE_ENCODER( 10, 02325, 02731, 03747 )
0236:    GENERATE_CR13_OPTIMUM_CONVOLUTIONAL_CODE_ENCODER( 11, 023, 0, 0 )
0237:    GENERATE_CR13_OPTIMUM_CONVOLUTIONAL_CODE_ENCODER( 12, 010533, 010675, 017661 )
0238:    GENERATE_CR13_OPTIMUM_CONVOLUTIONAL_CODE_ENCODER( 13, 021645, 035661, 037133 )
0239:
0240:    GENERATE_CR12_OPTIMUM_CONVOLUTIONAL_CODE_ENCODER( 2, 05, 07 )
0241:    GENERATE_CR12_OPTIMUM_CONVOLUTIONAL_CODE_ENCODER( 3, 013, 017 )
0242:    GENERATE_CR12_OPTIMUM_CONVOLUTIONAL_CODE_ENCODER( 4, 027, 031 )
0243:    GENERATE_CR12_OPTIMUM_CONVOLUTIONAL_CODE_ENCODER( 5, 053, 075 )
0244:    GENERATE_CR12_OPTIMUM_CONVOLUTIONAL_CODE_ENCODER( 6, 0117, 0155 )
0245:    GENERATE_CR12_OPTIMUM_CONVOLUTIONAL_CODE_ENCODER( 7, 0247, 0371 )
0246:    GENERATE_CR12_OPTIMUM_CONVOLUTIONAL_CODE_ENCODER( 8, 0561, 0753 )
0247:    GENERATE_CR12_OPTIMUM_CONVOLUTIONAL_CODE_ENCODER( 9, 01131, 01537 )
0248:    GENERATE_CR12_OPTIMUM_CONVOLUTIONAL_CODE_ENCODER( 10, 02473, 03217 )
0249:    GENERATE_CR12_OPTIMUM_CONVOLUTIONAL_CODE_ENCODER( 11, 04325, 06747 )
0250:    GENERATE_CR12_OPTIMUM_CONVOLUTIONAL_CODE_ENCODER( 12, 010627, 016765 )
0251:    GENERATE_CR12_OPTIMUM_CONVOLUTIONAL_CODE_ENCODER( 13, 027251, 037363 )
0252:
0253:    #undef GENERATE_CR12_OPTIMUM_CONVOLUTIONAL_CODE_ENCODER
0254:    #undef GENERATE_CR13_OPTIMUM_CONVOLUTIONAL_CODE_ENCODER
0255:    #undef GENERATE_CR14_OPTIMUM_CONVOLUTIONAL_CODE_ENCODER
0256:    #undef POST_INITIALIZE
0257:    #undef EQUATIONS
0258:    #undef PRE_INITIALIZE
0259:    #undef DESTRUCTOR
0260:    #undef CONSTRACTOR
0261:    #undef PARAMETERS_SMAT_1
0262:    #undef PARAMETERS_STATUS_BIT
0263:    #undef STATUS_NUMBER
0264:    #undef PARAMETERS_STATUS
0265:    #undef BASE_CLASS_NAME
0266:    #undef CLASS_NAME
0267:
0268:  }  }  //  End of the namespace ecc::convolutional
```

A.2.3 ビタビ復号

```
0001:  //  viterbi.hpp:
0002:  //    Definitions of Viterbi decoding classes.
0003:  ////////////////////////////////////////////////////////////////////////
0004:
0005:  #pragma once
0006:  #include "bits.hpp"
0007:  #include "shift_register.hpp"
0008:  #include "convolutional.hpp"
0009:  #include "dsp.hpp"
0010:  #include <ostream>
0011:  #include <iomanip>
0012:  #include <math.h>
0013:  #include <boost/numeric/ublas/matrix.hpp>
0014:
0015:  namespace ecc { namespace viterbi {
0016:    //  Definitions of the state transition and output matrixes. I define the elements of these matrixes.
0017:    template < typename ElementType >
0018:    struct state_transition_and_output_matrixes_elements
0019:    {
0020:      typedef typename ElementType element_type;
0021:      element_type state_;
0022:      element_type output_;
0023:    state_transition_and_output_matrixes_elements(const ElementType& state,const ElementType& output )
0024:        : state_(state), output_(output) { }
0025:      state_transition_and_output_matrixes_elements() { }
0026:    };
0027:
0028:    //  Typedefs of the struct state_transition_and_output_matrixes_elements.
0029:    typedef struct state_transition_and_output_matrixes_elements < unsigned long >
0030:      STATE_TRANSITION_AND_OUTPUT_MATRIXES_ELEMENTS, *LPSTATE_TRANSITION_AND_OUTPUT_MATRIXES_ELEMENTS;
0031:
0032:    //  I define the state transition and output matrixes. Elements of each matrix are defined above.
0033:    //  I define its matrix, e.g., smat[previously][input]=destination or output, where previously and
0034:    //  input values are powers of two, i.e., smat_max[2^status number][2^input digit number].
0035:    //  Note that, if we'd like to use the R=1/2, M=6 convolutional code encoder, set as 64 and 2,
0036:    //  due to 2^6 and R=1/??.
0037:    typedef boost::numeric::ublas::matrix < STATE_TRANSITION_AND_OUTPUT_MATRIXES_ELEMENTS >
0038:      STATE_TRANSITION_AND_OUTPUT_MATRIXES, *LPSTATE_TRANSITION_AND_OUTPUT_MATRIXES;
0039:
0040:    //  I define the generation function that is calculated the state transition  and output matrixes
0041:    //  from the convolutional code encoder. I've defined the arguments as follows:
0042:    //  status: status number (bits), input or output: number of encoder input or output (bit),
0043:    //  smat: smat[previously][input]=destination or output where previously and input values are powers of two,
0044:    //  i.e., smat_max[2^status_number][2^input digit number].
0045:    template < typename EncoderType >
0046:    void makeStateTransitionAndOutputMatrixes ( STATE_TRANSITION_AND_OUTPUT_MATRIXES& smat )
0047:    {
0048:      EncoderType encoder;
0049:      encoder.initialize();
0050:      smat.resize ( EncoderType::smat1, EncoderType::smat2 );
0051:      for ( unsigned long st = 0UL; st < EncoderType::status; ++st )
0052:        for ( unsigned long in = 0UL; in < EncoderType::input; ++in )
0053:        {
0054:          //  I've set the initial encoder status, which is calculated all encoder status.
0055:          encoder.setStatus(st);
0056:          //  I've calculated the output matrix here.
0057:          EncoderType::value_type ibits;
0058:          EncoderType::value_type obits;
0059:          ibits.bits_[0] = in << ( 32 - EncoderType::input_bit );
0060:          encoder.encoding ( ibits, obits, EncoderType::input_bit );
0061:          //  I've obtained the status of convolutional code encoder.
0062:          unsigned long stat = encoder.getStatus();
0063:          unsigned long outp = 0UL;
0064:          outp = ( obits.bits_[0] >> ( 32 - EncoderType::output_bit ) ) & ( EncoderType::output - 1 );
0065:          smat(st,in) = STATE_TRANSITION_AND_OUTPUT_MATRIXES_ELEMENTS ( stat, outp );
0066:        }
0067:    }
0068:
0069:    //  This function shows the values of state transition and output matrixes.
0070:    template < typename EncoderType >
0071:    void printStateTransitionAndOutputMatrixes (
0072:      std::ostream& out, const STATE_TRANSITION_AND_OUTPUT_MATRIXES& smat )
0073:    {
0074:      using namespace std;
0075:      out << "From --- Output/Input --> To" << endl;
0076:      for ( unsigned long st = 0; st < EncoderType::status; ++st )
0077:        for ( unsigned long in = 0; in < EncoderType::input; ++in )
0078:        {
0079:          out << hex << st << " --- ";
0080:          out << hex << smat(st, in).output_ << " / ";
0081:          out << hex << in << " --> ";
0082:          out << hex << smat(st, in).state_ << endl;
0083:        }
0084:      }
0085:    }
0086:
0087:    //  Definitions of the trellis. At first, we define the element.
0088:    template < typename DigitType, typename NodeType >
0089:    struct TrellisElement
0090:    {
0091:      typedef typename DigitType digit_type;
0092:      typedef typename NodeType node_type;
0093:      DigitType prev_;
0094:      DigitType branch_;
```

```
0095:       NodeType node_;
0096:       bool fill_;
0097:       TrellisElement < DigitType, NodeType > (
0098:         const DigitType& prev, const DigitType& branch, const NodeType& node, bool fill )
0099:         : prev_(prev), branch_(branch), node_(node), fill_(fill) { }
0100:       TrellisElement<DigitType, NodeType> (
0101:         const DigitType& prev, const DigitType& branch, const NodeType& node )
0102:         : prev_(prev), branch_(branch), node_(node), fill_(false) { }
0103:       TrellisElement < DigitType, NodeType > ()
0104:         : node_(-1), fill_(false) { }
0105:       TrellisElement<DigitType,NodeType>& operator= ( const TrellisElement<DigitType,NodeType>& that )
0106:       {
0107:         prev_ = that.prev_;
0108:         branch_ = that.branch_;
0109:         node_ = that.node_;
0110:         fill_ = that.fill_;
0111:         return *this;
0112:       }
0113:     };
0114:
0115:     //  I define the `<' and `>' operators, respectively.
0116:     template < typename DigitType, typename NodeType >
0117:     bool operator < ( const struct TrellisElement < DigitType, NodeType > & lhs,
0118:       const struct TrellisElement < DigitType, NodeType > & rhs )
0119:     {
0120:       return lhs.node_ < rhs.node_;
0121:     }
0122:     template < typename DigitType, typename NodeType >
0123:     bool operator > ( const struct TrellisElement < DigitType, NodeType > & lhs,
0124:       const struct TrellisElement < DigitType, NodeType > & rhs )
0125:     {
0126:       return lhs.node_ > rhs.node_;
0127:     }
0128:
0129:     //  Typedefs of the TrellisElement structure.
0130:     typedef struct TrellisElement < unsigned long, int >
0131:       HARD_DECISION_TRELLIS_ELEMENT, *LPHARD_DECISION_TRELLIS_ELEMENT;
0132:     typedef struct TrellisElement < unsigned long, float >
0133:       SOFT_DECISION_TRELLIS_ELEMENT, *LPSOFT_DECISION_TRELLIS_ELEMENT;
0134:
0135:     //  I define the trellis matrix.
0136:     typedef boost::numeric::ublas::matrix < HARD_DECISION_TRELLIS_ELEMENT >
0137:       HARD_DECISION_TRELLIS, *LPHARD_DECISION_TRELLIS;
0138:     typedef boost::numeric::ublas::matrix < SOFT_DECISION_TRELLIS_ELEMENT >
0139:       SOFT_DECISION_TRELLIS, *LPSOFT_DECISION_TRELLIS;
0140:
0141:     //  I define the stacks that requires to generate them
0142:     template < typename RealType >
0143:     class CTrellisStacks
0144:       : public std::vector < RealType > { };
0145:
0146:     //  Typedefs of the CTrellisStacks class.
0147:     typedef CTrellisStacks < HARD_DECISION_TRELLIS_ELEMENT >
0148:       HARD_DECISION_TRELLIS_STACKS, *LPHARD_DECISION_TRELLIS_STACKS;
0149:     typedef CTrellisStacks<SOFT_DECISION_TRELLIS_ELEMENT>
0150:       SOFT_DECISION_TRELLIS_STACKS, *LPSOFT_DECISION_TRELLIS_STACKS;
0151:
0152:     //  I define the temporary arrays which is required to generate the trellis.
0153:     template < typename RealType >
0154:     class CTrellisArrays
0155:     {
0156:     public:
0157:       typedef typename RealType real_type;
0158:     private:
0159:       std::vector < CTrellisStacks < RealType > > arrays_;
0160:     public:
0161:       explicit CTrellisArrays ( size_t size )
0162:       {
0163:         arrays_.resize(size);
0164:       }
0165:       virtual ~CTrellisArrays ( ) { }
0166:       void clear ( )
0167:       {
0168:         for ( size_t i = 0; i < arrays_.size(); ++i )
0169:           arrays_[i].clear();
0170:       }
0171:       void push ( size_t index, const RealType& element )
0172:       {
0173:         arrays_[index].push_back(element);
0174:       }
0175:       bool empty ( size_t index ) const
0176:       {
0177:         return arrays_[index].empty();
0178:       }
0179:
0180:       //  I've calculated the branch metrics and decide the survived paths.
0181:       RealType survive ( size_t index )
0182:       {
0183:         CTrellisStacks<RealType>::iterator ret;
0184:         ret = min_element ( arrays_[index].begin(), arrays_[index].end() );
0185:         return *ret;
0186:       }
0187:     };
0188:
0189:     //  Typedefs of the CTrellisArrays class.
0190:     typedef CTrellisArrays < HARD_DECISION_TRELLIS_ELEMENT >
0191:       HARD_DECISION_TRELLIS_ARRAYS, *LPHARD_DECISION_TRELLIS_ARRAYS;
0192:     typedef CTrellisArrays < SOFT_DECISION_TRELLIS_ELEMENT >
0193:       SOFT_DECISION_TRELLIS_ARRAYS, *LPSOFT_DECISION_TRELLIS_ARRAYS;
0194:
0195:     //  Definitions of the syndrome structure.
0196:     template < size_t FRAME_SIZE >
0197:     struct syndrome
0198:     {
0199:       static const size_t frame_size = FRAME_SIZE;
0200:       size_t error_positions_[ FRAME_SIZE ];
0201:       size_t error_number_;
0202:       size_t retransmission_number_;
0203:       syndrome ()
0204:         : error_number_ ( FRAME_SIZE ), retransmission_number_ (0)
0205:       {
0206:         for ( size_t i = 0; i < FRAME_SIZE; ++i )
0207:           error_positions_[i] = i;
0208:       }
0209:       explicit syndrome ( size_t error_number )
0210:         : error_number_ ( error_number ), retransmission_number_(0)
0211:       {
0212:         for ( size_t i = 0; i < error_number; ++i )
0213:           error_positions_[i] = i;
0214:       }
0215:       syndrome ( size_t error_number, size_t retransmission_number )
0216:         : error_number_ ( error_number ), retransmission_number_ ( retransmission_number )
0217:       {
0218:         for ( size_t i = 0; i < error_number; ++i )
0219:           error_positions_[i] = i;
0220:       }
0221:       syndrome ( const syndrome& that )
0222:         : error_number_ ( that.error_number_ ), retransmission_number_ ( that.retransmission_number_ )
0223:       {
0224:         for ( size_t i = 0; i < that.error_number_; ++i )
0225:           error_positions_[i] = that.error_positions_[i];
0226:       }
0227:       syndrome& operator= ( const syndrome& that )
0228:       {
0229:         error_number_ = that.error_number_;
0230:         retransmission_number_ = that.retransmission_number_;
0231:         for ( size_t i = 0; i < that.error_number_; ++i )
0232:           error_positions_[i] = that.error_positions_[i];
0233:         return *this;
0234:       }
0235:       size_t error ()
0236:       {
0237:         return ( error_number_ == FRAME_SIZE ) ? 2 : ( error_number_ == 0 ) ? 0 : 1;
0238:       }
0239:       void setNextPosition ( size_t next_position )
0240:       {
```

```
0241:         error_positions_ [ error_number_++ ] = next_position;
0242:       }
0243:     };
0244:
0245:     //  I can calculate the Hamming distance.
0246:     int hamming ( size_t size, unsigned long x, unsigned long y )
0247:     {
0248:       int ret = 0;
0249:       for ( size_t i = 0; i < size; ++i )
0250:         if ( ( ( x >> i ) & 0x1 ) != ( ( y >> i ) & 0x1 ) )
0251:           ++ret;
0252:       return ret;
0253:     }
0254:
0255:     //  I define the hard decison Viterbi decoding algorithm.
0256:     template < typename EncoderType, typename SequenceType1, typename SequenceType2 >
0257:     void hard_decode(const STATE_TRANSITION_AND_OUTPUT_MATRIXES& smat,const SequenceType1& s1, SequenceType2& s2)
0258:     {
0259:       const size_t symbols = SequenceType1::size / EncoderType::output_bit;
0260:       HARD_DECISION_TRELLIS trellis ( EncoderType::status, symbols );
0261:       HARD_DECISION_TRELLIS_ARRAYS temporary_array ( EncoderType::status );
0262:       size_t shift = SequenceType1::block_size - EncoderType::output_bit;
0263:       unsigned long mask = EncoderType::output - 1;
0264:       SequenceType1::bits_type r0 = ( s1.bits_[0] >> shift ) & mask;
0265:
0266:       for ( size_t in = 0; in < EncoderType::input; ++in )
0267:       {
0268:         int dist = hamming ( EncoderType::output_bit, r0, smat(0,in).output_ );
0269:         HARD_DECISION_TRELLIS_ELEMENT elem;
0270:         elem.prev_  = 0UL;
0271:         elem.node_  = dist;
0272:         elem.branch_  = in;
0273:         temporary_array.push ( smat(0,in).state_, elem );
0274:       }
0275:
0276:       for ( size_t sym = 0; sym < symbols-1; ++sym )
0277:       {
0278:         for ( size_t st = 0; st < EncoderType::status; ++st )
0279:         {
0280:           if ( temporary_array.empty(st) )
0281:             continue;
0282:           HARD_DECISION_TRELLIS_ELEMENT elem;
0283:           elem = temporary_array.survive(st);
0284:           elem.fill_  = true;
0285:           trellis(st,sym) = elem;
0286:         }
0287:         temporary_array.clear();
0288:
0289:         for ( size_t st = 0; st < EncoderType::status; ++st )
0290:         {
0291:           if ( !trellis(st,sym).fill_ )
0292:             continue;
0293:           size_t begin_position = ( sym + 1 ) * EncoderType::output_bit;
0294:           size_t shift =
0295:            SequenceType1::block_size-begin_position%SequenceType1::block_size-EncoderType::output_bit;
0296:           size_t index = begin_position / SequenceType1::block_size;
0297:           unsigned long mask = EncoderType::output - 1;
0298:           SequenceType1::bits_type r = ( s1.bits_[index] >> shift ) & mask;
0299:
0300:           for ( size_t in = 0; in < EncoderType::input; ++in )
0301:           {
0302:             int dist = hamming ( EncoderType::output_bit, r, smat(st, in).output_ );
0303:             dist += trellis(st, sym).node_;
0304:             HARD_DECISION_TRELLIS_ELEMENT elem;
0305:             elem.prev_  = st;
0306:             elem.node_  = dist;
0307:             elem.branch_  = in;
0308:             temporary_array.push ( smat(st, in).state_, elem );
0309:           }
0310:         }
0311:
0312:       for ( size_t st = 0; st < EncoderType::status; ++st )
0313:       {
0314:         if ( temporary_array.empty(st) )
0315:           continue;
0316:         HARD_DECISION_TRELLIS_ELEMENT elem;
0317:         elem = temporary_array.survive(st);
0318:         elem.fill_  = true;
0319:         trellis(st, symbols-1) = elem;
0320:       }
0321:
0322:       s2.clear();
0323:       s2.bits_[SequenceType2::length-1] = trellis(0UL, symbols-1).branch_;
0324:
0325:       unsigned long prev = trellis(0UL, symbols-1).prev_;
0326:       for ( int i = symbols-2; i >= 0; --i )
0327:       {
0328:         size_t begin_position = i * EncoderType::input_bit;
0329:         size_t shift =
0330:           SequenceType2::block_size - begin_position % SequenceType2::block_size - EncoderType::input_bit;
0331:         size_t index = begin_position / SequenceType2::block_size;
0332:         unsigned long mask = EncoderType::input - 1;
0333:         SequenceType2::bits_type r = trellis(prev, i).branch_;
0334:         s2.bits_[index] |= ( ( r & mask ) << shift );
0335:         prev = trellis(prev, i).prev_;
0336:       }
0337:     }
0338:
0339:     //  I've define the soft decison Viterbi decoding algorithm.
0340:     template < typename EncoderType, typename SequenceType1, typename SequenceType2 >
0341:     void soft_decode (const STATE_TRANSITION_AND_OUTPUT_MATRIXES& smat,const SequenceType1& s1,SequenceType2& s2 )
0342:     {
0343:       const size_t symbols = SequenceType1::size / EncoderType::output_bit;
0344:       SOFT_DECISION_TRELLIS trellis ( EncoderType::status, symbols );
0345:       SOFT_DECISION_TRELLIS_ARRAYS temporary_array ( EncoderType::status );
0346:
0347:       for ( size_t in = 0; in < EncoderType::input; ++in )
0348:       {
0349:         float dist = 0.0f;
0350:         for ( size_t i = 0; i < EncoderType::output_bit; ++i )
0351:         {
0352:           unsigned long prob_bit =  ( smat(0,in).output_ >> ( EncoderType::output_bit-i-1 ) ) & 0x1UL;
0353:           dist += s1[i].prob_[prob_bit];
0354:         }
0355:
0356:         SOFT_DECISION_TRELLIS_ELEMENT elem;
0357:         elem.prev_  = 0UL;
0358:         elem.node_  = dist;
0359:         elem.branch_  = in;
0360:         temporary_array.push ( smat(0,in).state_, elem );
0361:       }
0362:
0363:       for ( size_t sym = 0; sym < symbols-1; ++sym )
0364:       {
0365:         for ( size_t st = 0; st < EncoderType::status; ++st )
0366:         {
0367:           if ( temporary_array.empty(st) )
0368:             continue;
0369:           SOFT_DECISION_TRELLIS_ELEMENT elem;
0370:           elem = temporary_array.survive(st);
0371:           elem.fill_  = true;
0372:           trellis(st,sym) = elem;
0373:         }
0374:         temporary_array.clear();
0375:
0376:         for ( size_t st = 0; st < EncoderType::status; ++st )
0377:         {
0378:           if ( !trellis(st,sym).fill_ )
0379:             continue;
0380:           for ( size_t in = 0; in < EncoderType::input; ++in )
0381:           {
0382:             float dist = 0.0f;
0383:             for ( size_t i = 0; i < EncoderType::output_bit; ++i )
0384:             {
0385:               size_t index = EncoderType::output_bit*(sym+1) + i;
0386:               unsigned long prob_bit =
```

```
0387:                 (smat(st,in).output_ >> ( EncoderType::output_bit-i-1)) & 0x1;
0388:             dist += s1[index].prob_[prob_bit];
0389:           }
0390:           dist += trellis(st, sym).node_;
0391:           SOFT_DECISION_TRELLIS_ELEMENT elem;
0392:           elem.prev_  = st;
0393:           elem.node_  = dist;
0394:           elem.branch_  = in;
0395:           temporary_array.push ( smat(st,in).state_, elem );
0396:         }
0397:       }
0398:     }
0399:
0400:     //  I’ve terminated the trellis object, i.e., I’ve finished the status 00.
0401:     for ( size_t st = 0; st < EncoderType::status; ++st )
0402:     {
0403:       if ( temporary_array.empty(st) )
0404:         continue;
0405:       SOFT_DECISION_TRELLIS_ELEMENT elem;
0406:        elem = temporary_array.survive(st);
0407:       elem.fill_  = true;
0408:       trellis(st, symbols-1) = elem;
0409:     }
0410:
0411:     //  I’ve decoded the estimated sequences here.
0412:     s2.clear();
0413:     s2.bits_[SequenceType2::length-1] = trellis(0UL, symbols-1).branch_;
0414:     unsigned long prev = trellis(0UL, symbols-1).prev_;
0415:     for ( int i = symbols-2; i >= 0; --i )
0416:     {
0417:       size_t begin_position = i * EncoderType::input_bit;
0418:       size_t shift =
0419:          SequenceType2::block_size-begin_position%SequenceType2::block_size-EncoderType::input_bit;
0420:       size_t index = begin_position / SequenceType2::block_size;
0421:       unsigned long mask = EncoderType::input - 1;
0422:       SequenceType2::bits_type r = trellis(prev,i).branch_;
0423:       s2.bits_[index] |= ( ( r & mask ) << shift );
0424:       prev = trellis(prev,i).prev_;
0425:     }
0426:   }
0427:
0428: } }  //  End of the namespace ecc::viterbi
```

A.2.4 インターリーバ

```
0001: //  interleaver.hpp:
0002: //     Definitions of interleaver and deinterleaver classes.
0003: ////////////////////////////////////////////////////////////////////////
0004:
0005: #pragma once
0006:
0007: namespace ecc { namespace interleaver {
0008:   //  I define the block interleaver and deinterleaver classes.
0009:   //  This structure is high performance but it is only used for bit based interleaving.
0010:   template < size_t X, size_t Y >
0011:   struct block_bit_interleaver
0012:   {
0013:     static const size_t length_x = X;
0014:     static const size_t length_y = Y;
0015:     static const size_t size_x = X * 32;
0016:     static const size_t size_y = Y ;
0017:     static const size_t size = size_x * size_y;
0018:     boost::numeric::ublas::matrix < unsigned long > mat_;
0019:     block_bit_interleaver ( )
0020:       : mat_(X,Y) { }
0021:     void clear ( )
0022:     {
0023:       for ( size_t i = 0; i < X; ++i )
0024:         for ( size_t j = 0; j < Y; ++j )
0025:           mat_(i,j) = 0UL;
0026:     }
0027:     void setBlock ( size_t index, unsigned long value )
0028:     {
0029:       size_t lx = index % length_x;
0030:       size_t ly = index / length_x;
0031:       mat_(lx,ly) = value;
0032:     }
0033:     unsigned long getBlock ( size_t index ) const
0034:     {
0035:       size_t lx = index % length_x;
0036:       size_t ly = index / length_x;
0037:       return mat_(lx,ly);
0038:     }
0039:     void getBlock ( size_t index, unsigned long& value ) const
0040:     {
0041:       value = getBlock(index);
0042:     }
0043:     void setBit ( size_t index, unsigned long value )
0044:     {
0045:       size_t lx = ( index / 32 ) % length_x;
0046:       size_t ly = ( index / 32 ) / length_x;
0047:       size_t shift = 31 - ( index % 32 );
0048:       mat_(lx,ly) |= ( value << shift );
0049:     }
0050:     unsigned long getBit ( size_t index ) const
0051:     {
0052:       size_t lx = ( index / 32 ) % length_x;
0053:       size_t ly = ( index / 32 ) / length_x;
0054:       size_t shift = 31 - ( index % 32 );
0055:       return ( mat_(lx,ly) >> shift ) & 0x1UL;
0056:     }
0057:     void getBit ( size_t index, unsigned long& value ) const
0058:     {
0059:       value = getBit(index);
0060:     }
0061:     void setInvBit ( size_t index, unsigned long value )
0062:     {
0063:       size_t lx = ( index / length_y ) / 32;
0064:       size_t ly = index % length_y;
0065:       size_t shift = 31 - ( ( index / length_y ) % 32 );
0066:       mat_(lx,ly) |= ( value << shift );
0067:     }
0068:     unsigned long getInvBit ( size_t index ) const
0069:     {
0070:       size_t lx = ( index / length_y ) / 32;
0071:       size_t ly = index % length_y;
0072:       size_t shift = 31 - ( ( index / length_y ) % 32 );
0073:       return ( mat_(lx,ly) >> shift ) & 0x1UL;
0074:     }
0075:     void getInvBit ( size_t index, unsigned long& value ) const
0076:     {
0077:       value = getInvBit ( index );
0078:     }
0079:     //  The operator+ is equal to be getInvBit function.
0080:     unsigned long operator() ( size_t index ) const
0081:     {
0082:       return getInvBit(index);
0083:     }
0084:     void operator() ( size_t index, unsigned long value )
0085:     {
0086:       setInvBit(index, value);
0087:     }
0088:   };
0089:
0090:   template < typename InterleaverType, typename SequenceType >
0091:   void block_bit_interleaving(InterleaverType& interleaver,const SequenceType& s1,SequenceType& s2 )
0092:   {
0093:     interleaver.clear();
0094:     for ( size_t i = 0; i < SequenceType::length; ++i )
0095:       interleaver.setBlock ( i, s1.bits_[i] );
0096:     for ( size_t i = 0; i < SequenceType::length; ++i )
0097:     {
```

```
0098:         unsigned long ret = 0UL;
0099:         for ( size_t j = 0; j < 32; ++j )
0100:           ret |= ( interleaver(32*i+j) << (31-j) );
0101:         s2.bits_[i] = ret;
0102:       }
0103:     }
0104:
0105:     template < typename InterleaverType, typename SequenceType >
0106:     void block_bit_deinterleaving ( InterleaverType& deinterleaver, const SequenceType& s1, SequenceType& s2 )
0107:     {
0108:       deinterleaver.clear();
0109:       for ( size_t i = 0; i < SequenceType::size; ++i )
0110:         deinterleaver.setInvBit ( i, s1(i) );
0111:       for ( size_t i = 0; i < SequenceType::length; ++i )
0112:         s2.bits_[i] = deinterleaver.getBlock(i);
0113:     }
0114:
0115:     //  I define the block interleaver and deinterleaver classes.
0116:     template < size_t X, size_t Y, typename RealType = unsigned long >
0117:     struct block_interleaver
0118:     {
0119:       typedef typename RealType real_type;
0120:       static const size_t size_x = X;
0121:       static const size_t size_y = Y;
0122:       static const size_t size = X * Y;
0123:       boost::numeric::ublas::matrix < RealType > mat_;
0124:       block_interleaver ( )
0125:         : mat_ (X,Y) { }
0126:       void  clear ( )
0127:       {
0128:         for ( size_t i = 0; i < X; ++i )
0129:           for ( size_t j = 0; j < Y; ++j )
0130:             mat_(i,j).clear();
0131:       }
0132:       void setBit ( size_t index, const RealType& value )
0133:       {
0134:         size_t lx = index % size_x;
0135:         size_t ly = index / size_x;
0136:         mat_(lx,ly) = value;
0137:       }
0138:       RealType getBit ( size_t index ) const
0139:       {
0140:         size_t lx = index % size_x;
0141:         size_t ly = index / size_x;
0142:         return mat_(lx,ly);
0143:       }
0144:       void getBit ( size_t index, RealType& value ) const
0145:       {
0146:         value = getBit(index);
0147:       }
0148:       void setInvBit ( size_t index, const RealType& value )
0149:       {
0150:         size_t lx = index / size_y;
0151:         size_t ly = index % size_y;
0152:         mat_(lx,ly) = value;
0153:       }
0154:       RealType getInvBit ( size_t index ) const
0155:       {
0156:         size_t lx = index / size_y;
0157:         size_t ly = index % size_y;
0158:         return mat_(lx,ly);
0159:       }
0160:       void getInvBit ( size_t index, RealType& value ) const
0161:       {
0162:         value = getInvBit(index);
0163:       }
0164:       RealType operator() ( size_t index ) const
0165:       {
0166:         return getInvBit(index);
0167:       }
0168:       void operator() ( size_t index, const RealType& element )
0169:       {
0170:         setInvBit(index, element);
0171:       }
0172:     };
0173:
0174:     template < typename InterleaverType, typename SequenceElementType >
0175:     void block_interleaving ( InterleaverType& interleaver,
0176:       const std::vector < SequenceElementType >& s1, std::vector < SequenceElementType >& s2 )
0177:     {
0178:       const size_t vector_size = s1.size();
0179:       interleaver.clear();
0180:       for ( size_t i = 0; i < vector_size; ++i )
0181:         interleaver.setBit ( i, s1[i] );
0182:       for ( size_t i = 0; i < vector_size; ++i )
0183:         s2[i] = interleaver.getInvBit(i);
0184:     }
0185:
0186:     template < typename InterleaverType, typename SequenceElementType >
0187:     void block_deinterleaving ( InterleaverType& deinterleaver,
0188:       const std::vector < SequenceElementType >& s1, std::vector<SequenceElementType>& s2 )
0189:     {
0190:       const size_t vector_size = s1.size();
0191:       deinterleaver.clear();
0192:       for ( size_t i = 0; i < vector_size; ++i )
0193:         deinterleaver.setInvBit ( i, s1[i] );
0194:       for ( size_t i = 0; i < vector_size; ++i )
0195:         s2[i] = deinterleaver.getBit ( i );
0196:     }
0197:
0198:     template < typename InterleaverType, typename SequenceType >
0199:     void block_deinterleaving(InterleaverType& deinterleaver,const SequenceType& s1,SequenceType& s2 )
0200:     {
0201:       const size_t vector_size = SequenceType::size;
0202:       deinterleaver.clear();
0203:       for ( size_t i = 0; i < vector_size; ++i )
0204:         deinterleaver.setInvBit ( i, s1[i] );
0205:       for ( size_t i = 0; i < vector_size; ++i )
0206:         s2[i] = deinterleaver.getBit ( i );
0207:     }
0208:
0209:   } }  //  End of the namespace ecc::interleaver
```

A.3 変復調信号処理

```
0001:   //  dsp.hpp:
0002:   //    Definitions of digital signal processing classes.
0003:   ////////////////////////////////////////////////////////////////////
0004:
0005:   #pragma once
0006:   #include <ostream>
0007:   #include <vector>
0008:   #include <algorithm>
0009:   #include <boost/lambda/lambda.hpp>
0010:   #include <boost/static_assert.hpp>
0011:   #include <boost/numeric/ublas/matrix.hpp>
0012:   #include "iq.hpp"
0013:
0014:   namespace dsp {
0015:     //  I define the transformer functions between dB and true values.
0016:     float decibel_to_true ( float rhs )
0017:     {
0018:       return pow ( 10.0f, 0.1f * rhs );
0019:     }
0020:     double decibel_to_true ( double rhs )
0021:     {
0022:       return pow ( 10.0, 0.1 * rhs );
0023:     }
0024:     float true_to_decibel ( float rhs )
0025:     {
0026:       return 10.0f * log10 ( rhs );
0027:     }
```

```
0028:     double true_to_decibel ( double rhs )
0029:     {
0030:       return 10.0 * log10 ( rhs );
0031:     }
0032:
0033:     //  Definitions of the modulation methods
0034:     enum usage_modulation_methods
0035:     {
0036:       MODULATION_BPSK,
0037:       MODULATION_QPSK,
0038:       MODULATION_16QAM,
0039:       MODULATION_64QAM,  //  Future works
0040:     };
0041:
0042:     template < usage_modulation_methods METHOD >
0043:     struct modulation_type { };
0044:
0045:     template < >
0046:     struct modulation_type < MODULATION_BPSK >
0047:     {
0048:       typedef modulation_type < MODULATION_BPSK > modulation_method_type;
0049:       static const usage_modulation_methods method = MODULATION_BPSK;
0050:       static const size_t symbols = 2;
0051:       static const size_t bits_per_symbol = 1;
0052:     };
0053:
0054:     template < >
0055:     struct modulation_type < MODULATION_QPSK >
0056:     {
0057:       typedef modulation_type < MODULATION_QPSK > modulation_method_type;
0058:       static const usage_modulation_methods method = MODULATION_QPSK;
0059:       static const size_t symbols = 4;
0060:       static const size_t bits_per_symbol = 2;
0061:     };
0062:
0063:     template < >
0064:     struct modulation_type < MODULATION_16QAM >
0065:     {
0066:       typedef modulation_type < MODULATION_16QAM > modulation_method_type;
0067:       static const usage_modulation_methods method = MODULATION_16QAM;
0068:       static const size_t symbols = 16;
0069:       static const size_t bits_per_symbol = 4;
0070:     };
0071:
0072:     //  I define the bit/symbol mapper
0073:     template < typename ModulationMethodType >
0074:     class CMappingPattern { };
0075:
0076:     template < >
0077:     class CMappingPattern < modulation_type < MODULATION_BPSK > >
0078:     {
0079:     public:
0080:       typedef modulation_type < MODULATION_BPSK > modulation_method_type;
0081:     private:
0082:       std::vector < iq::CConstellation > pattern_;
0083:     public:
0084:       CMappingPattern ()
0085:         : pattern_(2)
0086:       {
0087:         using namespace iq;
0088:         const float scale = sqrt ( 2.0f );
0089:         pattern_[0] = CConstellation ( 1.0f, 0.0f) ^ scale;
0090:         pattern_[1] = CConstellation (-1.0f, 0.0f) ^ scale;
0091:       }
0092:       virtual ~CMappingPattern () { }
0093:       iq::CConstellation& operator[] ( size_t index )
0094:       {
0095:         return pattern_[index];
0096:       }
0097:       iq::CConstellation operator[] ( size_t index ) const
0098:       {
0099:         return pattern_[index];
0100:       }
0101:       iq::CConstellation at ( size_t index ) const
0102:       {
0103:         return pattern_[index];
0104:       }
0105:     };
0106:
0107:     template < >
0108:     class CMappingPattern < modulation_type < MODULATION_QPSK > >
0109:     {
0110:     public:
0111:       typedef modulation_type < MODULATION_QPSK > modulation_method_type;
0112:     private:
0113:       std::vector < iq::CConstellation > pattern_;
0114:     public:
0115:       CMappingPattern ()
0116:         : pattern_(4)
0117:       {
0118:         using namespace iq;
0119:         const float scale = sqrt ( 2.0f );
0120:         pattern_[0] = CConstellation( 1.0f,  0.0f) ^ scale;
0121:         pattern_[1] = CConstellation( 0.0f, -1.0f) ^ scale;
0122:         pattern_[2] = CConstellation( 0.0f,  1.0f) ^ scale;
0123:         pattern_[3] = CConstellation(-1.0f,  0.0f) ^ scale;
0124:       }
0125:       virtual ~CMappingPattern () { }
0126:       iq::CConstellation& operator[] ( size_t index )
0127:       {
0128:         return pattern_[index];
0129:       }
0130:       iq::CConstellation operator[] ( size_t index ) const
0131:       {
0132:         return pattern_[index];
0133:       }
0134:       iq::CConstellation at ( size_t index ) const
0135:       {
0136:         return pattern_[index];
0137:       }
0138:     };
0139:
0140:     template < >
0141:     class CMappingPattern < modulation_type < MODULATION_16QAM > >
0142:     {
0143:     public:
0144:       typedef modulation_type < MODULATION_16QAM > modulation_method_type;
0145:     private:
0146:       boost::numeric::ublas::matrix < iq::CConstellation > pattern_;
0147:     public:
0148:       CMappingPattern ( )
0149:         : pattern_ ( 4, 16 )
0150:       {
0151:         using namespace iq;
0152:         const float scale = 1.0f / sqrt ( 10.0f );
0153:         const float factor = 2.0f;
0154:         pattern_(0, 0) = CConstellation(-3.0f*scale, 3.0f*scale) ^ factor;
0155:         pattern_(0, 1) = CConstellation(-1.0f*scale, 3.0f*scale) ^ factor;
0156:         pattern_(0, 2) = CConstellation( 3.0f*scale, 3.0f*scale) ^ factor;
0157:         pattern_(0, 3) = CConstellation( 1.0f*scale, 3.0f*scale) ^ factor;
0158:         pattern_(0, 4) = CConstellation(-3.0f*scale, 1.0f*scale) ^ factor;
0159:         pattern_(0, 5) = CConstellation(-1.0f*scale, 1.0f*scale) ^ factor;
0160:         pattern_(0, 6) = CConstellation( 3.0f*scale, 1.0f*scale) ^ factor;
0161:         pattern_(0, 7) = CConstellation( 1.0f*scale, 1.0f*scale) ^ factor;
0162:         pattern_(0, 8) = CConstellation(-3.0f*scale,-3.0f*scale) ^ factor;
0163:         pattern_(0, 9) = CConstellation(-1.0f*scale,-3.0f*scale) ^ factor;
0164:         pattern_(0,10) = CConstellation( 3.0f*scale,-3.0f*scale) ^ factor;
0165:         pattern_(0,11) = CConstellation( 1.0f*scale,-3.0f*scale) ^ factor;
0166:         pattern_(0,12) = CConstellation(-3.0f*scale,-1.0f*scale) ^ factor;
0167:         pattern_(0,13) = CConstellation(-1.0f*scale,-1.0f*scale) ^ factor;
0168:         pattern_(0,14) = CConstellation( 3.0f*scale,-1.0f*scale) ^ factor;
0169:         pattern_(0,15) = CConstellation( 1.0f*scale,-1.0f*scale) ^ factor;
0170:         pattern_(1, 0) = CConstellation(-1.0f*scale,-3.0f*scale) ^ factor;
0171:         pattern_(1, 1) = CConstellation( 1.0f*scale,-3.0f*scale) ^ factor;
0172:         pattern_(1, 2) = CConstellation(-3.0f*scale,-3.0f*scale) ^ factor;
0173:         pattern_(1, 3) = CConstellation( 3.0f*scale,-3.0f*scale) ^ factor;
```

```
0174:         pattern_(1, 4) = CConstellation(-1.0f*scale, 3.0f*scale) ^ factor;
0175:         pattern_(1, 5) = CConstellation( 1.0f*scale, 3.0f*scale) ^ factor;
0176:         pattern_(1, 6) = CConstellation(-3.0f*scale, 3.0f*scale) ^ factor;
0177:         pattern_(1, 7) = CConstellation( 3.0f*scale, 3.0f*scale) ^ factor;
0178:         pattern_(1, 8) = CConstellation(-1.0f*scale,-1.0f*scale) ^ factor;
0179:         pattern_(1, 9) = CConstellation( 1.0f*scale,-1.0f*scale) ^ factor;
0180:         pattern_(1,10) = CConstellation(-3.0f*scale,-1.0f*scale) ^ factor;
0181:         pattern_(1,11) = CConstellation( 3.0f*scale,-1.0f*scale) ^ factor;
0182:         pattern_(1,12) = CConstellation(-1.0f*scale, 1.0f*scale) ^ factor;
0183:         pattern_(1,13) = CConstellation( 1.0f*scale, 1.0f*scale) ^ factor;
0184:         pattern_(1,14) = CConstellation(-3.0f*scale, 1.0f*scale) ^ factor;
0185:         pattern_(1,15) = CConstellation( 3.0f*scale, 1.0f*scale) ^ factor;
0186:         pattern_(2, 0) = CConstellation(-3.0f*scale,-1.0f*scale) ^ factor;
0187:         pattern_(2, 1) = CConstellation( 3.0f*scale,-1.0f*scale) ^ factor;
0188:         pattern_(2, 2) = CConstellation(-1.0f*scale,-1.0f*scale) ^ factor;
0189:         pattern_(2, 3) = CConstellation( 1.0f*scale,-1.0f*scale) ^ factor;
0190:         pattern_(2, 4) = CConstellation(-3.0f*scale, 1.0f*scale) ^ factor;
0191:         pattern_(2, 5) = CConstellation( 3.0f*scale, 1.0f*scale) ^ factor;
0192:         pattern_(2, 6) = CConstellation(-1.0f*scale, 1.0f*scale) ^ factor;
0193:         pattern_(2, 7) = CConstellation( 1.0f*scale, 1.0f*scale) ^ factor;
0194:         pattern_(2, 8) = CConstellation(-3.0f*scale,-3.0f*scale) ^ factor;
0195:         pattern_(2, 9) = CConstellation( 3.0f*scale,-3.0f*scale) ^ factor;
0196:         pattern_(2,10) = CConstellation(-1.0f*scale,-3.0f*scale) ^ factor;
0197:         pattern_(2,11) = CConstellation( 1.0f*scale,-3.0f*scale) ^ factor;
0198:         pattern_(2,12) = CConstellation(-3.0f*scale, 3.0f*scale) ^ factor;
0199:         pattern_(2,13) = CConstellation( 3.0f*scale, 3.0f*scale) ^ factor;
0200:         pattern_(2,14) = CConstellation(-1.0f*scale, 3.0f*scale) ^ factor;
0201:         pattern_(2,15) = CConstellation( 1.0f*scale, 3.0f*scale) ^ factor;
0202:         pattern_(3, 0) = CConstellation(-1.0f*scale, 1.0f*scale) ^ factor;
0203:         pattern_(3, 1) = CConstellation(-3.0f*scale, 1.0f*scale) ^ factor;
0204:         pattern_(3, 2) = CConstellation( 1.0f*scale, 1.0f*scale) ^ factor;
0205:         pattern_(3, 3) = CConstellation( 3.0f*scale, 1.0f*scale) ^ factor;
0206:         pattern_(3, 4) = CConstellation(-1.0f*scale, 3.0f*scale) ^ factor;
0207:         pattern_(3, 5) = CConstellation(-3.0f*scale, 3.0f*scale) ^ factor;
0208:         pattern_(3, 6) = CConstellation( 1.0f*scale, 3.0f*scale) ^ factor;
0209:         pattern_(3, 7) = CConstellation( 3.0f*scale, 3.0f*scale) ^ factor;
0210:         pattern_(3, 8) = CConstellation(-1.0f*scale,-1.0f*scale) ^ factor;
0211:         pattern_(3, 9) = CConstellation(-3.0f*scale,-1.0f*scale) ^ factor;
0212:         pattern_(3,10) = CConstellation( 1.0f*scale,-1.0f*scale) ^ factor;
0213:         pattern_(3,11) = CConstellation( 3.0f*scale,-1.0f*scale) ^ factor;
0214:         pattern_(3,12) = CConstellation(-1.0f*scale,-3.0f*scale) ^ factor;
0215:         pattern_(3,13) = CConstellation(-3.0f*scale,-3.0f*scale) ^ factor;
0216:         pattern_(3,14) = CConstellation( 1.0f*scale,-3.0f*scale) ^ factor;
0217:         pattern_(3,15) = CConstellation( 3.0f*scale,-3.0f*scale) ^ factor;
0218:       }
0219:       virtual ~CMappingPattern () { }
0220:       iq::CConstellation& operator() ( unsigned long index, size_t retransmission = 0 )
0221:       {
0222:         return pattern_(retransmission, index);
0223:       }
0224:       iq::CConstellation operator() ( unsigned long index, size_t retransmission = 0 ) const
0225:       {
0226:         return pattern_(retransmission, index);
0227:       }
0228:       iq::CConstellation at ( unsigned long index, size_t retransmission = 0 ) const
0229:       {
0230:         return pattern_(retransmission, index);
0231:       }
0232:     };
0233:
0234:     template < typename ModulationMethodType >
0235:     class CModulator { };
0236:
0237:     template < >
0238:     class CModulator < modulation_type < MODULATION_BPSK > >
0239:     {
0240:     public:
0241:       typedef modulation_type < MODULATION_BPSK > modulation_method_type;
0242:     private:
0243:       CMappingPattern < modulation_method_type > mapping_;
0244:     public:
0245:       CModulator () { }
0246:       virtual ~CModulator () { }
0247:       template < typename SequenceType1, typename SequenceType2 >
0248:       void doModulation ( const SequenceType1& s1, SequenceType2& s2 ) const
0249:       {
0250:         for ( size_t i = 0; i < SequenceType1::length; ++i )
0251:           for ( size_t j = 0; j < 32; ++j )
0252:             s2[32*i+j] = mapping_[ ( s1.bits_[i] >> ( 32 - j - 1 ) ) & 0x1 ];
0253:       }
0254:     };
0255:
0256:     template < >
0257:     class CModulator < modulation_type < MODULATION_QPSK> >
0258:     {
0259:     public:
0260:       typedef modulation_type < MODULATION_QPSK > modulation_method_type;
0261:     private:
0262:       CMappingPattern < modulation_method_type > mapping_;
0263:     public:
0264:       CModulator () { }
0265:       virtual ~CModulator () { }
0266:       template < typename SequenceType1, typename SequenceType2 >
0267:       void doModulation ( const SequenceType1& s1, SequenceType2& s2 ) const
0268:       {
0269:         for ( size_t i = 0; i < SequenceType1::length; ++i )
0270:           for ( size_t j = 0; j < 16; ++j )
0271:             s2[16*i+j] = mapping_[ ( s1.bits_[i] >> ( 32 - 2*j - 2 ) ) & 0x3 ];
0272:       }
0273:     };
0274:
0275:     template < >
0276:     class CModulator < modulation_type < MODULATION_16QAM > >
0277:     {
0278:     public:
0279:       typedef modulation_type < MODULATION_16QAM > modulation_method_type;
0280:     private:
0281:       CMappingPattern < modulation_method_type > mapping_;
0282:     public:
0283:       CModulator ( ) { }
0284:       virtual ~CModulator ( ) { }
0285:       template < typename SequenceType1, typename SequenceType2 >
0286:       void doModulation ( const SequenceType1& s1, SequenceType2& s2, int retrans = 0 ) const
0287:       {
0288:         for ( size_t i = 0; i < SequenceType1::length; ++i )
0289:           for ( size_t j = 0; j < 8; ++j )
0290:             s2[8*i+j] = mapping_(( s1.bits_[i] >> ( 32 - 4*j - 4 ) ) & 0xFUL, retrans );
0291:       }
0292:     };
0293:
0294:     //  Definitons of the demodulator or detector
0295:     struct SoftDecisionInformation
0296:     {
0297:       float prob_[2];
0298:       SoftDecisionInformation ()
0299:       {
0300:         for ( size_t i = 0; i < 2; ++i )
0301:           prob_[i] = 0.0f;
0302:       }
0303:       SoftDecisionInformation ( const SoftDecisionInformation& that )
0304:       {
0305:         for ( size_t i = 0; i < 2; ++i )
0306:           prob_[i] = that.prob_[i];
0307:       }
0308:       virtual ~SoftDecisionInformation () { }
0309:       SoftDecisionInformation& operator= ( const SoftDecisionInformation& that )
0310:       {
0311:         for ( size_t i = 0; i < 2; ++i )
0312:           prob_[i] = that.prob_[i];
0313:         return *this;
0314:       }
0315:       SoftDecisionInformation& operator+= ( const SoftDecisionInformation& rhs )
0316:       {
0317:         for ( size_t i = 0; i < 2; ++i )
0318:           prob_[i] += rhs.prob_[i];
0319:         return *this;
```

```
0320:       }
0321:       SoftDecisionInformation& operator-= ( const SoftDecisionInformation& rhs )
0322:       {
0323:         for ( size_t i = 0; i < 2; ++i )
0324:           prob_[i] -= rhs.prob_[i];
0325:         return *this;
0326:       }
0327:       SoftDecisionInformation& operator*= ( const SoftDecisionInformation& rhs )
0328:       {
0329:         for ( size_t i = 0; i < 2; ++i )
0330:           prob_[i] *= rhs.prob_[i];
0331:         return *this;
0332:       }
0333:       SoftDecisionInformation& operator/= ( const SoftDecisionInformation& rhs )
0334:       {
0335:         for ( size_t i = 0; i < 2; ++i )
0336:           prob_[i] /= rhs.prob_[i];
0337:         return *this;
0338:       }
0339:       void clear ()
0340:       {
0341:         for ( size_t i = 0; i < 2; ++i )
0342:           prob_[i] = 0.0f;
0343:       }
0344:     };
0345:
0346:     SoftDecisionInformation operator+ (const SoftDecisionInformation& lhs, const SoftDecisionInformation& rhs )
0347:     {
0348:       SoftDecisionInformation ret;
0349:       for ( size_t i = 0; i < 2; ++i )
0350:         ret.prob_[i] = lhs.prob_[i] + rhs.prob_[i];
0351:       return ret;
0352:     }
0353:     SoftDecisionInformation operator- ( const SoftDecisionInformation& lhs, const SoftDecisionInformation& rhs )
0354:     {
0355:       SoftDecisionInformation ret;
0356:       for ( size_t i = 0; i < 2; ++i )
0357:         ret.prob_[i] = lhs.prob_[i] - rhs.prob_[i];
0358:       return ret;
0359:     }
0360:     SoftDecisionInformation operator* ( const SoftDecisionInformation& lhs, const SoftDecisionInformation& rhs )
0361:     {
0362:       SoftDecisionInformation ret;
0363:       for ( size_t i = 0; i < 2; ++i )
0364:         ret.prob_[i] = lhs.prob_[i] * rhs.prob_[i];
0365:       return ret;
0366:     }
0367:     SoftDecisionInformation operator/ ( const SoftDecisionInformation& lhs, const SoftDecisionInformation& rhs )
0368:     {
0369:       SoftDecisionInformation ret;
0370:       for ( size_t i = 0; i < 2; ++i )
0371:         ret.prob_[i] = lhs.prob_[i] / rhs.prob_[i];
0372:       return ret;
0373:     }
0374:     SoftDecisionInformation operator/ ( const SoftDecisionInformation& lhs, float rhs )
0375:     {
0376:       SoftDecisionInformation ret;
0377:       for ( size_t i = 0; i < 2; ++i )
0378:         ret.prob_[i] = lhs.prob_[i] / rhs;
0379:       return ret;
0380:     }
0381:
0382:     //  Typedefs of struct SoftDecisionInformation
0383:     typedef struct SoftDecisionInformation SOFT_DECISION_INFORMATION, *LPSOFT_DECISION_INFORMATION;
0384:
0385:     //  I define the soft decision information sequence class
0386:     template < size_t SIZE >
0387:     class CSoftDecisionInformationSequence
0388:     {
0389:     public:
0390:       static const size_t size = SIZE;
0391:       typedef std::vector < SOFT_DECISION_INFORMATION > sequence_type;
0392:       typedef sequence_type::iterator sequence_iterator;
0393:       typedef sequence_type::const_iterator const_sequence_iterator;
0394:     private:
0395:       sequence_type sequence_;
0396:     public:
0397:       CSoftDecisionInformationSequence ()
0398:         : sequence_ ( SIZE, SOFT_DECISION_INFORMATION() ) { }
0399:       explicit CSoftDecisionInformationSequence ( const SOFT_DECISION_INFORMATION& rhs )
0400:       {
0401:         using namespace boost::lambda;
0402:         for_each ( sequence_.begin(), sequence_.end(), _1 = rhs );
0403:       }
0404:       CSoftDecisionInformationSequence ( const CSoftDecisionInformationSequence < SIZE >& that )
0405:       {
0406:         sequence_.assign ( that.sequence_.begin(), that.sequence_.end() );
0407:       }
0408:       virtual ~CSoftDecisionInformationSequence () { }
0409:       CSoftDecisionInformationSequence < SIZE > operator= (const CSoftDecisionInformationSequence < SIZE >& that)
0410:       {
0411:         sequence_.assign ( that.sequence_.begin(), that.sequence_.end() );
0412:         return *this;
0413:       }
0414:       CSoftDecisionInformationSequence < SIZE > operator= ( const SOFT_DECISION_INFORMATION& rhs )
0415:       {
0416:         using namespace boost::lambda;
0417:         for_each ( sequence_.begin(), sequence_.end(), _1 = rhs );
0418:         return *this;
0419:       }
0420:       //  For expression template
0421:       template < typename T >
0422:       CSoftDecisionInformationSequence < SIZE > operator= ( const T& rhs )
0423:       {
0424:         for ( size_t i = 0; i < SIZE; ++i )
0425:           sequence_[i] = rhs[i];
0426:         return *this;
0427:       }
0428:       CSoftDecisionInformationSequence < SIZE >& operator+= (const CSoftDecisionInformationSequence<SIZE>& rhs )
0429:       {
0430:         for ( size_t i = 0; i < SIZE; ++i )
0431:           sequence_[i] += rhs.sequence_[i];
0432:         return *this;
0433:       }
0434:       CSoftDecisionInformationSequence < SIZE >& operator-= (const CSoftDecisionInformationSequence<SIZE>& rhs )
0435:       {
0436:         for ( size_t i = 0; i < SIZE; ++i )
0437:           sequence_[i] -= rhs.sequence_[i];
0438:         return *this;
0439:       }
0440:       CSoftDecisionInformationSequence < SIZE >& operator*= (const CSoftDecisionInformationSequence<SIZE>& rhs )
0441:       {
0442:         for ( size_t i = 0; i < SIZE; ++i )
0443:           sequence_[i] *= rhs.sequence_[i];
0444:         return *this;
0445:       }
0446:       CSoftDecisionInformationSequence < SIZE >& operator/= (const CSoftDecisionInformationSequence<SIZE>& rhs )
0447:       {
0448:         for ( size_t i = 0; i < SIZE; ++i )
0449:           sequence_[i] /= rhs.sequence_[i];
0450:         return *this;
0451:       }
0452:       SOFT_DECISION_INFORMATION& operator[] ( size_t index )
0453:       {
0454:         return sequence_[index];
0455:       }
0456:       SOFT_DECISION_INFORMATION operator[] ( size_t index ) const
0457:       {
0458:         return sequence_[index];
0459:       }
0460:       SOFT_DECISION_INFORMATION at ( size_t index ) const
0461:       {
0462:         return sequence_[index];
0463:       }
0464:       void clear ()
0465:       {
```

```
0466:       for ( sequence_iterator i = sequence_.begin(); i != sequence_.end(); ++i )
0467:         i->clear();
0468:     }
0469:   };
0470:
0471:   //  I've implemented the expression template with two terms operators
0472:   template < typename LeftType, typename OperatorType, typename RightType >
0473:   class soft_decision_information_sequence_expression
0474:   {
0475:   private:
0476:     const LeftType& lhs_;
0477:     const RightType& rhs_;
0478:   public:
0479:     soft_decision_information_sequence_expression ( const LeftType& lhs, const RightType& rhs )
0480:       : lhs_ ( lhs ), rhs_ ( rhs ) { }
0481:     SOFT_DECISION_INFORMATION operator[] ( size_t index ) const
0482:     {
0483:       return OperatorType::apply ( lhs_[index], rhs_[index] );
0484:     }
0485:   };
0486:   struct plus
0487:   {
0488:     static SOFT_DECISION_INFORMATION apply (
0489:       const SOFT_DECISION_INFORMATION& lhs, const SOFT_DECISION_INFORMATION& rhs )
0490:     {
0491:       return lhs + rhs;
0492:     }
0493:   };
0494:   struct minus
0495:   {
0496:     static SOFT_DECISION_INFORMATION apply (
0497:       const SOFT_DECISION_INFORMATION& lhs, const SOFT_DECISION_INFORMATION& rhs )
0498:     {
0499:       return lhs - rhs;
0500:     }
0501:   };
0502:   struct multiply
0503:   {
0504:     static SOFT_DECISION_INFORMATION apply (
0505:       const SOFT_DECISION_INFORMATION& lhs, const SOFT_DECISION_INFORMATION& rhs )
0506:     {
0507:       return lhs * rhs;
0508:     }
0509:   };
0510:   struct division
0511:   {
0512:     static SOFT_DECISION_INFORMATION apply (
0513:       const SOFT_DECISION_INFORMATION& lhs, const SOFT_DECISION_INFORMATION& rhs )
0514:     {
0515:       return lhs / rhs;
0516:     }
0517:   };
0518:
0519:   template < typename LeftType, typename RightType >
0520:   soft_decision_information_sequence_expression < LeftType, plus, RightType >
0521:   operator+ ( const LeftType& lhs, const RightType& rhs )
0522:   {
0523:     return soft_decision_information_sequence_expression < LeftType, plus, RightType > ( lhs, rhs );
0524:   }
0525:   template < typename LeftType, typename RightType >
0526:   soft_decision_information_sequence_expression < LeftType, minus, RightType >
0527:   operator- ( const LeftType& lhs, const RightType& rhs )
0528:   {
0529:     return soft_decision_information_sequence_expression < LeftType, minus, RightType > ( lhs, rhs );
0530:   }
0531:   template < typename LeftType, typename RightType >
0532:   soft_decision_information_sequence_expression < LeftType, multiply, RightType >
0533:   operator* ( const LeftType& lhs, const RightType& rhs )
0534:   {
0535:     return soft_decision_information_sequence_expression < LeftType, multiply, RightType > ( lhs, rhs );
0536:   }
0537:
0538:   template < typename LeftType, typename RightType >
0539:   soft_decision_information_sequence_expression < LeftType, division, RightType >
0540:   operator/ ( const LeftType& lhs, const RightType& rhs )
0541:   {
0542:     return soft_decision_information_sequence_expression < LeftType, division, RightType > ( lhs, rhs );
0543:   }
0544:
0545:   //  I define the symbols class by using detecting
0546:   template < typename ModulationMethodType >
0547:   class CSymbols
0548:   {
0549:   public:
0550:     typedef typename ModulationMethodType  modulation_method_type;
0551:   private:
0552:     CMappingPattern < ModulationMethodType > mapping_;
0553:     std::vector < float > dis_;
0554:   protected:
0555:     float oexp ( float rhs ) const
0556:     {
0557:       double ret = exp ( static_cast < double > ( rhs ) );
0558:       return ( ret < 1.0e-30 ) ? 1.0e-30f : static_cast < float > ( ret );
0559:     }
0560:     float oln ( float rhs ) const
0561:     {
0562:       double ret = log ( static_cast < double > (rhs ) );
0563:       return ( ret < 1e-30 ) ? 1.0e-30f : static_cast < float > ( ret );
0564:     }
0565:   public:
0566:     CSymbols ( )
0567:     {
0568:       using namespace boost::lambda;
0569:       dis_.resize ( ModulationMethodType::symbols );
0570:       for_each ( dis_.begin(), dis_.end(), _1 = 0.0f );
0571:     }
0572:     CSymbols ( const CSymbols& that )
0573:     {
0574:       dis_.assign ( that.dist_.begin(), that.dist_.end() );
0575:     }
0576:     virtual ~CSymbols ( ) { }
0577:     CSymbols& operator= ( const CSymbols& that )
0578:     {
0579:       dis_.assign ( that.dis_.begin(), that.dis_.end() );
0580:       return *this;
0581:     }
0582:     float& operator[] ( size_t index )
0583:     {
0584:       return dis_[index];
0585:     }
0586:     float operator[] ( size_t index ) const
0587:     {
0588:       return dis_[index];
0589:     }
0590:     CSymbols < ModulationMethodType >& operator+= ( const CSymbols < ModulationMethodType >& rhs )
0591:     {
0592:       for ( size_t i = 0; i < ModulationMethodType::symbols; ++i )
0593:         dis_[i] += rhs.dis_[i];
0594:       return *this;
0595:     }
0596:     CSymbols < ModulationMethodType >& operator/= ( float rhs )
0597:     {
0598:       for_each ( dis_.begin(), dis_.end(), boost::lambda::_1 /= rhs );
0599:       return *this;
0600:     }
0601:     void distance ( const iq::CConstellation& rhs )
0602:     {
0603:       for ( size_t i = 0; i < ModulationMethodType::symbols; ++i )
0604:         dis_[i] += oexp ( - mapping_[i].distance(rhs) );
0605:     }
0606:     void distance ( int retrans, const iq::CConstellation& rhs )
0607:     {
```

```
0608:         for ( size_t i = 0; i < ModulationMethodType::symbols; ++i )
0609:           dis_[i] = oexp ( - mapping_.at(i,retrans).distance(rhs) );
0610:       }
0611:
0612:       //  Definitons of the hard decisions, argument value is position of byte, e.g., S = ( s3, s2, s1, s0 ).
0613:       template < typename RetType >
0614:       RetType hard_decision ( int bit ) const
0615:       {
0616:         RetType ret = max_element ( dis_.begin(), dis_.end() ) - dis_.begin();
0617:         return ( ret >> bit ) & 0x1;
0618:       }
0619:       SoftDecisionInformation soft_decision ( int bit ) const
0620:       {
0621:         SoftDecisionInformation ret;
0622:         for ( size_t i = 0; i < ModulationMethodType::symbols; ++i )
0623:         {
0624:           unsigned pos = ( i >> bit ) & 0x1;
0625:           ret.prob_[ pos ? 0 : 1 ] += dis_[i];
0626:         }
0627:         return ret;
0628:       }
0629:       float soft_decision_llr ( int bit ) const
0630:       {
0631:         SoftDecisionInformation ret = soft_decision ( bit );
0632:         return oln(ret.prob_[0]) - oln(ret.prob_[1]);
0633:       }
0634:       void print_distance ( std::ostream& out ) const
0635:       {
0636:         using namespace boost::lambda;
0637:         for_each ( dis_.begin(), dis_.end(), ( out << _1 << '\n' ) );
0638:       }
0639:     };
0640:
0641:     //  In this section, we define the demodulator class.
0642:     template < typename ModulationMethodType >
0643:     class CDemodulator { };
0644:     //  For BPSK.
0645:     template < >
0646:     class CDemodulator < modulation_type < MODULATION_BPSK > >
0647:     {
0648:     public:
0649:       typedef modulation_type < MODULATION_BPSK > modulation_method_type;
0650:     private:
0651:       CSymbols < modulation_method_type > symbols_;
0652:     public:
0653:       CDemodulator ( ) { }
0654:       virtual ~CDemodulator ( ) { }
0655:       template < typename SequenceType1, typename SequenceType2 >
0656:       void hard_decision ( const SequenceType1& recv, SequenceType2& det )
0657:       {
0658:         for ( size_t i = 0; i < SequenceType2::length; ++i )
0659:           for ( size_t j = 0; j < SequenceType2::block_size; ++j )
0660:           {
0661:             symbols_.distance ( recv[SequenceType2::block_size*i+j] );
0662:             det(i,j) = symbols.hard_decision(0);
0663:           }
0664:       }
0665:       template < typename SequenceType1, typename SequenceType2 >
0666:       void soft_decision ( const SequenceType1& recv, SequenceType2& det )
0667:       {
0668:         for ( size_t i = 0; i < SequenceType1::size; ++i )
0669:         {
0670:           symbols_.distance ( recv[i] );
0671:           det[i] = symbols_.soft_decision(0);
0672:         }
0673:       }
0674:     };
0675:     //  For QPSK.
0676:     template < >
0677:     class CDemodulator < modulation_type < MODULATION_QPSK > >
0678:     {
0679:     public:
0680:       typedef modulation_type < MODULATION_QPSK > modulation_method_type;
0681:     private:
0682:       CSymbols < modulation_method_type > symbols_;
0683:     public:
0684:       CDemodulator ( ) { }
0685:       virtual ~CDemodulator ( ) { }
0686:       template < typename SequenceType1, typename SequenceType2 >
0687:       void hard_decision ( const SequenceType1& recv, SequenceType2& det )
0688:       {
0689:         for ( size_t i = 0; i < SequenceType1::size; ++i )
0690:         {
0691:           symbols_.distance ( recv[i] );
0692:           for ( size_t j = 0; j < 2; ++j )
0693:             det( (i+j)/SequenceType2::block_size, (i+j)%SequenceType2::block_size ) = symbols.hard_decision(j);
0694:         }
0695:       }
0696:       template < typename SequenceType1, typename SequenceType2 >
0697:       void soft_decision ( const SequenceType1& recv, SequenceType2& det )
0698:       {
0699:         for ( size_t i = 0; i < SequenceType1::size; ++i )
0700:         {
0701:           symbols_.distance ( recv[i] );
0702:           for ( size_t j = 0; j < 2; ++j )
0703:             det[2*i+j] = symbols_.soft_decision(2-j-1);
0704:         }
0705:       }
0706:     };
0707:     //  For 16-QAM.
0708:     template < >
0709:     class CDemodulator < modulation_type < MODULATION_16QAM > >
0710:     {
0711:     public:
0712:       typedef modulation_type < MODULATION_16QAM > modulation_method_type;
0713:     private:
0714:       CSymbols < modulation_method_type > symbols_;
0715:     public:
0716:       CDemodulator ( ) { }
0717:       virtual ~CDemodulator ( ) { }
0718:       template < typename SequenceType1, typename SequenceType2 >
0719:       void hard_decision ( const SequenceType1& recv, SequenceType2& det, int retrans = 0 )
0720:       {
0721:         for ( size_t i = 0; i < SequenceType1::size; ++i )
0722:         {
0723:           symbols_.distance ( retrans, recv[i] );
0724:           for ( size_t j = 0; j < 4; ++j )
0725:             det(4*i+j, symbols_.hard_decision < SequenceType2::bits_type > (3-j) );
0726:         }
0727:       }
0728:       template < typename SequenceType1, typename SequenceType2 >
0729:       void soft_decision (const SequenceType1& recv, SequenceType2& det, int retrans = 0 )
0730:       {
0731:         for ( size_t i = 0; i < SequenceType1::size; ++i )
0732:         {
0733:           symbols_.distance ( retrans, recv[i] );
0734:           for ( size_t j = 0; j < 4; ++j )
0735:             det[4*i+j] = symbols_.soft_decision(3-j);
0736:         }
0737:       }
0738:       template < typename SequenceType1, typename SequenceType2 >
0739:       void hard_decision_nsmd ( const SequenceType1& recv0, const SequenceType1& recv1, SequenceType2& det )
0740:       {
0741:         for ( size_t i = 0; i < SequenceType1::size; ++i )
0742:         {
0743:           CSymbols < modulation_method_type > sym_0, sym_1;
0744:           sym_0.distance ( 0, recv0[i] );
0745:           sym_1.distance ( 0, recv1[i] );
0746:           symbols_ = sym_0;
0747:           symbols_ += sym_1;
0748:           for ( size_t j = 0; j < 4; ++j )
0749:             det(4*i+j, symbols_.hard_decision < SequenceType2::bits_type > (3-j) );
```

```
0750:         }
0751:       }
0752:
0753:       template < typename SequenceType1, typename SequenceType2 >
0754:       void hard_decision_nsmd ( const SequenceType1& recv0, const SequenceType1& recv1,
0755:         const SequenceType1& recv2, SequenceType2& det )
0756:       {
0757:         for ( size_t i = 0; i < SequenceType1::size; ++i )
0758:         {
0759:           CSymbols < modulation_method_type > sym_0, sym_1, sym_2;
0760:           sym_0.distance ( 0, recv0[i] );
0761:           sym_1.distance ( 0, recv1[i] );
0762:           sym_2.distance ( 0, recv2[i] );
0763:           symbols_ = sym_0;
0764:           symbols_ += sym_1;
0765:           symbols_ += sym_2;
0766:           for ( size_t j = 0; j < 4; ++j )
0767:             det(4*i+j, symbols_.hard_decision < SequenceType2::bits_type > (3-j) );
0768:         }
0769:       }
0770:       template < typename SequenceType1, typename SequenceType2 >
0771:       void hard_decision_nsmd ( const SequenceType1& recv0, const SequenceType1& recv1,
0772:         const SequenceType1& recv2, const SequenceType1& recv3, SequenceType2& det )
0773:       {
0774:         for ( size_t i = 0; i < SequenceType1::size; ++i )
0775:         {
0776:           CSymbols < modulation_method_type > sym_0, sym_1, sym_2, sym_3;
0777:           sym_0.distance ( 0, recv0[i] );
0778:           sym_1.distance ( 0, recv1[i] );
0779:           sym_2.distance ( 0, recv2[i] );
0780:           sym_3.distance ( 0, recv3[i] );
0781:           symbols_ = sym_0;
0782:           symbols_ += sym_1;
0783:           symbols_ += sym_2;
0784:           symbols_ += sym_3;
0785:           for ( size_t j = 0; j < 4; ++j )
0786:             det(4*i+j, symbols_.hard_decision < SequenceType2::bits_type > (3-j) );
0787:         }
0788:       }
0789:       template < typename SequenceType1, typename SequenceType2 >
0790:       void hard_decision_smd ( const SequenceType1& recv0, const SequenceType1& recv1, SequenceType2& det )
0791:       {
0792:         for ( size_t i = 0; i < SequenceType1::size; ++i )
0793:         {
0794:           CSymbols < modulation_method_type > sym_0, sym_1;
0795:           sym_0.distance ( 0, recv0[i] );
0796:           sym_1.distance ( 1, recv1[i] );
0797:           symbols_ = sym_0;
0798:           symbols_ += sym_1;
0799:           for ( size_t j = 0; j < 4; ++j )
0800:             det(4*i+j, symbols_.hard_decision < SequenceType2::bits_type > (3-j) );
0801:         }
0802:       }
0803:       template < typename SequenceType1, typename SequenceType2 >
0804:       void hard_decision_smd ( const SequenceType1& recv0, const SequenceType1& recv1,
0805:         const SequenceType1& recv2, SequenceType2& det )
0806:       {
0807:         for ( size_t i = 0; i < SequenceType1::size; ++i )
0808:         {
0809:           CSymbols < modulation_method_type > sym_0, sym_1, sym_2;
0810:           sym_0.distance ( 0, recv0[i] );
0811:           sym_1.distance ( 1, recv1[i] );
0812:           sym_2.distance ( 2, recv2[i] );
0813:           symbols_ = sym_0;
0814:           symbols_ += sym_1;
0815:           symbols_ += sym_2;
0816:           for ( size_t j = 0; j < 4; ++j )
0817:             det(4*i+j, symbols_.hard_decision < SequenceType2::bits_type > (3-j) );
0818:         }
0819:       }
0820:       template < typename SequenceType1, typename SequenceType2 >
0821:       void hard_decision_smd ( const SequenceType1& recv0, const SequenceType1& recv1,
0822:         const SequenceType1& recv2, const SequenceType1& recv3, SequenceType2& det )
0823:       {
0824:         for ( size_t i = 0; i < SequenceType1::size; ++i )
0825:         {
0826:           CSymbols < modulation_method_type > sym_0, sym_1, sym_2, sym_3;
0827:           sym_0.distance ( 0, recv0[i] );
0828:           sym_1.distance ( 1, recv1[i] );
0829:           sym_2.distance ( 2, recv2[i] );
0830:           sym_3.distance ( 3, recv3[i] );
0831:           symbols_ = sym_0;
0832:           symbols_ += sym_1;
0833:           symbols_ += sym_2;
0834:           symbols_ += sym_3;
0835:           for ( size_t j = 0; j < 4; ++j )
0836:             det(4*i+j, symbols_.hard_decision < SequenceType2::bits_type > (3-j) );
0837:         }
0838:       }
0839:
0840:     };
0841:   } //  End of the namespace dsp
```

A.4 無線チャネル信号処理

```
0001:  //  channel.hpp:
0002:  //     Definitions of the communications channel class.
0003:  ////////////////////////////////////////////////////////////////////
0004:
0005:  #pragma once
0006:  #include "random.hpp"
0007:  #include "clock.hpp"
0008:  #include "iq.hpp"
0009:  #include <math.h>
0010:  #include <time.h>
0011:  #include <vector>
0012:  #include <algorithm>
0013:  #include <boost/lambda/lambda.hpp>
0014:
0015:  namespace channel {
0016:    //  Circular constant.
0017:    static const float PI = 3.141592654f;
0018:
0019:    //  AWGN channel
0020:    template < typename SequenceType >
0021:    void awgn ( float Eb, random::GAUSSIAN_GENERATOR& ggen, SequenceType& signal, SequenceType& noise )
0022:    {
0023:      using namespace iq;
0024:      float REb = sqrt ( Eb );
0025:      for ( size_t i = 0; i < SequenceType::size; ++i )
0026:        noize[i] += iq::CConstellation ( REb * ggen(), REb * ggen() );
0027:      signal = signal + noize;
0028:    }
0029:
0030:    //  Quasi-static Rayleigh fading channel
0031:    template < typename SequenceType >
0032:    void quasi_static_rayleigh_fading(random::GAUSSIAN_GENERATOR& ggen,SequenceType& signal,SequenceType& noise )
0033:    {
0034:      using namespace iq;
0035:      for ( size_t i = 0; i < SequenceType::size; ++i )
0036:        noize[i] += iq::CConstellation ( ggen() * 0.707106781, ggen() * 0.707106781 );
0037:      signal ^= noize;
0038:    }
0039:
0040:    //  I define the any path under fading environment
0041:    class fading_path
0042:    {
0043:    private:
0044:      size_t n_;
0045:      std::vector < float > theta_;
0046:      std::vector < float > phi_;
0047:    public:
```

```
0048:       fading_path() { }
0049:       fading_path ( const fading_path& that )
0050:         : n_(that.n_), theta_(that.theta_), phi_(that.phi_) { }
0051:       virtual ~fading_path() { }
0052:       //  It is NECESSARY to FULFILL THIS FUNCTION when this class has been constructed.
0053:       void initialize ( random::UNIFORM_GENERATOR& ugen, int n = 10 )
0054:       {
0055:         using namespace boost::lambda;
0056:         n_ = n;
0057:         theta_.resize(n);
0058:         phi_.resize(n);
0059:         for_each ( theta_.begin(), theta_.end(), ( _1 = ugen() ) );
0060:         for_each ( phi_.begin(), phi_.end(), ( _1 = ugen() ) );
0061:       }
0062:       template < typename SequenceType >
0063:       void through ( CLOCK& clk, float fd, SequenceType& isig, SequenceType& rsig ) const
0064:       {
0065:         using namespace iq;
0066:         float fn = static_cast < float > ( n_ );
0067:         for ( size_t i = 0; i < SequenceType::size; ++i )
0068:         {
0069:           float real = 0.0f;
0070:           float imag = 0.0f;
0071:           for ( size_t j = 0; j < n_; ++j )
0072:             real += cos ( 2.0f*PI*fd*cos(theta_[j])*clk() + phi_[j] );
0073:           for ( size_t j = 0; j < n_; ++j )
0074:             real += sin ( 2.0f*PI*fd*cos(theta_[j])*clk() + phi_[j] );
0075:           rsig[i] = SequenceType::constellation_type ( real/fn, imag/fn );
0076:           ++clk;
0077:         }
0078:         isig *= rsig;
0079:       }
0080:     };
0081:
0082:     //  I define the wireless link considered the delay spread
0083:     template < size_t NUM >
0084:     struct rayleigh_link
0085:     {
0086:       std::vector < fading_path > path_;
0087:       rayleigh_link() { }
0088:       void initilalize ( random::UNIFORM_GENERATOR& ugen )
0089:       {
0090:         typedef std::vector < fading_path >::iterator ITR;
0091:         path_.resize ( NUM );
0092:         for ( ITR i = path_.begin(); i != path_.end(); ++i )
0093:           i->initialize ( ugen, 10 );
0094:       }
0095:     };
0096:
0097:     //  I define the parameters of delay spread, which is expressed as MICRO SECOND.
0098:     template < typename DelayType = float, typename AmplitudeType = float >
0099:     struct delay_spread_parameters_us
0100:     {
0101:       typedef DelayType delay_type;
0102:       typedef AmplitudeType amplitude_type;
0103:       DelayType delay_time_;  // [us]
0104:       AmplitudeType amplitude_;
0105:     };
0106:
0107:     //  Typedefs of the delay_spread_parameters_us structure.
0108:     typedef delay_spread_parameters_us < float, float >
0109:       DELAY_SPREAD_PARAMETERS_US, *LPDELAY_SPREAD_PARAMETERS_US;
0110:
0111:     //  We also define the parameters of delay spread, but it is DIFFERENT compared with the former implementation.
0112:     //  In this structure, we assumed that the time is expressed as DESECRATE TIME UNIT, called time position.
0113:     template < typename DesecrateDelayType = unsigned, typename AmplitudeType = float >
0114:     struct delay_spread_parameters_position
0115:     {
0116:       typedef DesecrateDelayType desecrate_delay_type;
0117:       typedef AmplitudeType amplitude_type;
0118:       DesecrateDelayType delay_unit_time_;  // [time unit]
0119:       AmplitudeType amplitude_;
0120:       template < typename DelayType >
0121:       void initialze(const struct delay_spread_parameters_us<DelayType, AmplitudeType>& params, const CLOCK& clk )
0122:       {
0123:         delay_unit_time_ = static_cast < DesecrateDelayType > ( params.delay_time_ / clk.resolution() );
0124:         amplitude_ = params.amplitude_;
0125:       }
0126:     };
0127:
0128:     //  Typedefs of the delay_spread_parameters_pos structure.
0129:     typedef struct delay_spread_parameters_position < unsigned , float >
0130:       DELAY_SPREAD_PARAMETERS_POSITION, *LPDELAY_SPREAD_PARAMETERS_POSITION;
0131:
0132:     //  We define the 3GPP channel model.
0133:     typedef struct rayleigh_link < 9 > LINK_3GPP, *LPLINK_3GPP;
0134:     //  We define the extended typical urban model environment here.
0135:     //  0[us](0):-1.0[dB], 50[us](3):-1.0[dB], 120[us](7):-1.0[dB], 200[us](12):0[dB], 230[us](14):0[dB],
0136:     //  500[us](30):0[dB], 1600[us](96):-3.0[dB], 2300[us](138):-5.0[dB], 5000[us](300):-7.0[dB]
0137:     static const DELAY_SPREAD_PARAMETERS_US DelaySpreadParamsFor3GPP_ETU[9] =
0138:       { { 0.891250938f, 0.0f }, { 0.891250938f, 50.0f }, { 0.891250938f, 120.0f }, { 1.0f, 200.0f  },
0139:         { 1.0f, 230.0f }, { 1.0f, 500.0f }, { 0.707945785f, 1600.0f }, { 0.562341325f, 2300.0f },
0140:         { 0.446683592f, 5000.0f }, };
0141:
0142:     template < typename SequenceType >
0143:     void typical_urban_model (CLOCK& clk, float fd, const LINK_3GPP& link, SequenceType& isig, SequenceType& rsig)
0144:     {
0145:       using namespace std;
0146:       using namespace iq;
0147:       vector < SequenceType > is ( 9, isig );
0148:       vector < SequenceType > rs ( 9, rsig );
0149:       CLOCK temp_clk;
0150:       for ( size_t i = 0; i < 9; ++i )
0151:       {
0152:         temp_clk = clk;
0153:         link.path_[i].through ( temp_clk, fd, is[i], rs[i] );
0154:       }
0155:       clk = temp_clk;
0156:       isig.clear();
0157:       rsig.clear();
0158:
0159:       DELAY_SPREAD_PARAMETERS_POSITION
0160:         delay_spread [ 9 ];
0161:       for ( size_t i = 0; i < 9; ++i )
0162:         delay_spread[i].initialze<DELAY_SPREAD_PARAMETERS_US::delay_type >(DelaySpreadParamsFor3GPP_ETU[i], clk );
0163:       for ( size_t i = 0; i < SequenceType::size; ++i )
0164:         for ( size_t j = 0; j < 9; ++j )
0165:           if ( i >= delay_spread[j].delay_unit_time_ )
0166:           {
0167:             isig[i] += ( is[j][i] ^ delay_spread[j].amplitude_ );
0168:             rsig[i] += ( rs[j][i] ^ delay_spread[j].amplitude_ );
0169:           }
0170:     }
0171:
0172:   } //  End of the namespace of channel.
```