第 5 部

ネットワーク管理

第 1 章

ネットワーク管理とは？

1.1 OSI フレームワーク

OSI では、ネットワーク管理の基本的なコンセプトおよびモデルに関する詳細なドキュメント「ISO DIS 7498-4: Information processing systems - Open Systems Interconnection, Basic Reference Model - Part4: OSI Managment Framework」の中で、ネットワーク管理アーキテクチャ、ネットワーク管理システムを語るときの主な要素として、以下の３つをあげている。

- ネットワーク管理を理解するためのモデル
- 管理対象を登録したり、指示したり、定義するための管理情報の共通構造
- リモート管理機構を実現するための関連プロトコルとサービス

1.2 アーキテクチャの概要

OSI のネットワーク管理は、２つのアプリケーションプロセスによって行なわれ、それぞれ、管理システム（管理ステーション）上にあるプロセスを “マネージャ(Manager)” とよび、管理されるシステム（ネットワークの要素）上にあるプロセスを “エージェント(Agent)” と呼ぶ。ネットワーク管理は、この２つのプロセスの共同作業によってネットワーク機器やネットワークを管理する上で有用なモニタの結果やコントロール情報をマネージャとエージェント間で交換することによって行なわれる。

マネージャからエージェントに要求する操作には、

- 読む (read a variable)
- 設定する (set a variable)
- 動作の実行を要求する (perform an action: such as self-test)

があり、エージェントからマネージャへは、

- イベント通知 (events or traps)

がある。

CMIP の仕様では、特にマネージャ／エージェントのような従属関係はなく、対称的な定義となっている。すなわち、マネージャ同士が相互に話すとか、ある時は、マネージャがエージェントの機能を果たすことも許されている。

1.3 管理モデル

ネットワーク管理には、いろいろなモデルがあるが、OSI では３つのモデルがよく使われる。

- **Organization Model:**
  “ドメイン” の考え方を導入するモデルで、“ドメイン” とは、ネットワーク管理を目的とした、インターネットやネットワークの管理区分である。“ドメイン” は、規模やセキュリティ、管理自治などを考えるのに利用される。

- **Functional Mode:**
  OSI 管理フレームワークでは５つの機能分類を考え、アプリケーションから見た時のネットワーク管理の問題を分類する場合に用いている。この５つの機能分類のことを “Specific Management Functional Areas(SMFAs) と呼んでいる。

  – 障害管理 (fault management)
  – 構成管理 (configuration management)
  – 性能管理 (performance management)
  – 課金管理 (accounting management)
  – セキュリティ管理 (security management)

- **Information Mode:**
  どのような情報を管理するかのモデルで、OSI では、例えば MIB(Management Information Base) のようなものを考えている。

1.4 Internet におけるネットワーク管理の歴史

初期のネットワークはシングルベンダのゲートウェイを用いて、エンド-エンドのプロトコルを使って作られていたので、マルチベンダの下でのネットワークマネージメントの必要はなかった。80 年代の半ばから、多くのネットワークが Internet に接続したたので、マルチベンダの下でのネットワークマネージメントの必要がでてきた。

1987 年 3 月にネットワークの研究に携わっていた人間のグループが会合をもち、研究および産業面でネットワーク管理が必要となっていることを指摘し、次の三つの活動が出てきた。

- SGMP (Simple Gateway Monitoring Protocol)
  4 人の技術者によって、2 カ月で作られたプロトコル。数カ月以内にいくつかの製造業者で実装された。この年の 8 月までは、控え目に受けとられていたが、もともと開発してたネットワークの外でも、非常に増加した。

- High-level Entity Management System(HEMS)
  上記会合以前に、研究プロジェクトとして始められていた。いくつか目新しい概念を持っており、開発しているサイト以外では日の目を見なかった。

- CMIP over TCP(CMOT)
  OSI でネットワークマネージメントがホットなトピックとなったのに応じて、それを TCP 上で用いるためにこの頃始まった。

1988 年 2 月、Internet Activities Board(IAB) は、これらのことを解決するための委員会を持った。その最初の Network Management Review グループの会合では、すぐに三つの分科会が作られ、それぞれ「正しい」解決をみた。そこでは、SGMP は Internet community でのネットワークマネージメントの短期的な解決のためのプロトコルであり、OSI ベースのアプローチは、将来的に長期的な解決に結び付くことを期待され、慎重な検討をする実験的なものととらえられた。
三つのワーキンググループとは以下のものである。

- MIB ワーキンググループ
  ネットワークマネージメントのために、SGMP の後継と CMOT の両方にちょうど良い一般的な枠組を作るための研究グループ。二つのドキュメントを作成した。一つは MIB に含まれている管理オブジェクトを名前づけし定義する Structure for Management Information(SMI) を規定し (RFC1065,1165)、もう一つは管理オブジェクトのリストの定義である MIB を規定している (RFC1066,1166)。

  SMI のドキュメントは一般的な構造と、management information のための機構の identification を明確に示す。management information には object information model の記述と management information を記述するために使う、一般的な型のセットを含む。MIB のドキュメントは、オブジェクトを明確にしている。

- SNMP ワーキンググループ
  別の研究グループは、SGMP を後継すべきプロトコルである SNMP のための研究グループ。1989 年 4 月、SNMP は「recommended protocol」に昇格して、事実上 TCP/IP ベースのインターネットでのネットワークマネージメントの標準となった。

- Netman ワーキンググループ
  ISO の CMIS/CMIP に基づくた長期的なネットワーク管理の研究グループ。TCP/IP 環境で ISO プロトコル使ってのネットワーク管理手法を記述した CMOT(CMIP over TCP/IP) のドキュメントを作成 (RFC1095)。

現在、SNMP(RFC1155)、MIB-I(RFC1156)、SNMP(RFC1157) は recommended Internet standard protocol であり、CMOT(RFC1095) は recommended draft Internet standard である。

1.5 SGMP

SGMP (A Simple Gateway Monitoring Protocol) は、ゲートウェイを監視するための情報を管理する、アプリケーションレイヤ上の簡単なプロトコルである。これは、即座にゲートウェイモニタの暫定的な応答が必要な場合に用いられるもので、より正確な情報を得るためのものではない。このプロトコルは、現在利用されている、SNMP(A Simple Network Management Protocol) の元になったプロトコルであり、RFC1028 で規定されている。現在では、historical protocol(obsolete protocol) であり、使わないことになっている。

1.5.1 プロトコル設計

このプロトコルは、ゲートウェイにモニタリングによる負荷をかけないようにするため、ゲートウェイ上に置かれる管理用の機能の数、複雑さを最小限にとどめている。
プロトコル設計の際のゴールは、

- プロトコルで規定する部分を最小限にする
- 拡張性を持たせる
- アーキテクチャに依存しない設計にする

の 3 つである。これらのゴールを目指すために、図 1.1のようなモデルを用い、以下のような規定を設けた。

- **規定 1**
  ゲートウェイ管理機能を、値の変更と取り出しの 2 つだけを行なうモデルにする。そして、リモートホスト上のプロトコルエンティティは、ゲートウェイ上のプロトコルエンティティと、ゲートウェイの状態を表す変数値の変更、取り出しだけを行なう。

  この結果として、管理機能は、

  1. assign a value (変数値の設定)

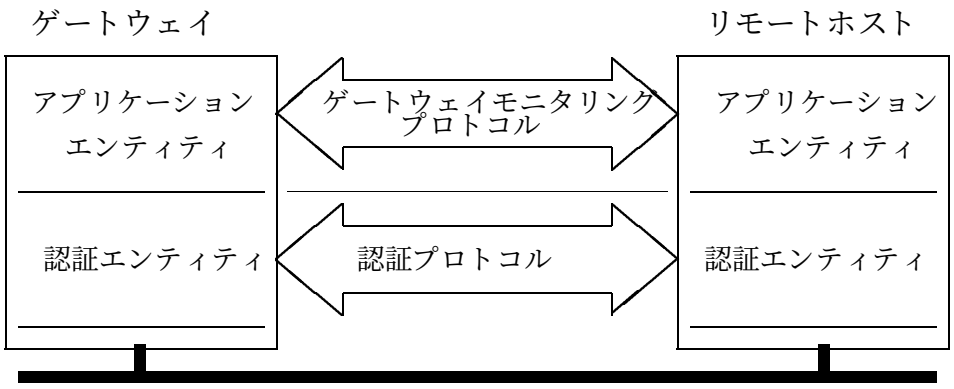


図 1.1: SGMP プロトコルモデル

2. retreive a value (変数値の取り出し)

の 2 つになる。また、管理用のコマンドの数や、セマンティクスは、このプロトコルではサポートしない。

- **規定 2**
  モニタリングプロトコル自身のサービスを提供する別のプロトコルレイヤに、認証の機能を加える。

  この結果として、モニタリングプロトコル自身の複雑さを減らすことができる。

- **規定 3**
  アプリケーションプロトコルエンティティは、下位の認証プロトコルを用いて、相手の身元を確認する。ゲートウェイの変数のアクセスコントロールのモデルは、アプリケーションエンティティと、認証プロトコルのセッションの間で行なう。

- **規定 4**
  プロトコルエンティティ間の情報交換の表現を、CCITT X.409 勧告の方式を利用する。

  この結果として、実装は難しくなるが、計算機のアーキテクチャに依存しない形になる。

- **規定 5**
  プロトコルの扱う変数を、整数 (int) か、文字列 (octet string) の 2 つの型にする。

- **規定 6**
  プロトコルメッセージの交換にデータグラムを用いる。

- **規定 7**
  単純、かつ拡張性を持たせるため、プロトコルの取り扱う変数の名前を、文字列 (octet string) にする。

この結果として、変数をランダムアクセスする際の、変数のグループ化を行ない、セマンティカルに逐次アクセスする機構を、単一の単純なメカニズムで提供することができる。また、名前空間を容易に拡張できる。

- **規定 8**
  プロトコルで出されるメッセージの数を最小限にとどめる。

  この結果として、ネットワーク管理機能の出すトラフィックを最小限におさえる。その代わりに、プロトコルにオーバヘッドがかかる。

1.5.2 ゲートウェイモニタリングプロトコル

ゲートウェイモニタリングプロトコルは、ゲートウェイの状態を表す変数値の変更、取り出しを行なうための、アプリケーションプロトコルである。このアプリケーションプロトコルエンティティ間の通信は、下位の認証プロトコルのサービスを使ったメッセージのやりとりによってなされる。従って、このメッセージは、下位の認証プロトコルの単一のメッセージで表現することができる。

ここでやりとりされるメッセージには、

1. Get Request Message Type、
2. Get Response Message Type
3. Trap Request Message Type
4. Set Request Message Type

がある。このタイプの識別は、CCITT X.409 勧告の方式に従って表現されたタグを用いて行なう。以下にこの四つのタイプを説明する。

- `The Get Request Message Type`
  Get Request タイプのメッセージは、ゲートウェイの変数値を取り出す時に、リモートホストから出される。受け取ったゲートウェイでは、Get Response タイプのメッセージを使って、要求したリモートホストにその変数の値を知らせる。

  メッセージには、メッセージのタイプと、

  - request_id、
  - error_status
  - error_index
  - var_op_list

  が含まれている。また、var_op_list は、変数名とその値の組の並びである。

- The Get Response Message Type
  Get Response タイプのメッセージは、Get Request タイプのメッセージを受け取ったゲートウェイが、要求を出したリモートホストに答える際に用いられる。

  メッセージには、メッセージのタイプと、

  - request_id
  - error_status
  - error_index
  - var_op_list

  が含まれている。また、var_op_list は、変数名とその値の組の並びである。

- The Trap Request Message Type
  Trap Request タイプのメッセージは、リモートホストからゲートウェイに送られるメッセージである。これを受け取ったゲートウェイは、指定された Trap が起こった時に、リモートホストにそれを知らせる。

  このタイプのメッセージには、以下のようなタイプがある。

  - The Cold Start Trap Type
  - The Warm Start Trap Type
  - The Link Failure Trap Type
  - The Authentication Failure Trap Type
  - The EGP Neighbor Loss Trap Type

- The Set Requst Message Type
  Set Request タイプのメッセージは、ゲートウェイの変数値を設定、変更する時に、リモートホストから出される。受け取ったゲートウェイでは、要求された変数の値を設定、変更し、Get Response タイプのメッセージを使って、リモートホストに要求を受け入れたことを知らせる。

1.5.3　認証プロトコル

認証プロトコルは、セッションレイヤのプロトコルである。このプロトコルを用いて、アプリケーションプロトコルのメッセージの認証を行なう。認証プロトコルエンティティ間の通信は、プロトコルメッセージの交換によってなされる。このメッセージは、一つの UDP データグラムで表現される。

認証プロトコルのセッションは、一方から他方にメッセージを送る、ハーフセッションの組で定義される。このハーフセッションは、

1. authentication function、
2. message interpretation function、
3. message representation function、

の 3 つの機能を持っている。

authentication 機能は、実際に認証を行なう。message interpretation 機能は、authentication 機能によって認証された認証プロトコルメッセージを、ユーザメッセージに変換する。逆に、meesage representation 機能は、ユーザメッセージを認証プロトコルメッセージに変換する。

このプロトコルには、

1. Data Request Message Type、

がある。

この Data Request タイプのメッセージは、以下の 4 つのフィールドを持つ。

- `Message Length Field`
  このフィールドは、メッセージの長さを表す。
- `Session ID Length Field`
  このフィールドは、メッセージのセッション ID フィールドの長さを octet で表す。
- `Session ID Field`
  このフィールドは、そのメッセージのプロトコルセッションの名前を表す。
- User Data Field
  このフィールドは、上位レイヤから渡されたメッセージを表す。上位レイヤから渡されたメッセージは、Session ID フィールドに書かれた名前のハーフセッション用の message representation 機能を用いてコード化される。

このメッセージを受け取った認証プロトコルエンティティは、Session ID フィールドに書かれた名前のハーフセッション用の authentication 機能を用いて相手をチェックし、これが認証されると、User Data フィールドを、message interpretation 機能でデコードした結果を上位レイヤ (モニタリングプロトコル) に渡す。認証されなかった場合には、そのメッセージは捨てられ、相手にその旨を知らせる。

1.5.4 変数名

アプリケーションプロトコルで扱われる変数には、2 通りの方法で名前が付けられる。一つは、16 進数で表されたもので、もう一つは、シンボリックな名前である (表 1.1)。

Get Request タイプのメッセージのセンマンティクスでは、関連する変数名は、アプリケーションプロトコルエンティティが識別する、辞書順に並べられた変数名の中に、連続して入っていなければならない。

numerical	*symbolic*
01 01 01	_GW_version_id
01 01 02	_GW_version_rev
01 02 01	_GW_cfg_nnets
01 03 01 03 01	_GW_net_if_type_net1
01 03 01 03 02	_GW_net_if_type_net2
01 03 01 04 01	_GW_net_if_speed_net1
01 03 01 04 02	_GW_net_if_speed_net1

表 1.1: 変数名の例

1.5.5 Required Variables

The _GW_version_id Variable
プロトコルエンティティの実装を表す octet string

The _GW_version_rev Variable
プロトコルエンティティの実装のリビジョンレベルを表す整数

The _GW_cfg_nnets Variable
ゲートウェイの持つ論理的ネットワークインタフェースの数を表す整数

The _GW_net_if_type Variable Class
ネットワークインタフェースのタイプを表す整数。このプロトコルでは、以下のタイプをサポートしている。

Value	Network Type
0	Unspecified
1	IEEE 802.3 MAC
2	IEEE 802.4 MAC
3	IEEE 802.5 MAC
4	Ethernet
5	ProNET-80
6	ProNET-10
7	FDDI
8	X.25
9	Point-to-Point Serial
10	Proprietary Point-to-Point Serial
11	ARPA 1822 HDH
12	ARPA 1822
13	AppleTalk
14	StarLAN

The _GW_net_if_speed Variable Class
ネットワークインタフェースのバンド幅 (bps) を表す整数

The _GW_net_if_in_pkts Variable Class
ネットワークインタフェースから受けとったパケット数を表す整数

The _GW_net_if_out_pkts Variable Class
ネットワークインタフェースから転送したパケット数を表す整数

The _GW_net_if_in_bytes Variable Class
ネットワークインタフェースから受けとった octet 数を表す整数

The _GW_net_if_out_bytes Variable Class
ネットワークインタフェースから転送た octet 数を表す整数

The _GW_net_if_in_error Variable Class
ネットワークインタフェース上で起こった受け取り error 数を表す整数

The _GW_net_if_out_error Variable Class
ネットワークインタフェース上で起こった転送 error 数を表す整数

The _GW_net_if_status Variable Class
ネットワークインタフェースの現在の状態を表す整数。状態は、以下の整数で表される

Value	Network Status
0	Interface Operating Normally
1	Interface Not Present
2	Interface Disabled
3	Interface Down
4	Interface Attempting Link

The _GW_pr_in_addr_value Variable Class
IP インタフェースの 32–bit インターネットアドレスを表す octet string

The _Gw_pr_in_addr_scope Variable Class
IP インタフェースのネットワークインタフェース名を表す octet string

The _GW_pr_in_egp_core Variable
インターネット core ゲートウェイ集合を考慮した、関連するゲートウェイを表す整数。この値が、0 でない場合は、関連するゲートウェイが、インターネット core の一部であることを示す

The _GW_pr_in_egp_as Variable
ゲートウェイの属する Autonomous システムを表す整数

The _GW_pr_in_egp_neighbor_addr Variable Class
EGP neighbor の 32–bit インターネットアドレスを表す octet string

The _GW_pr_in_egp_neighbor_state Variable Class

EGP neighbor を考慮したゲートウェイの EGP プロトコル状態を表す octet string。値には次のものがある。

- IDLE
- ACQUISITOIN
- DOWN
- UP
- CEASE

The _GW_pr_in_egp_errors Variable

EGP プロトコル error の数を表す整数

The _GW_pr_in_rt_gateway Variable Class

経路上の次のゲートウェイのインターネットアドレスを示す octet string

The _GW_or_in_rt_type Variable Class

経路のタイプを表す整数。経路のタイプには以下のようのものがある

Value	Route Type
0	Route to Nowhere – ignored
1	Route to Directly Connected Network
2	Route to a Remote Host
3	Route to a Remote Network
4	Route to a Sub-Network

The _GW_pr_in_rt_how–learned Variable Class

経路情報のでどころ (source) を表す oecte string。値には次のものがある。

- STSIC
- EGP
- RIP

The _GW_pr_in_rt_metric0 Variable Class

経路のコスト、メトリックを表す

The _GW_pr_in_rt_metric1 Variable Class

経路のコスト、メトリックを表す

DECnet Protocol Variables

DEC Digital Network Architecture のプロトコルとメカニズムに関する情報

XNS Protocol Variables

Xerox Network System のプロトコルとメカニズムに関する情報

第 2 章

管理情報：ISO SMI

管理情報 (Management Information) には、次の二つの顔がある。

- **SMI(Structure of Management Information):**
  管理情報の論理的な構造とこの管理情報を指示する方法を規定する
- **MIB(Management Information Base):**
  SMI を用いて表現され、実際にどのようなオブジェクトを管理するかを規定する

この章では、ISO SMI について説明し、Internet SMI および、MIB については後の章で説明する。

2.1 ISO SMI

ISO SMI は、管理対象オブジェクトの抽象化を定義している。ISO SMI では、Object Oriented の考え方を使って管理情報のモデル化を行なっている。

管理対象オブジェクト (managed object) とは、管理可能 (manageable) な物理的または論理的なネットワーク資源を管理することを目的に抽象化したものである。管理可能とは、ここでは、CMIP を利用して管理可能な資源のことで、例えば、プロトコルエンティティやモデム、コネクションなどが管理可能な物理的または論理的なネットワーク資源である。

個々の管理対象オブジェクトは、ある特定のオブジェクトクラス属している。オブジェクトクラスとは、同じようなもしくは似通ったプロパティをもった管理対象オブジェクトの集まりを示す。あるネットワークに存在する特定の管理対象オブジェクトは、オブジェクトインスタンス (Ojbect instance) と呼ばれ、これは、あるオブジェクトクラスの実態として表現される（例えば、あるクラスに属する管理対象オブジェクトは、特定な値によって制限される）。オブジェクトクラスの例としては、“transport connection” にようなものをあげることができる。実際のネットワークでは、管理対象オブジェクトとして（あるトランスポートコネクション）が複数存在し、それらは、このクラスのインスタンスとなる。

管理対象オブジェクトは、オブジェクトが持っているプロパティもしくは “アトリビュート” を指定することによって定義される。すなわち、オブジェクトによって実行される

CMIS のオペレーション (e.g., M-SET, M-CRATE)、これらのオペレーションによって強制される実行 (e.g., self-test)、オブジェクトが発生するイベント (event)、そしてオブジェクトが内包しているいろいろな関係に関する情報、これらのプロパティによって管理対象オブジェクトは、定義される。

管理対象オブジェクトは、プロパティを持ち、これは、アトリビュートとして参照される。アトリビュートは、アトミックなものである。アトリビュートの例としては、例えば、再送パケットの数のような特定の情報である。

個々のオブジェクトクラスやアトリビュートは、ユニークな識別しが付けられている (ASN.1 OBJECT IDENTIFIER)。

2.2 管理情報の階層 (Management Information Hierarchies)

管理対象オブジェクトは、お互いの関係を持っている。管理情報の特に重要な関係は次の二つである。

- 包含関係 (containment relationship)
- 相続関係 (inheritance relationship)

これらの関係によって管理対象オブジェクトの階層構造が作られる (containment hierarychy および inheritance hierarchy)。また、この他にオブジェクトクラスやアトリビュートの識別子を登録するための処理によって定義される階層構造がある (registeration hierarchy)。

2.2.1 The Registration Hierarchy

Registration hierarchy は、OBJECT IDENTIFIER をつけるために用いられる ANS.1 registration tree によって決定される。OBJECT IDENTIFIER は、数字の列びで、ASN.1 registration tree のルートノードからのパスによって構成される。例えば、

{ iso(1) standard(0) ips-osi-mips(9596) cmip(2) } (1.0.9596.2) は、CMIP 標準のユニークな識別子として用いられる。この木構造の個々のノードは関連する登録組織を持っていて、この組織が部分木のノードの番号を決定する。ネットワーク管理では、これらの OBJECT IDENTIFIER は、オブジェクトクラスやアトリビュートを識別するために用いられる。

Registration hierarchy は、管理対象オブジェクト間や管理対象オブジェクトとそのアトリビュートなどの特定な関係には関係ない。すなわち、containment relationship や inheritance hierarchy とは、一次独立であり、単に全体的にユニークな識別子を生成するためだけの目的である。

2.2.2 The Containment Hierarchy

Containment hierarcy は、オブジェクトとアトリビュートの包含関係 (is contained in) によって構成される。１つのクラスのオブジェクトは、同一もしくは、異なったクラスのオブジェクトを含むことができ、オブジェクトは、またアトリビュートを含むことができる。しかし、アトリビュートは、オブジェクトや他のアトリビュートを含むことはできない。例えば、“transport entity” クラスのオブジェクトは、“transport connection” クラスのオブジェクトを含むことができるし、“management domain” クラスのオブジェクトは、“node” クラスのオブジェクトを含むことができる。

他のオブジェクトクラスを内含しているオブジェクトクラスのことを “superior” オブジェクトクラスと呼び、あるオブジェクトクラスに内含されるオブジェクトクラスのことを “subordinate” オブジェクトクラスとよぶ。すべてのオブジェクトクラス (最上位を省く) は、containment tree 上では、少なくとも１つの可能な “superior” オブジェクトクラスを持っている。クラスの定義としては、１つ以上の “superior” クラスを持つことも許している。しかし、このようなクラスの個々のインスタンスは、可能なクラスに属しているインスタンスのたった１つのインスタンスでなければならない。

管理対象オブジェクトのインスタンスを識別するために containment hierarchy は、非常に重要である。例えば、“domain” オブジェクトクラスがあり、このオブジェクトクラスは、“node” オブジェクトクラスを含んでいて、“node” オブジェクトクラスは、“transport entity” オブジェクトクラスを含み、この “transport entity” オブジェクトは、“transport connection” オブジェクトクラスを含んでいると仮定しよう。このような関係では、ある特定の “transport connection” は、個々のオブジェクトクラスによるインスタンスインフォメーション “instance information” のつながりとして識別される。例えば、

```
{domain=‘‘organization,’’ node=‘‘herakes,’’ transportentity=tp4,
            transport connection=<TSAP-AddressA, TSAP-AddressB> }
```

となる。

個々のオブジェクトクラスにとってのインスタンスインフォメーションは、オブジェクトクラスの定義によって決められ、これは、特徴付けられたアトリビュート (“distinguished attribute(s)”) として知られる。この特徴付けされたアトリビュートは、アトリビュートの OBJECT IDENTIFIER とそのアトリビュートの値から構成される。クラスのインスタンスの区別をしている特徴付けられたアトリビュート (distinguished attribute) は、“relative distinguished name” の集合として呼ばれている。relative distinguished names の列びが管理対象オブジェクトの distinguished name である。先ほどの例は、transport connection の distinguished name の表現となる。

Containment relationship は、各コンポーネントを取り巻く存在を定義していることにもなる。すなわち、オブジェクトやアトリビュートが存在するというのは、そのオブジェクトが包含しているオブジェクトが存在している場合だけである。あるオブジェクトの削拾は、このオブジェクトに含まれているすべてのオブジェクトとアトリビュートの削拾となる。

2.2.3 Inheritance Hierarchy

Inheritance hierarchy は、オブジェクトクラスの継承されたプロパティ(inherits properties of) の関連によって構築される。あるオブジェクトクラスは、他のオブジェクトクラスの継承されたプロパティでもある。詳細化は、さらに付加的なプロパティを追加することによって行なわれる。この関係では、親クラスを “superclass” と呼び、継承するクラスを “subclass” と呼ぶ。例えば、“layer entity” クラスは、“network entity” の superclass となり、“X.25 netowrk entity” の superclass でもある。すなわち、“network entity” のためにアトリビュートの定義は、自動的に “X.25 network entity” でも定義される。このように、オブジェクトオリエンテッド手法を用いることによって簡単にオブジェクトクラスを定義することができる。

個々のクラス（最上位を拾く）は、少なくとも１つの superclass を持っていなければならない。しかし、subclass は、なくても、または複数持っていても構わない。“top” と呼ばれる特別なオブジェクトは、根本の superclass であり、これは、自分自身のプロパティを持っていない。

Inheritance hierarchy は、オブジェクトインスタンスの名前とは関係ない。オブジェクトクラスの定義をするにあたっての、管理しやすいそして拡張性に富んだ技術として有用である。

第 3 章

管理情報：Internet SMI

RFC1155 で定義されている Internet SMI は、管理対象オブジェクトの識別方法とそれらの定義方法について示している。その目的は TCP/IP ベースの internet を管理する際の管理情報を定義するために、共通の構造と表現方法を定めることである。プロトコル独立な SMI として設計され、SNMP と CMIP で共有することが可能なようになっている。管理に使用する管理情報オブジェクトとそのオブジェクトを管理するためのプロトコルについては言及しない。それらは、それぞれ、 MIB と SNMP で定められる。

3.1 オブジェクトタイプの定義

Internet SMI は、ルートノード {`iso(1) org(3) dod(6) internet(1)`} から下の部分木を使って管理オブジェクトをユニークに識別するための OBJECT IDENTIFIER を定義している。現在の Internet SMI は、オブジェクトタイプの形式とこれに付随した OBJECT-TYPE ASN.1 マクロの形式を定義している。オブジェクトタイプの定義は、次の五つのフィールドから構成されている。

- テキスト形式の名前とそれに対応する OBJECT IDENTIFIER
- ASN.1 構文
- オブジェクトタイプの意味の定義
- アクセス (read-only, read-write, write-only, not-accssible)
- ステータス (mandatory, optional, obsolete).

現状の Internet SMI では、管理対象オブジェクトに関連した action や event を定義するための如何なる機能も提供していない。

次にオブジェクトクラスの定義を示すが、これは OSI SMI で定義されている OBJECT-CLASS マクロを簡単にしたものからきている。

- **Definition:**
  オブジェクトクラスの記述

- **Subclass Of:**
  superclass の OBJECT CLASS DESCRIPTOR。このフィールドは、inheritance relationship を示すために使われる。

- **Superiors:**
  superior オブジェクトクラスの OBJECT CLASS DESCRIPTOR のリスト。このフィールドは、containment relationship を示すために使われる。

- **Names:**
  distinguished attributes の OBJECT TYPES を示す OBJECTDESCRIPTIOR。

- **Attributes:**
  attribute の OBJECT TYPE を示す OBJECT TYPE。

CMIS 管理対象オブジェクトクラス変数は、OBJECT IDENTIFIER で示され、例えば、「ip」オブジェクトクラスのための管理対象オブジェクトクラスは、

```
{mib 4} = 1.3.6.1.2.1.4.
```

となる。CMIS アトリビュート識別リスト変数は、アトリビュート識別子のリストで、これには、ローカルなものとグローバルなものがある。もし、グローバルな場合には、アトリビュートに依存した OBJECT IDENTIFIER がある。例えば、「ipForwarding」は、

```
{ip 1} = 1.3.6.1.2.4.1
```

となる。

アトリビュート識別子がローカルな場合には、このオブジェクトを示している OBJECT IDENTIFIER の最後の部分が整数となる。例えば、ipForwarding の場合、ローカルアトリビュート識別子は、1 となる。このようにローカルな識別子を用いる場合には、アトリビュートのための OBJECT IDENTIFIER の先頭の部分は、このオブジェクトクラスを含んでいる OBJECT IDENTIFIER でなくてはならない。ローカル識別子 CMIP PDU の管理対象オブジェクトクラスフィールド関連として解釈される。もし、ローカルアトリビュート識別子が CMIP PDU 内に存在すれば、このグローバルな形式での識別子は、このローカル識別子を示すための管理対象オブジェクトクラスの中にある OBJECT IDENTIFIER をつけることによって構成される。この値はスコープ (例えば baseObject のスコープ) を利用していないときにだけ意味を持つ。もし、スコープを利用している場合には、ローカル形式ではなく、グローバルなアトリビュート識別子の形式を用いなければならない。

3.2 管理情報の階層構造

3.2.1 Registration Hierarchy

グローバルオブジェクト Registration tree は、RFC 1065(“Management Information for TCP/IP-based internets”) に示されている。これは、単純にオブジェクトクラスとアトリビュートの識別子を定義している。

3.2.2 Containment Hierarchy

前に述べたように、Containment hierarchy は、オブジェクトのインスタンスを識別するのに用いられる。オブジェクトクラス定義の Names フィールドにそのクラスの distinguished アトリビュートが含まれている。“attribute” の名前である OBJECT IDENTIFIER とその値のセットのことを “attribute value assertion” と呼ぶ。

例えば、IP のルーティングテーブルのエントリのインスタンスを例にしてみると、オブジェクトクラス “ipRouteEntry” は、containment hierarchy の中で、“superior” クラスが ipRouting Table” であるというようなことによって行なわれる。この処理は、“system”,“ip”,“ipRoutingTable”,“ipRouteEntry” というパスが生成されるまで続けられる。パス中にある個々のオブジェクトクラスで、distinguished アトリビュートを見つけるために各オブジェクトクラスの Names フィールドを調べる。もし、Names フィールドがない場合には (例えば、“ip” や “ipRoutingTable” には Name フィールドがない)、そのレベルでのインスタンス情報がないということになる。“system” と “ipRouteEntry” は、Names フィールドがあり、これらは、このレベルでの侃待される情報となる。この情報を使って、IP のルーティングテーブルエントリのインスタンスを示すための distinguished 名を以下のように構成することができる。

```
baseManagedObjectInstance {
      distinguishedName {
         relativeDistinguishedName {    -- system
            attributeValueAssertion {
               attributeType { cmotSystemID }
               attributeValue "gateway1.acme.com"
            }
         },
         relativeDistinguishedName {    -- ipRouteEntry
            attributeValueAssertion {
               attributeType { ipRouteDest }
               attributeValue 10.0.0.51
            }
         }
```

```
    }
}
```

もし、“system” インスタンス情報がない場合には、このシステムが他の管理要求を受け付けていると仮定する。

オブジェクトインスタンスツリーは、管理システム (node) 以外の distinguished 名をそのコンポーネントに含むことができる。これは、管理ドメイン (Managment domain) やノードの集合を横断したオブジェクトの参照を可能としている。ネットワーク中の中間マネージャは、要求を出すことが可能で、個々の中間マネージャは、“system” 部を利用してどこに要求を出せばよいか、またどこからその結果を受けとるかを決定するのに利用できる。この名前を用いたテクニックは、個々の中間マネージャを “proxy” マネージャとして利用することに用いることができる。“proxy” マネージャは、このチェーンの中の次のノードのアドレスを理解でき、そして、要求や結果を転送するのに異なったプロトコルが利用できる。すなわち、“system” インスタンス情報は、“proxy” として装置の名前付けに利用できる。

3.2.3 Inheritance Hierarchy

Internet SMI は、inheritance relationship を使っていない。“Subclass Of” フィールドは、オブジェクトクラスの inheritance relationship を表現したり、将来的な拡張のためにある。

3.3 オブジェクト型

各オブジェクトの型は

- name
- syntax
- encoding

を持つ。

3.3.1 name

管理オブジェクトを識別するために使用される名前である。”OBJECT IDENTIFIER” として一意に表わされる。

- name は階層構造を持っている。

- OBJECT IDENTIFIER は木構造をたどるための整数値の並びである。
- root ノードにはラベルがなく、(少なくとも) 三つの子ノードが存在する。

  iso(1) ISO により管理されるノード。

  ccitt(0) CCITT により管理されるノード。

  joint-iso-ccitt(2) ISO と CCITT が共同で管理するノード。

- iso(1) ノードの下には他の組織が使用するサブツリー org(3) がある。さらにその下に U.S. Department of Defense の dod(6) が割り当てられている。
- DoD は Internet Activities Board が管理するノードを次のように割り当てている。

```
internet    OBJECT IDENTIFIER ::= { iso org(3) dod(6) 1 }
```

  これにより、Internet 内の OBJECT IDENRIFIER は全て次の prefix を持つことになる。

```
1.3.6.1.
```

- IAB は初めに 4 ノードを定めた。

```
directory       OBJECT IDENTIFIER ::= { internet 1 }
mgmt            OBJECT IDENTIFIER ::= { internet 2 }
experimental    OBJECT IDENTIFIER ::= { internet 3 }
private         OBJECT IDENTIFIER ::= { internet 4 }
```

  directory(1) 将来 OSI Directory で使用するために予約されている。

  mgmt(2) IAB-approved documents で定義されるオブジェクトのために使用される。mgmt(2) の管理は IAB から IANA (Internet Assigned Numbers Authority) に任されている。Internet standard MIB の OBJECT IDENTIFIER は

  `{ mgmt 1 }` もしくは `1.3.6.1.2.1`

  と表される。

  experimental(3) 実験用に使用される。experimental(3) の管理は IAB から IANA に任されている。たとえば実験をする人が 17 を割り当てられた場合には、OBJECT IDENTIFIER は

`{ experimental 17 }` もしくは `1.3.6.1.3.17`

と表される。

private(4) 一方的に定義されるオブジェクト用に使用される。private(4) の管理は IAB から IANA に任されている。private(4) には少なくとも一つ子ノードがある。

```
enterprises   OBJECT IDENTIFIER ::= { private 1 }
```

enterprises(1) ネットワーク機器を提供する組織が商品の形式を登録するために使用される。enterprises のサブツリーを割り当てられた企業は、その中に独自の MIB オブジェクトを定義することができる。企業は管理プロトコル上曖昧に解釈されるのを防ぐため、自分で定めたサブツリーの内容を登録する必要がある。

3.3.2 syntax

オブジェクト型に対応する抽象データ構造を定義する。ASN.1 に従ってはいるが、ASN.1 を全てサポートしているわけではない。

- ASN.1 の primitive type として、次の四つを使用する。これらは構造を持たない型である。
  - INTEGER
  - OCTET STRING
  - OBJECT IDENTIFIER
  - NULL
- ASN.1 の constructor type としては、SEQUENCE のみ使用する。SEQUENCE には二種類の使用方法があり、構造を持つ型として扱われる。
  - 次の形式でリストを扱うことができる。各 <type> には ASN.1 のいずれかの型を指定し、最終的に primitive type にまで展開できる必要がある。(DEFAULT と OPTIONAL は使用しない)

    ```
    SEQUENCE { <type1>, ..., <typeN> }
    ```

  - 次の形式でテーブルを扱うことができる。<entry> にはリストを指定する。

    ```
    SEQUENCE OF <entry>
    ```

- 以上に加え、いくつか application-wide の型が定義されている。それらはIMPLICITに ASN.1 の primitive type、リスト、テーブルおよび他の application-wide の型に展開される。現在は次のような application-wide の型が定義されている。

  NetworkAddress いくつかのプロトコルファミリから一つ CHOICE してアドレスを表す。現在は Internet ファミリだけ定義されている。

  IpAddress 32 ビットのインターネットアドレスを表す。長さ 4 の OCTET STRING にネットワークバイトオーダで含まれる。

  Counter 単調増加する非負の整数値を表す。最大値に達すると 0 に戻る。[1]

  Gauge 増減する非負の整数値を表す。最大値は越えない。

  TimeTicks ある時点から経過した時間を、$\frac{1}{100}$ 秒単位で計った数値を表す。

  Opaque 任意の ASN.1 形式をサポートするために使用される。値は ASN.1 の basic endoding rule に従ってオクテット列として解釈され、次に OCTET STRING として解釈される二重構造となっている。正しいインプリメントにおいては Opaque に encode されたデータを受け取れて処理できる必要があるが、必ずしも内容まで解釈できる必要はない。

3.3.3 encoding

オブジェクト型を syntax によって表現する際に使われる値を指す。簡単にいえばオブジェクトをネットワーク上で扱う場合の数値である。

3.4 SMI の定義

```
RFC1155-SMI DEFINITIONS ::= BEGIN

EXPORTS -- EVERYTHING
        internet, directory, mgmt,
        experimental, private, enterprises,
        OBJECT-TYPE, ObjectName, ObjectSyntax, SimpleSyntax,
        ApplicationSyntax, NetworkAddress, IpAddress,
        Counter, Gauge, TimeTicks, Opaque;

-- the path to the root

internet      OBJECT IDENTIFIER ::= { iso org(3) dod(6) 1 }
```

[1] SMI では最大値は $2^{32} - 1$ (十進で 4294967295) としている。

```
directory     OBJECT IDENTIFIER ::= { internet 1 }

mgmt          OBJECT IDENTIFIER ::= { internet 2 }

experimental  OBJECT IDENTIFIER ::= { internet 3 }

private       OBJECT IDENTIFIER ::= { internet 4 }
enterprises   OBJECT IDENTIFIER ::= { private 1 }

-- definition of object types

OBJECT-TYPE MACRO ::=
BEGIN
    TYPE NOTATION ::= "SYNTAX" type (TYPE ObjectSyntax)
                      "ACCESS" Access
                      "STATUS" Status
    VALUE NOTATION ::= value (VALUE ObjectName)

    Access ::= "read-only"
                    | "read-write"
                    | "write-only"
                    | "not-accessible"
    Status ::= "mandatory"
                    | "optional"
                    | "obsolete"
END

   -- names of objects in the MIB

   ObjectName ::=
       OBJECT IDENTIFIER

   -- syntax of objects in the MIB

   ObjectSyntax ::=
       CHOICE {
           simple
               SimpleSyntax,

   -- note that simple SEQUENCEs are not directly
```

```
-- mentioned here to keep things simple (i.e.,
-- prevent mis-use).  However, application-wide
-- types which are IMPLICITly encoded simple
-- SEQUENCEs may appear in the following CHOICE

        application-wide
            ApplicationSyntax
    }

   SimpleSyntax ::=
       CHOICE {
           number
               INTEGER,

           string
               OCTET STRING,

           object
               OBJECT IDENTIFIER,

           empty
               NULL
       }

   ApplicationSyntax ::=
       CHOICE {
           address
               NetworkAddress,

           counter
               Counter,

           gauge
               Gauge,

           ticks
               TimeTicks,

           arbitrary
               Opaque
```

```
-- other application-wide types, as they are
-- defined, will be added here
    }

-- application-wide types

NetworkAddress ::=
    CHOICE {
        internet
            IpAddress
    }

IpAddress ::=
    [APPLICATION 0]          -- in network-byte order
        IMPLICIT OCTET STRING (SIZE (4))

Counter ::=
    [APPLICATION 1]
        IMPLICIT INTEGER (0..4294967295)

Gauge ::=
    [APPLICATION 2]
        IMPLICIT INTEGER (0..4294967295)

TimeTicks ::=
    [APPLICATION 3]
        IMPLICIT INTEGER (0..4294967295)

Opaque ::=
    [APPLICATION 4]          -- arbitrary ASN.1 value,
        IMPLICIT OCTET STRING   --   "double-wrapped"

END
```

第 4 章

管理情報：MIB

ここでは、RFC1156 「Management Information Base for Network Management of TCP/IP-based internets」に基づき、TCP/IP プロトコル体系を基盤とするインターネットにおけるネットワークマネージメントプロトコルを使用する際の MIB について説明する。

この MIB の仕様は多くの Internet ホストを監視、制御するための変数を定義するものであるが、すべての定義されている変数グループが、すべてのホストに対して必須ものではないというのが、IAB のポリシである。

例えば、EGP グループは EGP を使っているゲートウェイにとっては必須であるが、EGP が走っていないような個々のホストではそうではない。同様に、TCP が走っているホストでは TCP グループは必須であるが、そうでないゲートウェイなどではその必要はない。つまり、必須であるとは、あるグループの中のどれかの変数でもサポートされていたら、そのグループの全ての変数がサポートされているべきであることを言う。

Internet の構成要素が変わったり新たに加わったりという要求を監視、制御できるためのさらなる MIB グループや変数が定義されることが期待されている。

初期の MIB の中で必要と考えられた基準は次のものであった。

1. 故障管理や構成管理のために本質的に必要とされたオブジェクト
2. 制御性の弱いオブジェクトが許された（弱いとは、そのオブジェクトを操作することによって起きる被害が限られるという意味である）。また、この基準は現在のマネージメントプロトコルが強力な制御オペレーションに対して十分安全ではないことを反映している。
3. その時点での使用と有用性の証拠が要求された。
4. 製造業者がソフトウェアを完全に開発するのを簡単にするために、オブジェクトの数を約 100 個に制限しようという試みがなされた。
5. 必要以上の変数は避けるために、ほかの MIB から導くことができるものはオブジェクトから除外されることが要求された。
6. 実装に特有なオブジェクトは除外された。

7. クリティカルセクションの実装を避けることが合意された。一般的な指標としては、各層での各クリティカルセクションについて一つのカウンタを用いるということであった。

4.1　オブジェクト

オブジェクトは MIB という仮想的な情報集積所を通してアクセスされる。MIB の中のオブジェクトは ASN.1 を用いて定義されている。

各オブジェクトは、名前、シンタックスそしてコード化されたものを持っている。名前とは管理上割り当てられたオブジェクト識別子で、これはオブジェクトタイプを指定している。オブジェクトタイプとはオブジェクトインスタンスとともに、オブジェクトを一意に識別する。また、オブジェクトタイプを参照する人間のために、しばしば、OBJECT DESCRIPTER と呼ばれる文字列が使われる。

オブジェクトタイプのシンタックスは、オブジェクトタイプの抽象データ構造を定義する。ここでは ASN.1 が用いられる。

オブジェクトタイプのコード化はオブジェクトタイプのシンタックスに応じて単純に表される。このシンタックスとコード化は、データがネットワークを通る時に必要な概念である。ここでは、ASN.1 の基本符号化規則の用い方を定めている。

4.1.1　オブジェクトグループ

このオブジェクトリストには本質的な要素のみが含まれているので、固有のオブジェクトがオプションとして加わる必要はない。そのオブジェクトは以下のようなグループに分けられている。

- System
- Interfaces
- Address Translation
- IP
- ICMP
- TCP
- UDP
- EGP

このようにグループ分けする理由は二つある。

1. オブジェクト識別子割り当ての意味づけのため

2. agent を実装するための方法を提供するため

あるグループのセマンティクスが適応可能なら、そのグループ内の全てのオブジェクトが実装されねばならない（例えば、もし EGP プロトコルが実装されていたら EGP グループは実装しなければならない)。

4.1.2 定義の形式

オブジェクトタイプは以下のフィールドを用いて定義されている：

OBJECT
OBJECT IDENTIFIER に応じた OBJECT DESCRIPTOR と呼ばれる名前（textual）。

Syntax
ASN.1 を用いたオブジェクトタイプ用アブストラクトシンタックス。これは SMI で定義された *ObjectSyntax* に基づいていなければならない。

Definition
オブジェクトタイプのセマンティクスの記述（textual)。この MIB はマルチベンダ環境用に作られているので、実装はこの定義を十分に満たしていなければならない。

Access
read-only、*read-write*、*write-only*、*not-accesible* のうちの一つ。

Status
mandatory、*optional*、*obsolete* のうちの一つ。

4.2 オブジェクトの定義

ASN.1 で表すと次のようになる。

```
RFC1156-MIB

DEFINITIONS ::= BEGIN

IMPORTS
        mgmt, OBJECT-TYPE, NetworkAddress, IpAddress,
        Counter, Gauge, TimeTicks
            FROM RFC1155-SMI;
```

```
mib         OBJECT IDENTIFIER ::= { mgmt 1 }

system      OBJECT IDENTIFIER ::= { mib 1 }
interfaces  OBJECT IDENTIFIER ::= { mib 2 }
at          OBJECT IDENTIFIER ::= { mib 3 }
ip          OBJECT IDENTIFIER ::= { mib 4 }
icmp        OBJECT IDENTIFIER ::= { mib 5 }
tcp         OBJECT IDENTIFIER ::= { mib 6 }
udp         OBJECT IDENTIFIER ::= { mib 7 }
egp         OBJECT IDENTIFIER ::= { mib 8 }

END
```

4.2.1 The System Group

System group の実装は全てのシステムにおいて必須のものである。

4.2.2 The Interfaces Group

Interfaces group の実装は全てのシステムに於いて必須のものである。

4.2.3 The Address Translation Group

Address Translation grop の実装は全てのシステムに於いて必須のものである。

この Address Translation group はネットワークアドレス（例えば、IP アドレス）からサブネットアドレスへの変換テーブルも含むべきである。しかし、ここでは適当な定義をしてないため、サブネットアドレスはあたかも物理アドレスかのようになっている。

変換テーブルの例を挙げてみる。たとえば、ブロードキャストメディアとして ARP を用いる場合、変換テーブルは ARP キャッシュと同じ意味を持つ。アルゴリズミックな変換機能が必要のない X.25 ネットワークでは、変換テーブルはネットワークアドレスから X.121 アドレスへの変換機能と同じ意味を持つことになる。

4.2.4 The IP Group

IP group の実装は全てのシステムに於いて必須のものである。

- **The IP Address Table**
  IP アドレスに関連した情報を持っている。

- **The IP Routing Table**
  ルーティングテーブルエントリを持っている。持っていない情報をさらに取りに行くような行為は、ネットワークマネージメントプロトコルによってなされるべきである。

4.2.5 The ICMP Group

ICMP group の実装は全てのシステムに於いて必須のものである。
ICMP group は ICMP の入出力の統計情報を持っている。
このバージョンの MIB では単純化のため、ICMP としての合計メッセージ数だけで、ここのメッセージ毎の数は管理しない。

4.2.6 The TCP Group

TCP group の実装は全てのシステムに於いて必須のものである。
このグループのオブジェクトインスタンスは、TCP コネクションが張られている時に限り存在するといった、一時的なものである。

4.2.7 The UDP Group

UDP group の実装は、UDP プロトコルを実装している全てのシステムに於いて必須のものである。

4.2.8 The EGP Group

EGP group の実装は、EGP プロトコルを実装している全てのシステムに於いて必須のものである。

- **The EGP Neighor Table**
  EGP 近傍に関する情報を持っている。

4.3 定義

RFC1156-MIB 全体の定義は、長いのでここでは省略する。

第 5 章

管理情報：MIB-II

ここでは、RFC1158 「Management Information Base for Network Management of TCP/IP-based internets: MIB-II」に基づき、TCP/IP プロトコル体系を基盤とするインターネットにおけるネットワークマネージメントプロトコルを使用する際の MIB の第二版、つまり MIB-II について説明する。

MIB-II は MIB-I の内容との互換性を失うことなくさらに拡張したものである。MIB は Internet の Recommended な Standart Protocol であるが、MIB-II は Elective な Proposed Standard Protocol である。

MIB-II は MIB を増加、改良したものであり、以下の二つの概念に基づいて設計されている。

1. 変更は Internet での要求に基づいてなされる。
2. 変更は、MIB-I から MIB-II への移行がスムーズに済むよう上位互換でなされる。

Internet の構成要素が変わったり新たに加わったりという要求を監視、制御できるためのさらなる MIB グループや変数が定義されることが予想されている。

短期的に、単純かつ機能するシステムを作るという IAB の指示を満たすために、MIB-I の中で基準とされていた七つの項目のうち、次に示す一つが除外された。

7. クリティカルセクションの実装を避けることが合意された。一般的な指標としては、各層での各クリティカルセクションについて一つのカウンタを用いるということであった。

5.1 MIB-I からの変更点

MIB-II の特徴は次の通りである。

1. 新たな運用上の要求を反映したオブジェクトの追加
2. SMI/MIB や SNMP との上位互換性
3. マルチプロトコルエンティティに対するサポートの改良

4. わかりやすさ、読みやすさを改良するための MIB の文章的な整理

MIB-II で定義されたオブジェクトは以下の OBJECT IDENTIFIER を持っている。

```
mib-2      OBJECT IDENTIFIER ::= { mgmt 1 }
```

5.1.1 Deprecated オブジェクト

より良い実装を目指すため、MIB-II では新しく「deprecated」という用語をオブジェクトを記述するために用いることがでいる。このオブジェクトは、現在の MIB ではサポートされているが、新しい MIB（例えば、MIB-III）では除外されるのがほとんど確かなものである。

MIB-II では、以下の一つのオブジェクトを deprecated の対象にしている。

```
atTable
```

これにより、全ての Address Translation グループが除外候補になる。また、あるオブジェクトを除外することによりそのオブジェクトが持ち合わせた機能をなくすのではなく、MIB-II の中で同等のあるいはそれ以上の機能を持った新しいオブジェクトが定義される。

5.1.2 Display Strings

従来の MIB の中に、オクテット列は印刷可能文字、つまり人間が読めるものを含むべきだという MIB の誤った解釈があった。MIB の慣習では、

```
DisplayString ::= OCTET STRING
```

というように導かれている。DisplayString は NVT ASCII 文字でなければならないという制限を受けている。

sysDescr や ifDescr は DisplayString として新たに定義されている。このことはシンタックスやセマンティクスにはななんら影響を与えない。DisplayString の記法は MIB-II やそれ以降のバージョンの中でオブジェクトの説明のために加えられた一つの機能にすぎない。

また、OCTET STRING で定義されたオブジェクトは、二進数データ（0-255 これは十進数）を含むことがあることに注意しなければならない。

5.1.3 System Group

以下の 4 つのグループが新しく加えられた。

- sysContact

- sysName
- sysLocation
- sysServices

これらはそれぞれあるノードのログイン情報、管理情報、配置情報、サービス情報を持っている。

5.1.4 Interfaces Group

ifNumber オブジェクトの定義は、IP をサポートしている全てのインタフェースを対象にしているので正確ではない（つまり、サポートしていても実際にはそのデバイスを持たない場合もある）。ifNumber オブジェクトはそれに応じて記述は変更される。
ifTable はステータスフィールドが read-write と誤っている。これを read-only にする。と同時に ifTable オブジェクトの ifType のエントリいくつかの新しい値を増やした。

- ppp(23)
- softwareLoopback(24)
- eon(25)
- ethernet-3Mbit(26)
- nsip(27)
- slip(28)

さらに、ifTable のエントリに ifSpecific というインタフェースをしている媒体（例えば、イーサネット）に関する情報を提供するエントリを加える。

5.1.5 Address Translation Group

MIB-I ではこのグループはネットワークアドレスから物理アドレスへのマップを行なう。この実装には経験から 2 つの仮定が必要である。

1. シングルネットワークプロトコル環境であること。
2. マップはネットワークアドレスから物理アドレスへしかしない。

IP、CLNP の両方をサポートするようなマルチプロトコルへの対応や、逆引きへの対応が必要であるが、これらは上の仮定に当てはまらない。そこで、この atTable オブジェクトは除外対象なのである。

このマルチプロトコルや、逆マップに対応するため、MIB-II では、ネットワークプロトコルグループ内に 2 つのアドレス変換テーブルを持つようにしている。つまり、IP グループでは IP アドレスから物理アドレスへの変換用テーブルを一つ、CLNP では両方向のマップに対応した 2 つのテーブルを持つよう定義されている。

以下のことには注意されたい： 2 つのテーブルから 1 つを選択することは、実装しやすいよう設計しなければならない。同時に、テーブルが一つの場合についてもサポートしていなければならない。

5.1.5.1 IP Group

ipForwarding のアクセスフィールドの値を read-only から read-write へ変更する。加えて、以下のような新しいエントリを導入する。

ipAddrTable オブジェクトには、`ipAdEntReasmMaxSize` を追加する。これは、あるインタフェースから入ってきた IP フラグメントデータグラムでリアセンブル可能な最大 IP データグラムのサイズを示す。

ipRoutingTable オブジェクトには、`ipRouteMask` を追加する。これはサブネットを用いている時使う。

IP グループに新しいオブジェクト、`ipNetToMediaTable` を追加する。これは atTable オブジェクトの変わりに、アドレス変換をテーブルをサポートする。

5.1.5.2 ICMP Group

変更なし。

5.1.5.3 TCP Group

次の二つの変数を加える。

- **tcpInErrs**
  入ってきた TCP セグメントの内エラーとなったもの数。

- **tcpOutRsts**
  生じた TCP リセットの数。

5.1.5.4 UDP Group

新しく udpTable を加える。

5.1.5.5 EGP Group

経験から、egpNeighborTable オブジェクトの値に、EGP エンティティに対応した AS（自律システム）を与える egpAs を加える。

5.1.5.6 Transmission Group

MIB-I では異なった通信媒体の区別をしてないので、それに対応する新しいグループを加える。

5.1.5.7 SNMP Group

ネットワーク統計情報を持つために SNMP グループを加える。

5.2　オブジェクトグループ

このオブジェクトリストには本質的な要素のみが含まれているので、固有のオブジェクトがオプションとして加わる必要はない。そのオブジェクトは以下のようなグループに分けられている。

- System
- Interfaces
- Address Translation（除外）
- IP
- ICMP
- TCP
- UDP
- EGP
- Transmission
- SNMP

このようにグループ分けする理由は二つある。

1. オブジェクト識別子割り当ての意味づけのため
2. agent を実装するための方法を提供するため

あるグループのセマンティクスが適応可能なら、そのグループ内の全てのオブジェクトが実装されねばならない（例えば、もし EGP プロトコルが実装されていたら EGP グループは実装しなければならない）。
オブジェクトタイプは以下のフィールドを用いて定義されている。

OBJECT
OBJECT IDENTIFIER に応じた OBJECT DESCRIPTOR と呼ばれる名前（textual）。

Syntax
ASN.1 を用いたオブジェクトタイプ用アブストラクトシンタックス。これは SMI で定義された *ObjectSyntax* に基づいていなければならない。

Definition

オブジェクトタイプのセマンティクスの記述（textual）。この MIB はマルチベンダ環境用に作られているので、実装はこの定義を十分に満たしていなければならない。

Access

read-only、*read-write*、*write-only*、*not-accesible* のうちの一つ。

Status

mandatory、*optional*、*obsolete* のうちの一つ。

5.3 オブジェクトの定義

ASN.1 で表すと次のようになる。

```
RFC1158-MIB

DEFINITIONS ::= BEGIN

IMPORTS
        mgmt, OBJECT-TYPE, NetworkAddress, IpAddress,
        Counter, Gauge, TimeTicks
            FROM RFC1155-SMI;

DisplayString ::=
            OCTET STRING


mib-2      OBJECT IDENTIFIER ::= { mgmt 1 }    -- MIB-II

system     OBJECT IDENTIFIER ::= { mib-2 1 }
interfaces OBJECT IDENTIFIER ::= { mib-2 2 }
at         OBJECT IDENTIFIER ::= { mib-2 3 }
ip         OBJECT IDENTIFIER ::= { mib-2 4 }
icmp       OBJECT IDENTIFIER ::= { mib-2 5 }
tcp        OBJECT IDENTIFIER ::= { mib-2 6 }
udp        OBJECT IDENTIFIER ::= { mib-2 7 }
egp        OBJECT IDENTIFIER ::= { mib-2 8 }
-- cmot    OBJECT IDENTIFIER ::= { mib-2 9 }
transmission OBJECT IDENTIFIER ::= { mib-2 10 }
snmp       OBJECT IDENTIFIER ::= { mib-2 11 }
```

```
END
```

5.3.1 The System Group

System group の実装は全てのシステムにおいて必須のものである。

5.3.2 The Interfaces Group

Interfaces group の実装は全てのシステムに於いて必須のものである。

- **The Interfaces table**
  インタフェースについての情報を持っている。ここのインタフェースはサブネットワークレイヤのような働きをする。IP アドレスのサブネット分割方法とは別のものであることに注意。

5.3.3 The Address Translation Group

Address Translation grop の実装は全てのシステムにおいて必須のものである。しかしながら、MIB-II では、このグループは除外されることになる。MIB-I を用いているノードのために残してはおくが、MIB-III ではほとんどなくなるであろう。またこれらは MIB-II では、network protocol group に含まれる。

この Address Translation group はネットワークアドレス（例えば、IP アドレス）からサブネットアドレスへの変換テーブルも含むべきである。しかし、ここでは適当な定義をしてないため、サブネットアドレスはあたかも物理アドレスかのようになっている。

変換テーブルの例を挙げてみる。たとえば、ブロードキャストメディアとして ARP を用いる場合、変換テーブルは ARP キャッシュと同じ意味を持つ。アルゴリズミックな変換機能が必要のない X.25 ネットワークでは、変換テーブルはネットワークアドレスから X.121 アドレスへの変換機能と同じ意味を持つことになる。

5.3.4 The IP Group

IP group の実装は全てのシステムに於いて必須のものである。

- **The IP Address Table**
  IP アドレスに関連した情報を持っている。

- **The IP Routing Table**
  ルーティングテーブルエントリを持っている。持っていない情報をさらに取りに行くような行為は、ネットワークマネージメントプロトコルによってなされるべきである。

5.3.5 The ICMP Group

ICMP group の実装は全てのシステムに於いて必須のものである。
ICMP group は ICMP の入出力の統計情報を持っている。

5.3.6 The TCP Group

TCP group の実装は全てのシステムに於いて必須のものである。
このグループのオブジェクトインスタンスは、TCP コネクションが張られている時に限り存在するといった、一時的なものである。

- **The TCP Connection table**
  現在張られている TCP コネクションに関する情報を持っている。

5.3.7 The UDP Group

UDP group の実装は、UDP プロトコルを実装している全てのシステムに於いて必須のものである。

- **The UDP Listner table**
  UDP を用いたローカルアプリケーションがデータグラムを受け入れたかどうかについての情報を持っている。

5.3.8 The EGP Group

EGP group の実装は、EGP プロトコルを実装している全てのシステムに於いて必須のものである。

5.3.9 The Transmission Group

ここのインタフェースに基づいた通信媒体に関するもので、自分のシステムが用いている媒体については必須である。

5.3.10 The SNMP Group

SNMP group の実装は、SNMP プロトコルを実装している全てのシステムにおいて必須のものである。SNMP の実装によっては、以下に定義されたオブジェクトの値がゼロの場合もある。

5.4　定義

RFC1158-MIB 全体の定義は、長いのでここでは省略する。

第 6 章

管理プロトコル：OSI プロトコル体系

6.1 CMOT

CMOT とは、CMIS/CMIP(Common Management Information Services/Common Management Information Protocol) over TCP/IP の略で、RFC-1095 でプロトコルが規定されている 。関連するドキュメントとしては次のものがある。

ASN.1: Abstract Syntax Notation 1

ISO 8824: "Information processing systems - Open Systmes Interconnection, Specification of Abstract Syntax Notation One(ASN.1)," Geneva, March 1988.

ISO 8825: "Information processing systems - Open Systems Interconnection, Specification of Basic Encoding Rules for Abstract Notation One (ASN.1)", Geneva, March 1988.

ACSE: Assocation Control

ISO 8649: "Inforamtion processing systems - Open Systems Interconnection, Service Definition for Association Control Service Element".

ISO 8650: "Inforamtion processing systems - Open Systems Interconnection, Protocol Specification for Association Control Service Element".

ROSE: Remote Operations CCITT Recommendation X.219, Working Document for ISO 9072-1: "Information processing system - Text communication Remote Operations: Model, Notation and Service Definition", Gloucester, November 1987.

CCITT Recommendation X.229, Working Document for ISO 9072-2: "Information processing systems - Text COmmunication, Remote Operations: Protocol Specification", Gloucester, November 1987.

CMIS: Common Information Service

ISO DIS 9595-2: "Information processing systems- Open Systems Interconnection, Management Information Service Definition - Part 2: Common Management Information Service", 22 December 1988.

CMIP: Common Information Procotol

ISO DIS 9595-2: “Information processing systems- Open Systems Interconnection, Management Information Service Definition - Part 2: Common Management Information Protocol”, 22 December 1988.

RFC 1085: Specification of a leightweight presentation layer protocol

Rose, M., “ISO Presentation Services on top of TCP/IP - based internets”, RFC 1085, December 1988.

RFC 1065:

Rose, M., and K. McCloghrie, “Structure and Identification of Management Information for TCP/IP - based internets”, RFC 1065, August 1988

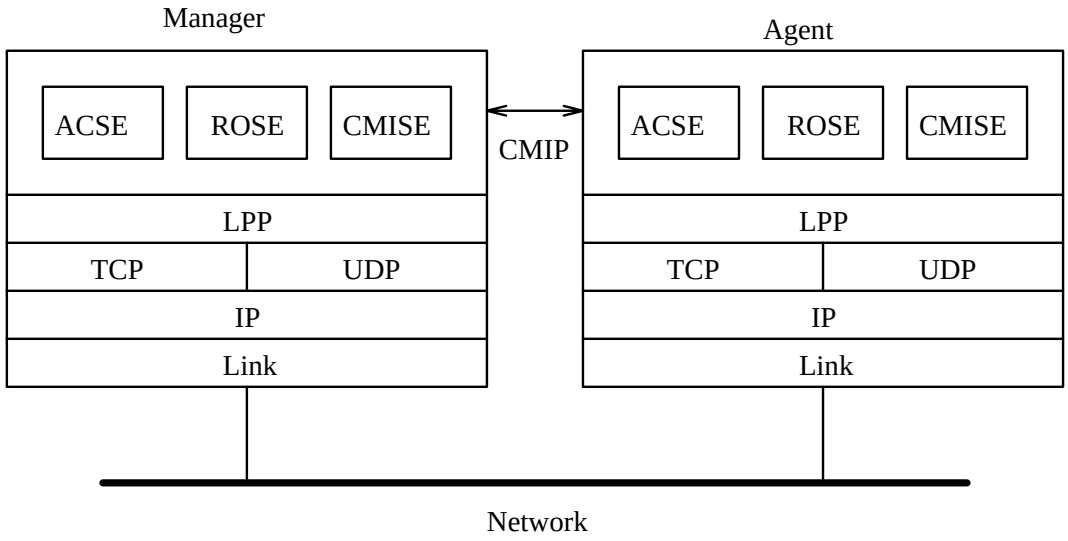
RFC 1066:

McCloghrie, K., and M. Rose, “Management Informatio Base for Network Management of TCP/IP - based internets”, RFC 1066, August 1988

CMOT とは、基本的に OSI 管理フレームワークを基本として、TCP/IP をトランスポートとして利用したものである。OSI の管理モデルに対応すると、Organization モデル的には、シングルドメインとなり、マルチドメインへの拡張は、遠い将来の課題である。Functional モデルでは、５つの基本機能を提供しているが、個々の機能をどの程度提供しているかは、対象とする管理オブジェクトによって決定される。Information モデルとしては、２つの違った SMI がある。１つは ISO によって定義されているもので、もう１つが IETF WG が提案している MIB である。

6.2　プロトコルアーキテクチャ

CMOT のプロトコルアーキテクチャの目的は、OSI のネットワーク管理プロトコルアーキテクチャを TCP/IP 環境にマップすることであり、インターネットワーク環境におけるトランスポートレイヤの機能を使ってアプリケーションレイヤの機能としてこれを実現する。



基本的には、UDP と TCP 上に ROSE,ACSE,CMIP を実装して実現する。この実装においては、RFC 1085 による Lightweight presentation protocol(LPP) を利用する。

Lightweight presetntation protocol とは、ISO のアプリケーション（ROSE や ACSE 等）を TCP/IP 上に構築する場合、プロトコル間のギャップを埋めるための方法の１つで、実際には、ACSE,ROSE,CMISE だけが必要とするプレゼンテーションサービスのみを TCP と UDP 上に作ったプロトコルである。

6.2.1 Proxy Management

Proxy の一般的な意味は、ある実存物が他の行なうことがらを代行するという意味で、ネットワーク管理でも他のマネージャの振舞いを変わって行なうマネージャのことをさす。Proxy Management の有用性は、“security, administrative reasmon, impatible communication mecahnisms, protocol” などの理由でマネージャが管理されるデバイスと直接コミュニケーションできない場合にその有用性がある。

CMOT における proxy には、

1. Transparent sense: CMOT を理解できないデバイスの代わりをする。
2. Intermidate Manager: 複数のエージェントからの情報をまとめる。

の２つがある。

6.3 CMOT プロトコル概要

ここでは、CMOT アーキテクチャのプロトコル仕様について説明する。

将来的な実装に必要な合意とプロトコル標準の集合の選択のことを一般的にプロファイル（profile）と呼んでいる。CMOT プロファイルのための選択基準は、Internet コミュ

ニティーと国際標準 (ISO and CCITT) コミュニティーによって提唱されている標準を利用するということである。基本的には、CMOT プロファイルは、ネットワーク管理に関する ISO 標準と草案を基本としている。また、他の ISO アプリケーションレイヤ標準 (ROSE, ACSE) などは、ISO の管理プロトコル（CMIP）をサポートするものとして使用する。また、Internet での使用のために TCP/IP などを定めた Internet 標準も利用する。

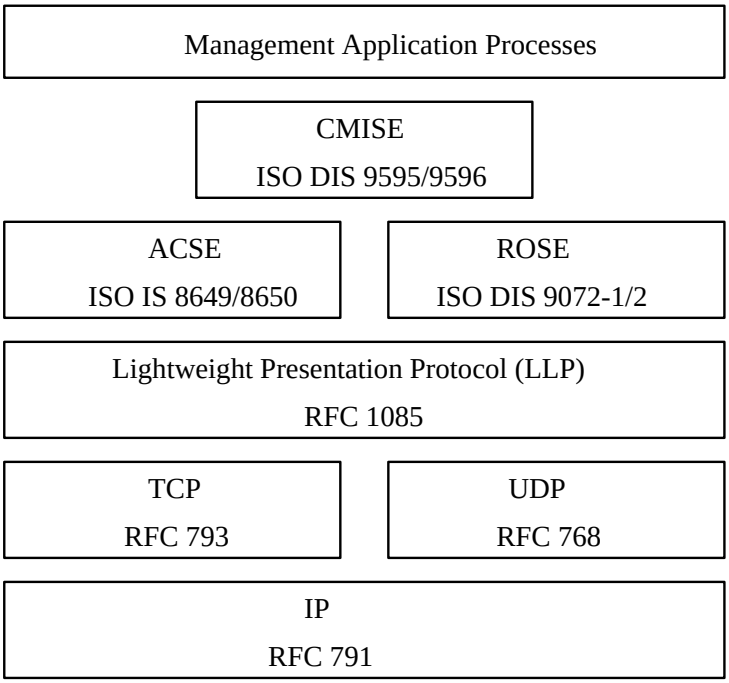
また、少なくとも２つのグループからのい件も聞き入れる。１つは、NIST(National Institute of Science and Technology) の Network Managment Special Interst Group(NMSIG) と OSI の Network Management Forum である。これらは、ネットワーク管理に関する ISO 標準を利用した同様のプロファイルの定義を行なっている。この違いは、一方が ISO プロトコル情に構築することを検討しているのに対して、多尾方は、TCP/IP を基本としている・である。

6.3.1　CMOT プロトコル体系

以下の７つのプロトコルの集合が CMOT プロトコル体系である。

1. ISO ACSE
2. ISO DIS ROSE
3. ISO DIS CMIP
4. the lightweight presentation protocol(LPP)
5. UDP
6. TCP
7. IP

これらの関係は、以下のようになる。



6.3.2 Conformace Rrquiements

ACSE, ROSE, CMIP, LLP そして IP を実装し、少なくとも UDP もしくは、TCP のどちらかに上に LLP を実装しなければならない。もちろん、１つのシステムで TCP と UDP の両方をサポートしてもよい。

適合システムとしては、CMIS オペレーションのすべては、必要としない。次の章で述べる機能グループの中の１つを提供知ればよい。

6.4 CMIS サービス

CMOT で許される CMIP サービスを以下に示す。

Service	Type
M-INITIALISE	confirmed
M-TERMINATE	confirmed
M-ABORT	non-confirmed
M-EVENT-REPORT	confirmed/non-confirmed
M-GET	confirmed
M-SET	confirmed/non-confirmed
M-ACTION	confirmed/non-confirmed
M-CREATE	confirmed
M-DELETE	confirmed

6.4.1　機能ユニット

CMIS サービスでは、機能ユニットと呼ばれるのもを指定している。個々の機能ユニットは、実際のサービスの実行者 (performer) もしくはサービスの要求者 (invoker) に相当する。

Functional Unit	Service Primitives	Mode
conf. event report invoker(0)	M-EVENT-REPORT Req/Conf	C
conf. event report performer(1)	M-EVENT-REPORT Ind/Rsp	C
event report invoker(2)	M-EVENT-REPORT Req	U
event report performer(3)	M-EVENT-REPORT Ind	U
confirmed get invoker(4)	M-GET Req/Conf	N/A
confirmed get performer(5)	M-GET Ind/Rsp	N/A
confirmed set invoker(6)	M-SET Req/Conf	C
confirmed set performer(7)	M-SET Ind/Rsp	C
set invoker(8)	M-SET Req	U
set performer(9)	M-SET Ind	U
confirmed action invoker(10)	M-ACTION Req/Conf	C
confirmed action performer(11)	M-ACTION Ind/Rsp	C
action invoker(12)	M-ACTION Req	U
action performer(13)	M-ACTION Ind	U
confirmed create invoker(14)	M-CREATE Req/Conf	N/A
confirmed create performer(15)	M-CREATE Ind/Rsp	N/A
confirmed delete invoker(16)	M-DELETE Req/Conf	N/A
confirmed delete performer(17)	M-DELETE Ind/Rsp	N/A
multiple reply(18)	Linked Identification	N/A
multiple object selection(19)	Scope, Filter, Sync.	N/A
extended service(20)	Extended Presentation	N/A

C = confirmed, U = non-confirmed, N/A = not applicable

6.4.2　機能ユニットグループ

機能ユニットは、複雑さやコードのサイズを削減するための機能で、エージェントやマネージャは、自分自身がサポートしている機能ユニットグループの番号をもっている。ネゴシエーション時に、この番号を M-INITIALISE サービスのパラメータとして渡し合うことによってどのようなサービスが提供できるかを知らせあう。マネージャ用には、５つの機能ユニットグループ：Event Monitor, Monitoring Manager, Simple Manager, Controlling Manager, そして Full manager があり、エージェントには、: Event Sender, Monitored Agent, SImple Agent, Controlled Agent そして Full Agent がある。

Functional Unit Groups	Event Report	Get	Set	Create/ Delete	Action	Mult. Reply	Mult. Object Select
1. Event Monitor	U	no	no	no	no	no	no
2. Event Sender	U	no	no	no	no	no	no
3. Monitoring Mgr.	U	yes	no	no	no	no	no
4. Monitored Agent	U	yes	no	no	no	no	no
5. Simple Manager	U	yes	C	no	no	yes	no*
6. Simple Agent	U	yes	C	no	no	yes	no*
7. Controlling Mgr.	U	yes	U/C	yes	no	yes	yes
8. Controlled Agent	U	yes	U/C	yes	no	yes	yes
9. Full Manager	U/C	yes	U/C	yes	U/C	yes	yes
10. Full Agent	U/C	yes	U/C	yes	U/C	yes	yes

```
C = confirmed, U = non-confirmed
* Simple Managers and Agents must support "oneLevel" scoping for all
  and only those cases where it is required to access a whole table
  and may support synchronization other than "best effort"; no support
  for filtering is required.
```

6.4.3 M-INITIALISE のパラメータ

M-INITIALISE は ACSE A-ASSOCIATE サービスにより提供される。

起動をかけた CMISE-service-user と応答する CMISE-service-user の間で、機能ユニットの交換が必要であり、そのために、21 bits BIT STRING を定義している。各ビットが functional unit に対応しており、その対応を次に示す。

Functional Unit \ Functional Unit Group No.	1	2	3	4	5	6	7	8	9	10
conf. event report invoker(0)	0	0	0	0	0	0	0	0	0	1
conf. event report perf.(1)	0	0	0	0	0	0	0	0	1	0
event report invoker(2)	0	1	0	1	0	1	0	1	0	1
event report performer(3)	1	0	1	0	1	0	1	0	1	0
confirmed get invoker(4)	0	0	1	0	1	0	1	0	1	0
confirmed get performer(5)	0	0	0	1	0	1	0	1	0	1
confirmed set invoker(6)	0	0	0	0	1	0	1	0	1	0
confirmed set performer(7)	0	0	0	0	0	1	0	1	0	1
set invoker(8)	0	0	0	0	0	0	1	0	1	0
set performer(9)	0	0	0	0	0	0	0	1	0	1
confirmed action invoker(10)	0	0	0	0	0	0	0	0	1	0
confirmed action performer(11)	0	0	0	0	0	0	0	0	0	1
action invoker(12)	0	0	0	0	0	0	0	0	1	0
action performer(13)	0	0	0	0	0	0	0	0	0	1
confirmed create invoker(14)	0	0	0	0	0	0	1	0	1	0
confirmed create performer(15)	0	0	0	0	0	0	0	1	0	1
confirmed delete invoker(16)	0	0	0	0	0	0	1	0	1	0
confirmed delete performer(17)	0	0	0	0	0	0	0	1	0	1
multiple reply(18)	0	0	0	0	1	1	1	1	1	1
multiple object selection(19)	0	0	0	0	0	0	1	1	1	1
extended service(20)	0	0	0	0	0	0	0	0	0	0
	M	A	M	A	M	A	M	A	M	A

`1 = supported, 0 = not supported, M = manager, A = agent`

ここでの、negotiation は次のように行なわれる。

1 起動をかける CMISE-service-user(manager or agent) が functional unit group を表す functional units を送る

2 応答する CMISE-service-user は functional unit group を表す functional units を送る

3 もし二つのアプリケーション entity によってサポートされている functional unit groups が意味のある通信を許していなければ、どちらかの entity が association を拒否する

4 意味のある通信は entity が少なくとも一つ以上のもう一方の entity によってサポートされている CMIS オペレーションを invoke または perfome する能力として定義される

5 association が確立された後、システムはサポートできる functional units に対して正しい応答をし、他の要求に対してはプロトコルに従って拒否しなければならない。

M-INITIALISE パラメータにおける `User Information` はオプションであり、送るためにエンティティは必要としない。マネージャまたはエージェントによりサポートされる MIB の拡張を運ぶことに使える可能性がある。

M-INITIALISE パラメータにおけるアクセス制御は、オプションであり、ACSE を使って各 assocation basis ごとにサポートされる。各 A-ASSOCIATE で使われることが推奨されているが、要求はされていない。要求ごとに行なうことも可能であり、セキュリティ機能の実装に使われることが可能である。定義されていまえば、TCP/IP の認証機構を実装するために使われると期待されている。

6.4.4 Supporting サービス

M-INITIALISE, M-TERMINATE, M-ABORT サービスは ACSE の使用を仮定している。以下の ACSE サービスが必要である。

- A-ASSOCIATE
- A-RELEASE
- A-ABORT
- A-P-ABORT

残りの CMIP プロトコルは ROSE の以下のサービスを使う。

- RO-INVOKE
- RO-RESULT
- RO-ERROR
- RO-REJECT

6.4.5 CMIP Agreements

CMIP 標準で規定されている CMIP agreements への付加 agreements。全部で 13 個ある。

- Invoke Identifier
- Object Class
- Object Instance
- Access Control

- Synchronization
- Scope
- Filter
- Attribute Identifier
- Event Type Identifier
- Action Type Identifier
- Time Fields
- Response PDUs
- Error PDUs

6.5 ACSE

Association Control Service Element(ACSE) は、アプリケーション association の確立とリリースに必要とされる。ISO 8649,8650 で定義されている。

6.5.1 ACSE サービス

ISO8649 に詳しく載っており、必須である。

- `A-ASSOCIATE`
  confirmed service。アプリケーション entity 間のアプリケーション association の起動。
- `A-RELEASE`
  confirmed service。アプリケーション entity 間のアプリケーション association のリリースを情報の喪失なしに行なう。
- `A-ABORT`
  unconfirmed service。association の正常でないリリース。情報の喪失を伴う可能性がある。
- `A-P-ABORT`
  provider-initiated service。基盤となるプレゼンテーションサービスによる正常でないアプリケーション association のリリース。情報の喪失を伴う可能性がある。

ACSE services の presentation services と ACSE APDUs へのマッピングを次に示す。

ACSE Service	ISO 8649 Reference	Related Presentation Service	Associated APDUs
A-ASSOCIATE	9.1	P-CONNECT	AARQ, AARE
A-RELEASE	9.2	P-RELEASE	RLRQ, RLRE
A-ABORT	9.3	P-U-ABORT	ABRT
A-P-ABORT	9.4	P-P-ABORT	(none)

6.5.2 Supporting サービス

ACSEは以下のISOプレゼンテーション層サービスを利用している。これはLPP(Lightweight Presentation Protocol:RFC1085) が提供する。

- P-CONNECT
- P-RELEASE
- P-U-ABORT
- P-P-ABORT

6.5.3 Supporting Services

- ACSE は以下の ISO プレゼンテーション層サービスを利用
  - P-CONNECT
  - P-RELEASE
  - P-U-ABORT
  - P-P-ABORT
- LPP(Lightweight Presentation Protocol:RFC1085) が提供するであろう

6.5.4 ACSE プロトコル

ISO 8650 で定義されている。五つの ACSE APDU はすべて標準で規定されており、必須である。

6.6 ROSE

Remote Operations Service Element(ROSE) は、ISO 9072-1, 9072-2 で定義されている。リモートオペレーションの起動を行なうものであり、二つのアプリケーションエンティティ間でアソシエーションが確立していて使える。ROSE は CMISE をサポートするために使われ、管理アプリケーションプロセスから直接使われることを意図していない

6.6.1 ROSE Services

ISO 9072-1 に詳しく定義されており、必須である。次の五つがある。

- RO-INVOKE
  - unconfirmed service
  - 起動をする ROSE-user が使用
  - オペレーションの実行
- RO-RESULT
  - unconfirmed service
  - 起動された ROSE-user が使用
  - 前の RO-INVOKE が成功した時の応答
- RO-ERROR
  - unconfirmed service
  - 起動された ROSE-user が使用
  - 前の RO-INVOKE が失敗した時の応答
- RO-REJECT-U
  - unconfirmed service
  - ROSE-user が使用
  - RO-INVOKE の要求を拒絶に使われる
  - RO-RESULT や RO-ERROR という応答の拒絶にも使われる
- RO-REJECT-P
  - provider-initiated service
  - ROSE-provider が使用
  - ROSE-user に検出された問題を知らせる

ROSE サービスから ISO プレゼンテーションサービスと ROSE APDU へのマッピングは次のようになっている。

ROSE Service	ISO 9072-1 Reference	Related Presentation Service	Associated APDUs
RO-INVOKE	10.1	P-DATA	ROIV
RO-RESULT	10.2	P-DATA	RORS
RO-ERROR	10.3	P-DATA	ROER
RO-REJECT-U	10.4	P-DATA	RORJ
RO-REJECT-P	10.5	P-DATA	RORJ

6.6.2 Supporting サービス

LPP が提供する。Reliable Transfer Service Element (RTSE) へのマッピングは不可能である。またプレゼンテーションサービスとともには data token は使われない。

6.6.3 ROSE プロトコル

ISO 9072-2 で定義されており、四つの APDU はすべて必須である。multiple reply functional units が選択される場合には、linked-id protocol element の correct origination と reception をサポートする機能が必要である。

また、各 PDU を FIFO で配送するので、Priority parameter は無視される。

6.7 Lightweight Presentation プロトコル

LPP は RFC1085 で定義されており、ACSE と ROSE をサポートするために必要な最小限の ISO プレゼンテーションサービスであり、次のサービスがある。

- P-CONNECT
- P-RELEASE
- P-U-ABORT
- P-P-ABORT
- P-DATA

6.7.1 Supporting サービス

P-CONNECT で示される quality of service に応じて、UDP または TCP を選ぶが、実際に、リモートシステムが LPP を UDP と TCP のどちらを使ってサポートしているかを発見するには次の二つの方法がある。

- directory service のようなものを使う
- リモートアプリケーション entity と association を確立しようと 試みる
  - 望んだ quality of service が駄目なら、ローカルの presentation-service-provided が
    negative P-CONNECT.CONFIRMATION primitive を返す

また次のウェルノウンポート番号を用いる。

```
cmot manager     163/tcp
cmot manager     163/udp
cmot agent       164/tcp
cmot agent       164/udp
```

UDP が使われる際には、484 オクテットを越える lightweight presentation PDU を受け取る必要はない。これはフラグメント化を防ぐためである。

第 7 章

管理プロトコル：SNMP

7.1 はじめに

SNMP は Simple Network Management Protocol の略で、TCP/IP ベースの internet を管理するためにすぐに使える短期的な解決として、IAB により実装を奨励 (recommended) された標準プロトコル (standard protocol) である。プロトコルの詳細については RFC-1157 で規定されている。

7.2 SNMP のアーキテクチャ

SNMP のアーキテクチャモデルはネットワーク管理ステーションとネットワーク要素から構成される。

- ネットワーク管理ステーションは、ネットワーク要素を監視し、制御するための管理アプリケーションを実行する。
- ネットワーク要素とは、ホスト、ゲートウェイ、ターミナルサーバのようなデバイスを指す。
- ネットワーク管理エージェントは、ネットワーク要素上で、管理ステーションから要求されるネットワーク管理機能の実行を司る。

SNMP は、ネットワーク管理ステーションとネットワーク要素上のエージェントが管理情報をやりとりするために使われる。

7.2.1 アーキテクチャの目標

SNMP のアーキテクチャは、次の 3 つの目標のもとに構築された。

1. 管理エージェント自体によって実現される管理機能の数や複雑さを最小化する。
2. ネットワークオペレーションや管理の、付加的でおそらく予測できないような状況に対応できるように、監視や制御の機能面での方法論に十分な拡張性を持たせる。

3. プロトコルのアーキテクチャを、できる限り特定のホストやゲートウェイのアーキテクチャやメカニズムとは独立にする。

7.2.2　アーキテクチャの構成要素

7.2.2.1　管理情報の範囲

SNMP のオペレーションによってやりとりされる管理情報はつぎの 2 つである。

1. インターネット標準 MIB[1]
2. インターネット標準 SMI に従って定義された、全ての非集合型オブジェクト

7.2.2.2　管理情報の表現

SNMP のオペレーションによってやりとりされる管理情報は、次の制約にしたがって表現され、コード化される。

- SMI において、非集合型オブジェクトの定義のために指定された ASN.1 言語のサブセットを用いて表現される。
- SNMP は ASN.1 の基本コード化規則のサブセットのみを用いる。
  - 全てのコード化においては、一定長形式を用いる。
  - できる限り構造的コード化よりもむしろ非構造的コード化が用いられる。

  この制約は、最上位のプロトコルデータユニットからそれらに含まれるデータオブジェクトまで、ASN.1 のコード化の全ての局面において適用される。

7.2.2.3　管理情報に対する操作

- SNMP においてサポートされる管理エージェントの機能は次の 2 つである。
  - 変数値の変更
  - 変数値の読みだし
- 動作の実行の命令[2]は、動作の引金にとなるパラメータ値の設定により間接的に行う。

  例 システムのリブート→リブートまでの秒数を示すパラメータをセット
- SNMP によるネットワーク監視の基本戦略
  1. 管理センター側から適当な間隔で情報をポーリングする。
  2. トラップメッセージ 7.3.2によりポーリングのタイミングや焦点を変化させる。

[1] MIB の集合型オブジェクトは、SNMP ではサポートされない。
[2] SNMP では明示的に定義されてはいない。

- トラップメッセージの数は次の２つの理由で最小限に制限されている。

  1. プロトコルの簡潔さを保つ。
  2. ネットワーク管理機能によって生成されるトラッフィックの総量を最小化する。

7.2.2.4 プロトコルメッセージの交換の形態と意味

- 管理エンティティ間の管理情報の交換はプロトコルメッセージの交換を通して行う。
- SNMP メッセージの交換は、管理エージェントの複雑さを最小化するために、
  - 信頼性のないデータグラムサービス (例えば UDP プロトコル) のみを利用する。
  - 全てのメッセージは完全かつ独立に単一のトランスポートデータグラムによって表される。

7.2.2.5 管理上の関係の定義

SNMP のアーキテクチャは、プロトコルに参加するエンティティ間に様々な管理上の関係を認めている。

- SNMP アプリケーションエンティティ

  SNMP を使ってお互いに通信し合う、管理ステーションやネットワーク構成要素上にあるエンティティ。

- プロトコルエンティティ

  SNMP を実装し、SNMP アプリケーションエンティティをサポートする同僚プロセス。

- SNMP コミュニティー

  ある SNMP エージェントと SNMP アプリケーションエンティティの任意の集合との組。各 SNMP コミュニティーは、コミュニティー名を持つ。

- 真正 SNMP メッセージ

  メッセージ中に書かれている SNMP コミュニティーに、本当に属している SNMP アプリケーションエンティティから発せられたメッセージ。

- 認証機構

  SNMP メッセージが特定のコミュニティーの真正 SNMP メッセージとして識別されるのに使われるルールの集合。

- 認証サービス

1 つまたはそれ以上の認証機構に従って、真正 SNMP メッセージを識別する機能を実現したもの。[3]

- SNMP MIB ビュー

  あるネットワーク要素に関係のある MIB 上のオブジェクトの部分集合。SNMP MIB ビューに表されているオブジェクトのタイプの名前は、オブジェクトタイプの名前空間中の単一の部分木に属している必要はないことに注意。

- SNMP アクセスモード

  READ-ONLY または READ-WRITE のいずれか。

- SNMP コミュニティープロフィール

  SNMP アクセスモードと SNMP MIB ビューの組。
  与えられた SNMP コミュニティープロフィールの MIB ビューに属する全ての変数に対して、

  1. 与えられた変数が、MIB において、”Access: none ”と定義されているならば、どんなオペレータのオペランドとしても利用できない。
  2. 与えられた変数が、MIB において、”Access: read-write ”または、”Access: write-only ”と定義されており、また、与えられたプロフィールのアクセスモードが READ-WRITE ならば、その変数は get、set、trap の操作のオペランドとして利用可能である。
  3. それ以外ならば、その変数は、get、trap の操作のオペランドとして利用可能である。
  4. ”write-only ”な変数が get や trap の操作のオペランドとして用いられた場合は、返される変数の値は、インプリメンテーションに依存する。

- SNMP アクセスポリシー

  SNMP コミュニティーと SNMP コミュニティープロフィールの組。
  アクセスポリシーは、特定の SNMP コミュニティーの SNMP エージェントがそのコミュニティーに属する他のメンバーに与える特定のコミュニティープロフィールを表す。

  SNMP アプリケーションエンティティ間の全ての管理上の関係は SNMP アクセスポリシーの点から構造的に定義される。

[3] 真正 SNMP メッセージを（暗号等の技術を用いて）高い信頼性の元に識別する認証サービスが要求されている。しかし、現時点では、全ての SNMP メッセージを真正 SNMP メッセージと見なすような”trivial”な認証サービスしかサポートされていない。

- SNMP 代理アクセスポリシー

  あるネットワーク要素上に、ある SNMP コミュニティーに属するエージェントが存在するとする。そして、そのエージェントが属するコミュニティーの MIB ビューが、そのネットワーク要素とは関係ないものであるとする。この時、そのエージェントが提供しているアクセスポリシーのことを代理アクセスポリシーと呼ぶ。

  代理アクセスポリシーは、次のような特徴を持つ。

  1. 不注意に定義すると、管理のループを招く。
  2. 管理プロトコルやトランスポートプロトコルを用いて通信できないネットワーク構成要素（モデムや多重化装置等）を監視し、制御することを可能にする。
  3. ネットワーク構成要素が、手の込んだアクセス制御ポリシを持つことを避ける。例えば、代理エージェントは、ネットワーク構成要素の複雑さを増やすことなく、MIB 中の変数の多様な部分集合が異なる管理ステーションからアクセス可能とするための洗練されたアクセス制御を実現する。

- SNMP 代理エージェント

  代理アクセスポリシーに関係のある SNMP エージェント。

7.2.2.6 管理されるオブジェクトに対する参照の形式と意味

SMI では、管理プロトコルの定義は次の点に焦点を当てたものであることを要求している。

- 曖昧な MIB の参照の解決
- 複数の MIB のバージョンが存在する場合の MIB の参照の解決
- MIB で定義されたオブジェクトのタイプの特定のインスタンスの識別

SNMP では、これら３つの要求に対して次のように答えている。

1. 曖昧な MIB の参照の解決

   次の理由により、MIB で定義されたどのオブジェクトタイプに対するどんな SNMP の参照もそのタイプの複数のインスタンスに解決することはあり得ないことが分る。

   - どんな SNMP の操作も、概念的には単一のネットワーク要素に関係のあるオブジェクトに制限される。
   - MIB オブジェクトに対する、全ての SNMP の参照は唯一の変数名によって行われる。

2. MIB のバージョンにまたがる参照の解決

   どんな SNMP の操作により参照されるオブジェクトのインスタンスも、

   - 操作要求の一部として記述されたもの
   - MIB 中でその直後にあるもの (get-next 操作の場合)。

   のいずれかである。

   従って、Internet 標準 MIB のあるバージョンの一部としてのオブジェクトへの参照は、次の 2 つが満たされる場合を除いて、それ以外のどんなバージョンのオブジェクトにも解決しない。

   (a) 要求された操作が get-next である。

   (b) 指定されたオブジェクトの名前が辞書順序的に Internet 標準 MIB の指定されたバージョンの一部として表された全てのオブジェクトの名前の中で最後のものである。

3. オブジェクトのインスタンスの識別

   MIB に含まれる全てのオブジェクトのタイプの名前は、

   - Internet 標準 MIB
   - SMI の名前づけの慣習に則った他のドキュメント

   のいずれかによって明示的に定義される。

   MIB で定義されたすべてのオブジェクトタイプのインスタンスは、ユニークな ”変数名 ”により識別される。この変数名の付け方はオブジェクトタイプ毎に決められる。

   - ifTable オブジェクトタイプ

     ifTable は、そのエンティティが持つインタフェースに関する情報の一覧表を表すオブジェクトタイプである。インタフェース毎に ifEntry という、そのインターフェースに関する情報をまとめたオブジェクト (一覧表の” 行” に相当する。) が定義され、全てのインタフェースのための ifEntry をまとめて ifTable が構成される。

     以下、*Table という名前のオブジェクトは全てこれと同様な構成を持つ。

     さて、ifEntry オブジェクトタイプを構成する各オブジェクト (ifIndex、ifDescr 等) のインスタンスを表す変数名は、

     オブジェクトタイプ名.⟨ インタフェース番号 ⟩

     という形式である。ただし、⟨ インタフェース番号 ⟩ は、ifIndex オブジェクトタイプのインスタンス値に一致しなければならない。

- atTable オブジェクトタイプ

  atTable は、ネットワークアドレスと物理アドレスの対応表[4]を表すオブジェクトタイプである。

  atEntry オブジェクトタイプを構成する各オブジェクト (atIfIndex、atPhysAddress 等) のインスタンスを表す変数名は、

  オブジェクトタイプ名.⟨ インタフェース番号 ⟩.1.⟨IP アドレス ⟩

  という形式である。ただし、⟨ インタフェース番号 ⟩ は、atIndex オブジェクトタイプのインスタンス値に一致していなければならない。また、⟨IP アドレス ⟩ は、そのインタフェースに付けられた IP アドレスである。

- ipAddrTable オブジェクトタイプ

  ipAddrTable は、そのエンティティが持つ IP アドレスに関する情報の一覧表を表すオブジェクトタイプである。

  ipAddrEntry オブジェクトタイプを構成する各オブジェクト (ipAdEntAddr、ipAdEntIfIndex 等) のインスタンスを表す変数名は、

  オブジェクトタイプ名.⟨IP アドレス ⟩

  という形式である。ただし、⟨IP アドレス ⟩ は、ipAdEntAddr オブジェクトタイプのインスタンス値に一致していなければならない。

- ipRoutingTable オブジェクトタイプ

  ipRoutingTable は、ルーティングテーブルを表すオブジェクトタイプである。

  ipRoutingEntry オブジェクトタイプを構成する各オブジェクト (ipRouteDest、ipRouteIfIndex 等) のインスタンスを表す変数名は、

  オブジェクトタイプ名.⟨IP アドレス ⟩

  という形式である。ただし、⟨IP アドレス ⟩ は、ipRouteDest オブジェクトタイプのインスタンス値に一致していなければならない。

- tcpConnTable オブジェクトタイプ

  tcpConnTable は、TCP のコネクションに関する情報の一覧表を表すオブジェクトタイプである。

  tcpConnEntry オブジェクトタイプを構成する各オブジェクト (tcpConnState、tcpConnLocalAddress 等) のインスタンスを表す変数名は、

[4] 例えば ARP テーブル

オブジェクトタイプ名.⟨IP アドレス 1⟩.⟨ ポート番号 1⟩.⟨IP アドレス 2⟩.⟨ ポート番号 2⟩

という形式である。ただし、

- ⟨IP アドレス 1⟩ ··· tcpConnLocalAddress オブジェクトタイプのインスタンス値
- ⟨ ポート番号 1⟩ ··· tcpConnLocalPort オブジェクトタイプのインスタンス値
- ⟨IP アドレス 2⟩ ··· tcpConnRemoteAddress オブジェクトタイプのインスタンス値
- ⟨ ポート番号 2⟩ ··· tcpConnRemotePort オブジェクトタイプのインスタンス値

にそれぞれ一致していなければならない。

- egpNeighTable オブジェクトタイプ

  egpNeighTable は、EGP 近隣に関する情報の一覧表を表すオブジェクトタイプである。

  egpNeighEntry オブジェクトタイプを構成する各オブジェクト (egpNeighState、egpNeighAddr) のインスタンスを表す変数名は、

  オブジェクトタイプ名.⟨IP アドレス ⟩

  という形式である。ただし、⟨IP アドレス ⟩ は、egpNeighAddr オブジェクトタイプのインスタンス値に一致していなければならない。

- その他

  以上のいずれでもないオブジェクトタイプのインスタンスを表す変数名は、

  オブジェクトタイプ名.0

  という形式である。

7.3　プロトコル仕様

SNMP は次のような特徴をもつプロトコルである。

- ネットワーク管理プロトコルはエージェントの MIB 変数が検査されたり変更されたりするようなアプリケーションプロトコルである。
- プロトコルエンティティ間の通信は、メッセージの交換によって行われる。

- 各メッセージは ASN.1 の基本的なコード化ルールを持ちいて、完全かつ独立に、単一の UDP データグラム中に表現される。
- プロトコルエンティティには、次の 2 つの UDP ポート番号が割り当てられている。
  - **161 番** ··· トラップメッセージを除く全てのメッセージ受信用
  - **162 番** ··· トラップメッセージ受信用
- プロトコルのインプリメンテーションは 484 オクテット長を越えるメッセージを受け取る必要はない。[5]
- SNMP の全てのインプリメンテーションは次の 5 つの PDU をサポートすることが不可欠である。
  - GetRequest-PDU
  - GetNextRequest-PDU
  - GetResponse-PDU
  - SetRequest-PDU
  - Trap-PDU
- 一つのメッセージは、次の 3 つの要素から構成される。
  1. バージョン識別子
  2. SNMP コミュニティー名
  3. プロトコルデータ単位（PDU）

7.3.1 メッセージ処理手続

SNMP のメッセージを送受信する際の処理手続きは次のようである。

- メッセージ送信側のプロトコルエンティティのトップレベルの動作
  1. PDU を ASN.1 のオブジェクトとして構成する。
  2. この ASN.1 のオブジェクトを、コミュニティー名と起点トランスポートアドレスと終点トランスポートアドレスとともに認証サービスに渡す。そして、認証サービスは必要な処理を加えた結果作られた ASN.1 オブジェクトを返す。
  3. コミュニティー名と、上の結果得られた ASN.1 オブジェクトから ASN.1 のメッセージオブジェクトを構成する。

[5] 可能な時にはいつでもより大きなデータグラムをサポートすることが奨励される。

4. 新しく得られた ASN.1 オブジェクトを、ASN.1 の基本コード化ルールを用いてビット列に変換し、トランスポートサービスに送り出す。

- メッセージ受信側のプロトコルエンティティのトップレベルの動作

1. 受け取ったデータグラム中のビット列から ASN.1 メッセージオブジェクトを再構成する。もし失敗したならば、そのデータグラムを捨てそれ以上の処理は行わない。
2. SNMP メッセージのバージョン番号を確認する。もしバージョン番号が一致していなければ、そのデータグラムを捨てそれ以上の処理は行わない。
3. ASN.1 メッセージオブジェクト中のコミュニティー名とユーザデータをデータグラムの起点と終点のトランスポートアドレスとともに認証サービスに渡す。この認証サービスは、別の ASN.1 オブジェクト、もしくは認証失敗のシグナルを返す。後者の場合にはこの失敗を記録し、トラップを生成し、そのデータグラムを捨て、それ以上の処理は行わない。
4. 認証サービスから返された ASN.1 オブジェクトから PDU オブジェクトを再構成する。もし失敗したら、そのデータグラムを捨てそれ以上の処理は行わない。そうでなければ、指定された SNMP コミュニティーに対するプロファイルにしたがって PDU を処理する。

7.3.2 PDU

Trap-PDU 以外の PDU は、次の 4 つの要素から構成される。

- request-id
- error-status
- error-index
- variable-bindings

それぞれの要素は、次のような意味を持つ。

- request-id は要求と応答を対応させるために用いられる整数値である。また、信頼性の ないデータグラムサービスが用いられるような場合には、二重化され たメッセージを識別するためにも用いることができる。
- error-status は、どんなエラーが起こったかを示す整数値である。それぞれの値とそれに与えられたニーモニックは、
  - 0 ··· noError
  - 1 ··· tooBig

- 2 ··· noSuchName
- 3 ··· badValue
- 4 ··· readOnly
- 5 ··· genErr

である。

- error-index は、エラーがどこで起こったかを示す。
- variable-bindings は変数名とそれに対応する値との単純なリストである。変数の名前のみが考慮され、その値は考慮されないような PDU もある（例えば、GetRequest-PDU）。このような場合、プロトコルエンティティは変数束縛の値の部分を無視する。しかし、値の部分は ASN.1 の文法に正しく従い、コード化されねばならない。このような変数束縛の値の部分に対して、ASN.1 の NULL 値を用いることが推奨される。

7.3.2.1 GetRequest-PDU

GetRequest-PDU は、SNMP エージェントから指定した変数の値を検索する時に用いられる。変数の指定は、variable-vindings フィールドに変数名だけ（値の部分は 0 にする）を与えることにより行う。

- GetRequest-PDU を受けとると、受信側のプロトコルエンティティは以下のルールに従って応答を行う。

  1. variable-bindings フィールド中のどのオブジェクトに対しても、もし、関係する MIB view の中で get 操作が利用可能などのオブジェクトの名前も完全に一致しなかったならば、受け取り側のエンティティはメッセージの送り側に、error-status フィールドの値が noSuchName であり、error-index フィールドの値が受け取ったメッセージの中のそのオブジェクト名を指し示していることを除いて、受け取ったメッセージと同じ形の GetResponse-PDU を送る。
  2. variable-bindings フィールド中のどのオブジェクトに対しても、もし、そのオブジェクトが（SMI で定義されている）集合型であるならば、受け取り側のエンティティはメッセージの送り側に、error-status フィールドの値が noSuchName であり、error-index フィールドの値が受け取ったメッセージの中のそのオブジェクト名を指し示していることを除いて、受け取ったメッセージと同じ形の GetResponse-PDU を送る。
  3. もし、下に示すようにして生成された GetResponse-PDU の大きさが、ローカルな制限値を越えたならば、受け取り側のエンティティはメッセージの送り側に、error-status フィールドの値が tooBig であり、error-index フィールドの値が 0 であることを除いて、受け取ったメッセージと同じ形の GetResponse-PDU を送る。

4. variable-bindings フィールド中のどのオブジェクトに対しても、もし、上述のルールのどれにも当てはまらないような理由からオブジェクトの値が検索できなかったならば、受け取り側のエンティティはメッセージの送り側に、error-status フィールドの値が genErr であり、error-index フィールドの値が受け取ったメッセージの中のそのオブジェクト名を指し示していることを除いて、受け取ったメッセージと同じ形の GetResponse-PDU を送る。

- 上述のどのルールも適用されないならば、受け側のプロトコルエンティティは受け取ったメッセージの送り側に、受け取ったメッセージの variable-bindings フィールドの中の記された各オブジェクトに対して、対応する部分がその変数の名前と値を表しているような GetResponse-PDU を送る。GetResponse-PDU の error-status フィールドの値は、noError であり、また、error-index フィールドの値は 0 である。GetResponse-PDU の request-id フィールドは受け取ったメッセージのものと同じ値である。

7.3.2.2 GetNextRequest-PDU

GetNextRequest-PDU は、GetRequest-PDU と同様に変数値の検索に用いられるが、variable-bindings フィールドに与えた変数名を持つ変数の値ではなく、MIB においてその変数の辞書的順序で次にある変数の値を要求する。

- GetNextRequest-PDU を受けとると、受信側のプロトコルエンティティは以下のルールに従って応答を行う。

1. variable-bindings フィールド中のどのオブジェクトに対しても、もし、そのオブジェクトの名前が、関係する MIB view の中で get 操作が利用可能などのオブジェクトの辞書的順序で次に位置付けられる名前にも完全に一致しなかったならば、受け取り側のエンティティはメッセージの送り側に、error-status フィールドの値が noSuchName であり、error-index フィールドの値が受け取ったメッセージの中のそのオブジェクト名を指し示していることを除いて、受け取ったメッセージと同じ形の GetResponse-PDU を送る。
2. もし、下に示すようにして生成された GetResponse-PDU の大きさが、ローカルな制限値を越えたならば、受け取り側のエンティティはメッセージの送り側に、error-status フィールドの値が tooBig であり、error-index フィールドの値が 0 であることを除いて、受け取ったメッセージと同じ形の GetResponse-PDU を送る。
3. variable-bindings フィールド中のどのオブジェクトに対しても、もし、上述のルールのどれにも当てはまらないような理由から指定されたオブジェクトの辞書的順序で次に位置付けられるオブジェクトの値が検索できなかったならば、受け取り側のエンティティはメッセージの送り側に、error-status フィールドの値が genErr であり、error-index フィールドの値が受け取ったメッセージの中の

そのオブジェクト名を指し示していることを除いて、受け取ったメッセージと同じ形の GetResponse-PDU を送る。

- 上述のどのルールも適用されないならば、受け側のプロトコルエンティティは受け取ったメッセージの送り側に、受け取ったメッセージの variable-bindings フィールドの中に記された各オブジェクトの名前に対して、対応する部分が、関連する MIB view において get 操作がアクセス可能な全てのオブジェクトの名前の辞書的順序において、その値が直後に位置付けられるようなオブジェクトの名前と値を表した GetResponse-PDU を送る。GetResponse-PDU の error-status フィールドの値は、noError であり、また、error-index フィールドの値は 0 である。GetResponse-PDU の request-id フィールドは受け取ったメッセージのものと同じ値である。

- GetRequest-PDU ではなく GetNextRequest-PDU を利用することにより、MIB のテーブルオブジェクトの各エントリの持つ値を次々に検索することが容易になる。

  例えば、下記のような SNMP メッセージのやりとりによって、SNMP アプリケーションエンティティは特定のネットワーク構成要素のルーティングテーブルを得ることができる。このルーティングテーブルは 3 つのエントリを持つものとする。

Destination	NextHop	Metric
10.0.0.99	89.1.1.42	5
9.1.2.3	99.0.0.3	3
10.0.0.51	89.1.1.42	5

  – 管理ステーションはエージェントに次の GetNextRequest-PDU を送る。

  ```
  GetNextRequest (   ipRouteDest, ipRouteNextHop, ipRouteMetric1 )
  ```

  – エージェントは次の GetResponse-PDU により応答する。

  ```
  GetResponse (   ( ipRouteDest.9.1.2.3 = "9.1.2.3" ),
                  ( ipRouteNextHop.9.1.2.3 = "99.0.0.3" ),
                  ( ipRouteMetric1.9.1.2.3 = 3 ))
  ```

  – 管理ステーションは次のように続ける。

  ```
  GetNextRequest (   ipRouteDest.9.1.2.3,
                     ipRouteNextHop.9.1.2.3,
                     ipRouteMetric1.9.1.2.3 )
  ```

  – エージェントは、次のように応答する。

  ```
  GetResponse (   ( ipRouteDest.10.0.0.51 = "10.0.0.51" ),
                  ( ipRouteNextHop.10.0.0.51 = "89.1.1.42" ),
                  ( ipRouteMetric1.10.0.0.51 = 5 ))
  ```

  – 管理ステーションは次のように続ける。

GetNextRequest (ipRouteDest.10.0.0.51,
ipRouteNextHop.10.0.0.51,
ipRouteMetric1.10.0.0.51)

– エージェントは、次のように応答する。

GetResponse ((ipRouteDest.10.0.0.99 = ”10.0.0.99”),
(ipRouteNextHop.10.0.0.99 = ”89.1.1.42”),
(ipRouteMetric1.10.0.0.99 = 5))

– 管理ステーションは次のように続ける。

GetNextRequest (ipRouteDest.10.0.0.99,
ipRouteNextHop.10.0.0.99,
ipRouteMetric1.10.0.0.99)

– この時、テーブルにもうこれ以上のエントリがないので、エージェントは MIB の中で辞書的順序で次にあるオブジェクトの名前を返す。この応答はルーティングテーブルの終りを管理ステーションに伝える信号となる。

- 上記の例のように、GetNextRequest は GetRequest に比べて非常に便利であるため、しばしば”Powerful” GetNextRequest と呼ばれる。

7.3.2.3 GetResponse-PDU

- GetResponse-PDU は、管理エージェントが要求された変数の値を管理ステーションに返す時に用いられる。
- GetResponse-PDU は、GetRequest-PDU や GetNextRequest-PDU や SetRequest-PDU を受けとった時のみ、プロトコルエンティティによって生成される。
- GetResponse-PDU を受けとると、受け取り側のプロトコルエンティティはその中身を SNMP アプリケーションエンティティに与える。

7.3.2.4 SetRequest-PDU

- SetRequest-PDU は、管理エージェントに指定した変数の値をセットすることを要求するために用いられる。
- SetRequest-PDU を受けとると、受信側のプロトコルエンティティは以下のルールに従って応答を行う。

1. variable-bindings フィールド中のどのオブジェクトに対しても、もし、そのオブジェクトが、関係する MIB view の中で set 操作が利用可能でなかったならば、受け取り側のエンティティはメッセージの送り側に、error-status フィールドの値が noSuchName であり、error-index フィールドの値が受け取ったメッセージの中のそのオブジェクト名を指し示していることを除いて、受け取ったメッセージと同じ形の GetResponse-PDU を送る。

2. variable-bindings フィールド中のどのオブジェクトに対しても、もし、その値のフィールドが、ASN.1 言語にしたがって、その変数に対して要求されるタイプ、長さ、値であることが明らかにされなかったならば、け取り側のエンティティはメッセージの送り側に、error-status フィールドの値が badValue であり、error-index フィールドの値が受け取ったメッセージの中のそのオブジェクト名を指し示していることを除いて、受け取ったメッセージと同じ形の GetResponse-PDU を送る。
3. もし、下に示すようにして生成された Get Response タイプのメッセージの大きさが、ローカルな制限値を越えたならば、受け取り側のエンティティーはメッセージの送り側に、error-status フィールドの値が tooBig であり、error-index フィールドの値が 0 であることを除いて、受け取ったメッセージと同じ形の GetResponse-PDU を送る。
4. variable-bindings フィールド中のどのオブジェクトに対しても、もし、上述のルールのどれにも当てはまらないような理由から指定されたオブジェクトの値を変更できなかったならば、受け取り側のエンティティはメッセージの送り側に、error-status フィールドの値が genErr であり、error-index フィールドの値が受け取ったメッセージの中のそのオブジェクト名を指し示していることを除いて、受け取ったメッセージと同じ形の GetResponse-PDU を送る。

- 上述のどのルールも適用されないならば、受け取ったメッセージの variable-bindings フィールドの中に記された各オブジェクトに対して、対応する値がその変数に代入される。SetRequest-PDU により指定された各変数の代入は同じメッセージの中に記述された他の全ての代入に関してあたかも同時であるかのごとく実行されるべきである。そして、受け取り側のエンティティは、受け取ったメッセージの送り側に error-status フィールドの値が noError であり、また、error-index フィールドの値は 0 である以外は同一の形の GetResponse-PDU を送る。

7.3.2.5 Trap-PDU

- トラップ PDU は、次の６つの要素から構成される。
  - enterprize

    トラップ PDU を発生したネットワーク要素のタイプ。sysObjectID として登録された値を用いる。
  - agent-addr

    トラップ PDU を発生したネットワーク要素のアドレス。
  - generic-trap

    一般的なトラップの種類を表す整数値。各値が何を意味するかは後述。

- specific-trap

  企業毎に決められた、その企業特有のトラップの種類を表す整数値。

- time-stamp

  ネットワークエンティティが最後にリセットされてからトラップが発生するまでの時間

- variable-bindings

  トラップに関係のある変数の名前と値の組

- SNMP アプリケーションエンティティが Trap-PDU を送る相手先のアドレスを選択する方法はインプリメンテーションにごとに決められる。

- Trap-PDU を受け取ると、受け側のプロトコルエンティティはその内容を SNMP アプリケーションエンティティに渡す。

- Trap-PDU の variable-bindings 部分の意味は、インプリメンテーションごとに決められる。

- generic-trap フィールドの値の解釈は、

  1. coldStart(0)

     送り側のプロトコルエンティティがエージェントのコンフィギュレーションやプロトコルエンティティのインプリメンテーションが変更されるように、自分自身を再初期化していることを表す。

  2. warmStart(1)

     送り側のプロトコルエンティティが agent のコンフィギュレーションやプロトコルエンティティのインプリメンテーションを変更することなく、自分自身を再初期化していることを表す。

  3. linkDown(2)

     送り側のプロトコルエンティティが、agent のコンフィギュレーション中に表されている通信リンクの一つに誤りを認識したことを表す。linkDown のタイプの Trap-PDU は、その variable-bindings の最初の要素として、影響を受けたインタフェースの ifIndex のインスタンスの名前と値を持つ。

  4. linkUp(3)

     送り側のプロトコルエンティティが、agent のコンフィギュレーション中に表されている通信リンクの一つが立ち上がったことを認識したことを表す。linkUp のタイプの Trap-PDU は、その variable-bindings の最初の要素として、影響を受けたインタフェースの ifIndex　のインスタンスの名前と値を持つ。

5. authenticationFailure(4)

   送り側のプロトコルエンティティが正しく認証されないプロトコルメッセージの受取手であったことを表す。SNMP のインプリメンテーションはこのトラップを発生することができなければならないが、インプリメンテーションごとに定められる機構を通じて、このようなトラップの放出を抑制することもまたできなければならない。

6. egpNeighborLoss(5)

   送り側のプロトコルエンティティが EGP 同僚であったような EGP 隣接がダウンしもはや同僚関係が保てないということを表す。
   egpNeighborLoss タイプの Trap-PDU は、その variable-bindings の最初の要素として、影響する隣接の egpNeighAddr のインスタンスの名前と値を持つ。

7. enterpriseSpecific(6)

   送り側のプロトコルエンティティがある、企業ごとに決められたイベントが発生したということを認識したことを表す。specific-trap フィールドが、発生した特定のトラップを識別する。

第 8 章

広域分散環境におけるネットワーク管理

8.1 はじめに

コンピュータネットワークでは複数の計算機が相互に接続され、情報の受け渡しが行われている。このネットワーク全体を正常に運用するためには、相互接続されている計算機のそれぞれが全て正しく経路を制御し情報の伝達を行わなければならない。そのためにネットワークに参加している全ての計算機に対して管理を行う必要がある。今までは、数人のネットワーク管理者がネットワーク全体の構成を理解し管理を行なってきたが、ネットワークの規模が拡大するにつれ、人手によるネットワーク管理には限界があることが明らかになってきている。例えばネットワークの管理のうち、特に障害から復旧するためにはネットワーク全体に関する高度な知識が要求される。今までは数人の管理者だけが所有する知識を基本として障害復旧を行なってきていたが、管理者が不在のときなどの障害には対処できなくなってきているのが実状である。

このような背景により、近年ネットワーク管理に関する研究が盛んに行なわれるようになってきた。そして、ネットワーク管理に利用できる標準的な管理プロトコルと、それを実装したツールが開発されてきている。SNMP などに代表されるネットワーク管理プロトコルは、このような背景によって開発されたネットワーク管理プロトコルである。

また、広域ネットワークでは複数のネットワークが相互接続されれおりそれら相互の接続の状況などネットワーク全体の構成はより複雑になっている。そこで、ここでの管理を考えた場合、ローカルエリアネットワークを管理する場合に比べて非常に多くの知識が必要であるため、全体の管理は困難になってきている。実際に WIDE インターネットでも、管理が重要な問題となっている。

ここでは、ネットワーク管理、特に広域ネットワークの管理について論じる。

8.2 広域ネットワークにおける SNMP プロトコルの問題点

ネットワーク管理プロトコルのうち、インターネットで現在実装されているのは TCP/IP プロトコルに従ったネットワーク管理用の標準的なプロトコルとされている SNMP である。これは管理情報データベース (MIB) に定義された管理情報を扱い、MIB での定義は SMI に定義された構造に従っている。

SNMP の基本的な考え方と操作は、以下の通りである。

- 一つのネットワーク管理ステーションに対して複数のエージェントが存在する。(図8.1参照)

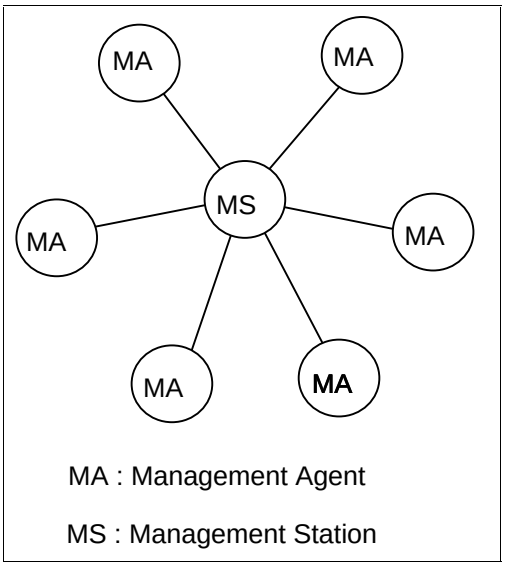


図 8.1: SNMP の管理モデル

- エージェントは管理情報を持つ。
- ステーションはエージェントの持つ MIB に定義さたネットワーク情報に対してその値の “獲得” と “設定” の二つの操作を行う。
- 設定されている条件に適合する事象の発生により、エージェントは管理ステーションに向けてメッセージを送る。
- SNMP プロトコルが適用できないデバイス (モデム等) に対し、それに代わって答える代理エージェントを設定できる。

このように、現在の SNMP などに代表されるネットワーク管理プロトコルは、ネットワーク接続されている全てのネットワークデバイスを一括して管理することによってネットワーク全体の管理を行う、という考え方になっている。しかし、広域分散環境におけるネットワーク管理にこのような方法は適していない。

理由として、まず第一に管理組織の違いが挙げられる。広域分散環境とは複数のネットワークが相互に接続されたネットワークであり、その管理組織はそれぞれの組織に委ねられている。よって、広域ネットワークに接続されている全ての計算機の組織をまたがった直接的な管理は不可能である。

第二に、管理情報の通信量が問題となる。一つの管理ステーションが直接管理しあているエージェントの数が少ない場合、ステーションがエージェントである全ての計算機の

情報を獲得することは可能である。しかし、仮に広域分散環境で一つの管理ステーションがネットワーク上の全ての計算機を直接に一元的に管理すると、管理情報を受け渡しするためネットワーク上を流れるデータ量が膨大になることが予想される。

第三に、ネットワークの規模が拡大し構成なども複雑となると、広域分散環境での一元的なネットワーク管理は不可能である。つまり、広域ネットワーク管理の場合、複数のネットワークを管理するので、ネットワークに接続されている計算機一つ一つの情報よりも、それぞれのネットワークの概要をとらえる情報が必要である。例えば、SNMP ではあるゲートウェイに入ってくるパケットの数を調べることは可能だが、複数のゲートウェイ間で受け渡しされるパケットの数は、それぞれのゲートウェイで得られた数を処理しなければならない。WIDE インターネットで、それぞれのバックボーンノード間で受け渡しされるパケットの数を調べるとする。SNMP で東京のバックボーンノードの管理情報を得た場合、そのバックボーンに出入りするパケットの数は直接得られるが、東京のバックボーンノードから大阪のバックボーンノードへ向かうパケット数、または東京のバックボーンノードから九州のバックボーンノードに向かうパケット数などは直接得られない。

また、あるネットワーク内の計算機に障害が発生した場合、複数のネットワークを管理するためには、最終的にはそれが広域ネットワーク全体に与える影響が必要なのである。あるネットワーク中に発生した障害は、他のネットワークにとって自分が影響を受けずに解決されることが重要なのであって、その詳しい原因まで理解する必要はない。

SNMP を用いながらこれらの問題点を解決する管理方法として、階層的な管理が考えられる。

8.3　代理エージェントを用いた階層的なネットワーク管理

SNMP では、インターネットプロトコルに従っていないために直接管理できないフォーリンデバイスを管理するため、そのデバイスに代わって SNMP プロトコルに従った情報の受け渡しを行う代理エージェントを規定している。つまり代理エージェントとなる計算機は、管理ステーションがデバイスの情報を問い合わせる時、そのデバイスに代わって答える。この代理エージェントに、デバイスの情報以外の情報を持たせることを考えることができる。(図 8.2参照) 広域ネットワークは、複数のネットワークを相互に接続したものである。そこで全体の管理のために、それぞれのネットワークを代表して外からの問い合わせに答えるという代理エージェントを考える。この代理エージェント用いて、ネットワーク情報の階層的な管理を考える。(図 8.3参照) SNMP では一つの管理ステーションが複数の管理されるエージェントに問い合わせるというモデルである (図 8.1)。広域ネットワークの管理をこのまま行うと、1 つの管理ステーションと管理される全てのエージェントとの間で情報の受け渡しが行われることになる。

代理エージェントを設定した場合、図 8.3のようになる。ここで、双方向の矢印で結ばれたステーションと複数のエージェントは、情報の受け渡しを行っている。また、一方向の矢印で結ばれたステーションとエージェントの組は、ステーションが得た情報を処

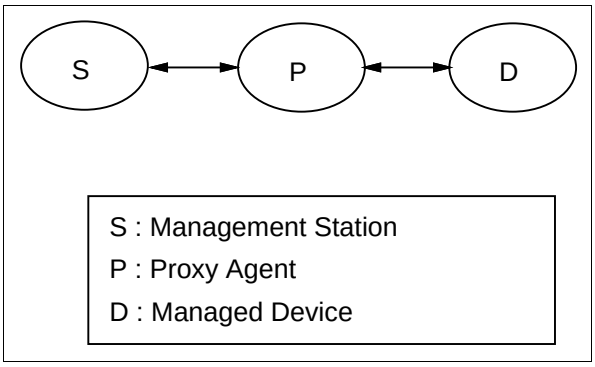


図 8.2: 代理エージェント

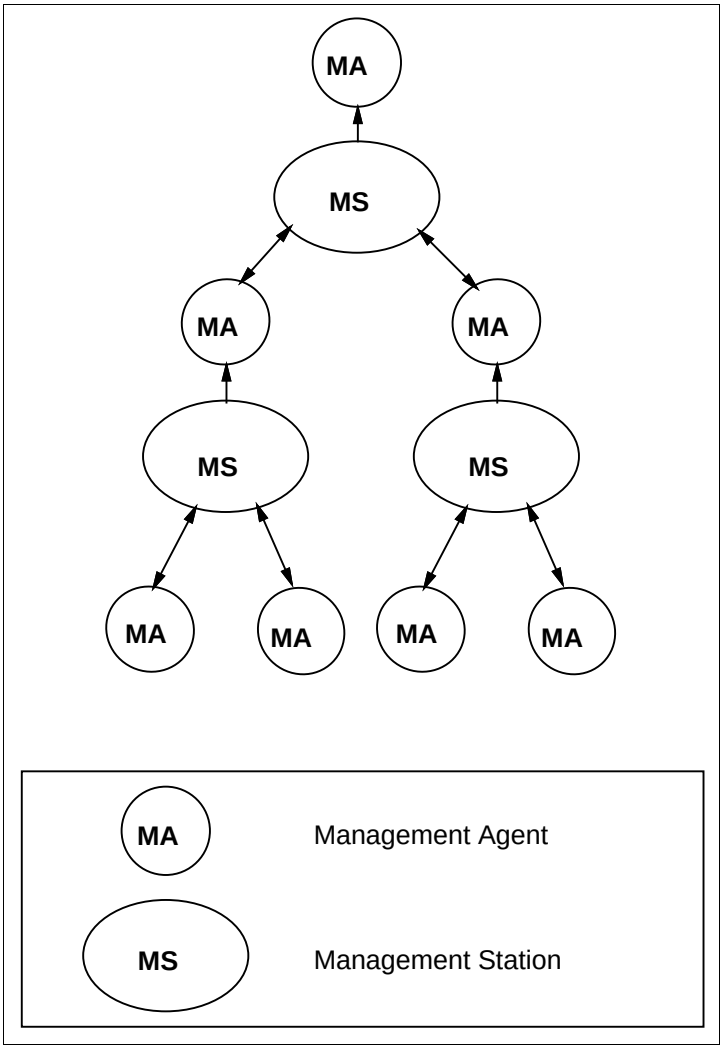


図 8.3: 階層的なエージェント

理した結果を代理エージェントが持つことを示している。このエージェントとステーションの組を一つの計算機で実行すると、管理ステーションがエージェントの立場となって管理されることになる。代理エージェントは、もともとの MIB には定義されていないが広域ネットワーク管理に必要と思われる情報を、管理ステーションが得た情報を処理することによって新たに生成して持つ。

これをある広域ネットワーク上に実現した例を次の図 8.4に示す。ここで図の平面上の

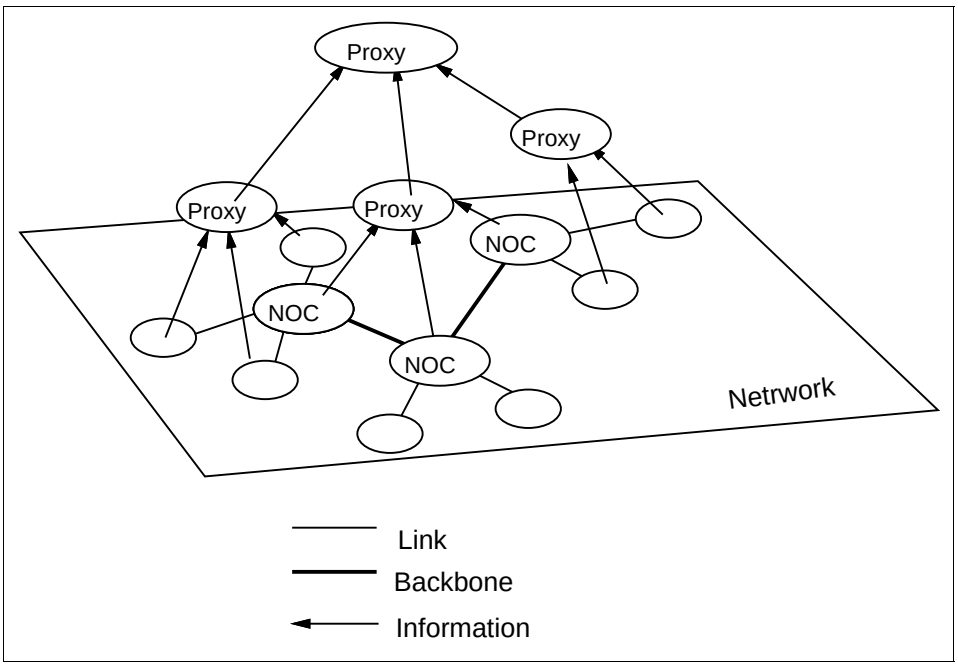


図 8.4: 階層的なエージェントの例

丸が、ネットワーク間を接続しているゲートウェイでありる。'NOC' は太い回線であるバックボーンをつなぐゲートウェイ (バックボーンノード) で、ここに複数のネットワークが接続されている。別々の NOC に接続されているネットワーク間で通信する場合、パケットはそのネットワークのゲートウェイから NOC に渡され、バックボーン回線を通り、 NOC を通りゲートウェイに渡され目的の相手に渡される。この図では代理エージェントが

- NOC に接続されているゲートウェイの情報を処理してできる情報
- 複数の NOC の情報を処理してできた情報
- 上の 2 つの情報さらに処理した情報

を持つことを示している。

例えば、WIDE インターネットにおける代理エージェントの例を考えてみると

- 大学のネットワークと, 会社のネットワークをそれぞれまとめる。

- それぞれのバックボーンノードに接続されたゲートウェイをまとめる。
- バックボーンノード全てをまとめる。

さらに、バックボーンノードに接続されたゲートウェイをまとめた代理エージェントが持つ情報の具体的な例を次に示す。

- バックボーンノードを通るパケットの数
- 最も深いところ (遠いことろ) までのメトリック
- バックボーンノードの下にあるそれぞれのネットワークについて
  - それぞれまで到達可能であるか。
  - ホストの台数。
  - ネットワークの数。
  - サブネットマスク。

バックボーンノード全てをまとめた代理エージェントが持つ情報の例としては次のものが挙げられる。

- あるバックボーンノードから、別のバックボーンノードに対するパケット数
  (その 2 つの間に複数の経路があるなら、それら両方を合わせたもの)

広域ネットワーク全体の管理を行うためには、これらの階層的なネットワーク情報を生成する方法と、代理エージェントが情報を伝える機構を考える必要がある。

8.4 代理エージェントの実装方法

SNMP の標準的なモデルでは、ルータ、ブリッジ、ホスト計算機などのネットワークデバイスは UDP 上で SNMP のメッセージを交換する。UDP/IP でなく、他の種類のプロトコルやトランスポートプロトコルで通信しているデバイスとの間で SNMP メッセージを交換するために、代理管理が必要となる。具体的には、IEEE 802 ネットワーク上の LLC のフレーム、リンクレイヤ上のプロトコルである 802.5 や FDDI などがある。これらを代理エージェントで管理する場合の、アドレスの指定方法が Internet Draft の 'Proxy Via Community' に提案されている。代理エージェントを実装する場合、管理ステーションと、代理エージェントと、管理されるデバイスの間でアドレス情報を共有する方法が問題になる。この解決策として、代理になる SNMP メッセージのコミュニティーストリングの中に明示的にアドレス情報を持つ。この方法は、CPU などのリソースをできるだけ消費せず、できるだけ柔軟な方法を目標としている。SNMP ではコミュニティーネームによって、管理されるデバイスと一つ以上の管理ステーションとの間の管理上の関係

を決める。ここで示された方法は、代理管理を表すためのアドレッシング情報を表すことによって、管理上のポリシを示している。

代理のための情報の存在を示しているコミュニティー ストリングは次の形式で示される。

```
name@tag.address
```

name がコミュニティー名である。tag と address は、どちらも 2 進数で表される。address が代理管理されるデバイスの実際のアドレスを表す。

さらに、tag は、次の 2 つからなる。

```
protocol_type.addredd_type
```

tagのフィールドが両方0なら、代理のアドレッシング情報はないことを示す。protocol_type の値は、代理エージェントが管理されるデバイスに対して用いる管理プロトコルを示す。address_type の値は、管理されるデバイスに送るために用いる実際のアドレスの型を示す。アドレスの長さは address_type の値で決まる。

SNMP の中で @ の記号を代理管理のみに許し、MIB に新たな定義を付け加えることによって代理管理が実現できる。

以上の方法によって、SNMP メッセージを管理ステーションから代理エージェントに、さらにデバイスに送ることができる。これは 1 レベルのメッセージを送っているわけだが、SNMP メッセージを代理ステーションからさらに別の代理ステーションにフォワードして最終的に目的のデバイスに渡す、という複数のレベルから成る代理エージェントを実現したいという要求もある。そこで複数のレベルからなる代理エージェントを用いた管理を行う方法としては、次のように SNMP メッセージのコミュニティーフィールドを続ける。

```
name@Proxy A@Proxy B@.....@Proxy Z@end device
```

実行させる時にはコミュニティーフィールド中の@の数を見て、@が

- ないなら、UDP は代理されておらずステーションは通常通りの UDP として扱う。
- 1 つなら 1 レベルの代理を表すので、アドレスは目的のステーションのものである。
- 2 つ以上なら、次に書いてあるアドレスにフォワード する。

このようにして、マルチレベルの代理エージェントを実装することができる。

8.5　まとめ

SNMP による管理は、広域でのネットワーク管理にはそのまま利用できない。そこで、代理エージェントの考え方を用いて階層的な情報の管理を行うと、広域ネットワークの管理を効率的に行えると考えられる。デバイスについての代理エージェントを実現する場合の、アドレッシングの方法は既に提案されている。

第 9 章

WIDE インターネット ネットワークトラフィック解析

9.1 はじめに

本章では、WIDE NetStat WG において本年度おこなった WIDE Internet のトラフィック解析について、その方法および結果について報告する。

現在の WIDE Internet の構成を図 9.1に示す。

図 9.1にも示したように WIDE Internet にはすでに約 40 組織が参加するとともに、TISN や JAIN などの国内の IP Network とも相互接続されている。そのため WIDE Internet のバックボーンを流れるトラフィックは日々増加の傾向を示している。

今後の WIDE Internet の回線増強やバックボーンのトポロジー等の再構成を行うためには、現状のトラフィックの正確な把握と理解は不可欠である。また、WIDE Internet を利用した広域分散環境でのサービスを考える場合、ネットワーク上のトラフィックの傾向などを把握する必要がある。

ネットワークのトラフィック調査方法には、いろいろな方法が考えられる。

1. 既存のネットワーク管理技術を利用する。
   (a) SNMP
   (b) nnstat
   (c) etherfind
   (d) tcpdump
2. 新たな技術を開発する。

今回のトラフィック調査および解析の手段として、我々は、既存のネットワーク管理技術の評価として SNMP を用いたデータ収集を行なうとともに、新たな技術の開発に対する調査として独自のデータ収集をおこなった。また、これらの方法によって収集したデータの解析方法の手段として、perl によるデータ加工、および Language S による解析を行なった。図 9.2 第２章では、SNMP を用いたトラフィック量の解析について報告し、第３章ではパケットモニタによるトラフィック解析について報告する。

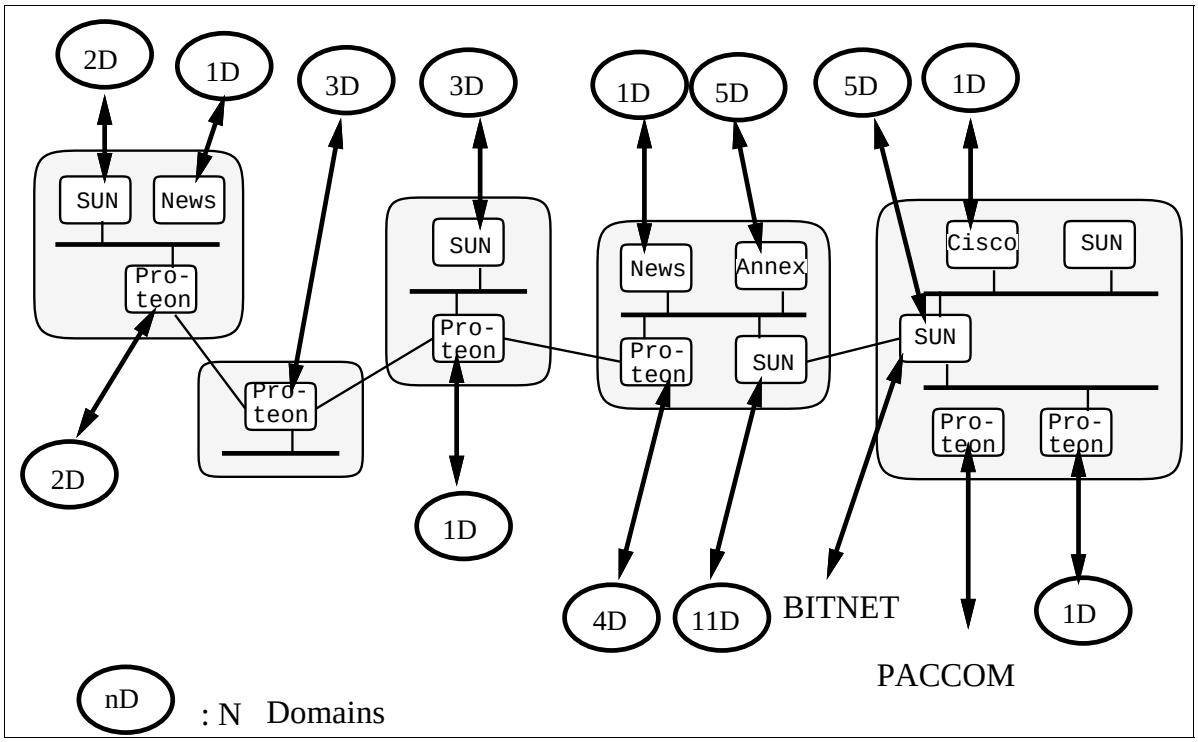


図 9.1: WIDE Internet 接続図

9.1.1 SNMP によるデータ収集

WIDE Internet のバックボーンのトラフィックに関するデータ収集の方法として、近年多くのネットワーク機器がサポートをはじめた SNMP を利用した。以下の節では、SNMP を用いた WIDE Internet のトラフィック量に関する情報収集について述べる。

9.1.2 NetStat WG での SNMP による情報収集

SNMP は、UDP 上のアプリケーションプロトコルとして実現され、1 つの情報を得るために、ASN.1 フォーマットにエンコードされた約 100 バイトのリクエストパケットと約 100 バイトのリプライパケットの交換によって情報の収集をおこなう。

SNMP による各ネットワーク機器からの情報収集は、一般的には、どのホストからも可能であるが、WIDE Internet では不必要な SNMP のトラフィックを軽減するためにバックボーンを構成しているゲートウェイからの SNMP による情報収集は、ある決められたホストからのみを許し、他のホストからの SNMP リクエストを受け取らないようにしている。これは、SNMP の通信が、WIDE Internet のバックボーンのような広いバンド幅をもった通信回線ではそれほど影響を及ぼさないが、3.4KHz 帯域を利用した SLIP などを使用している回線では、かなり大きなトラフィックになるからである。

今回の SNMP を利用した情報収集システムは、CMU によって開発された SNMP のパッケージの中の SNMP のライブラリを利用した。

CMU で開発された SNMP のライブラリを使って SNMP エージェントとの通信を行な

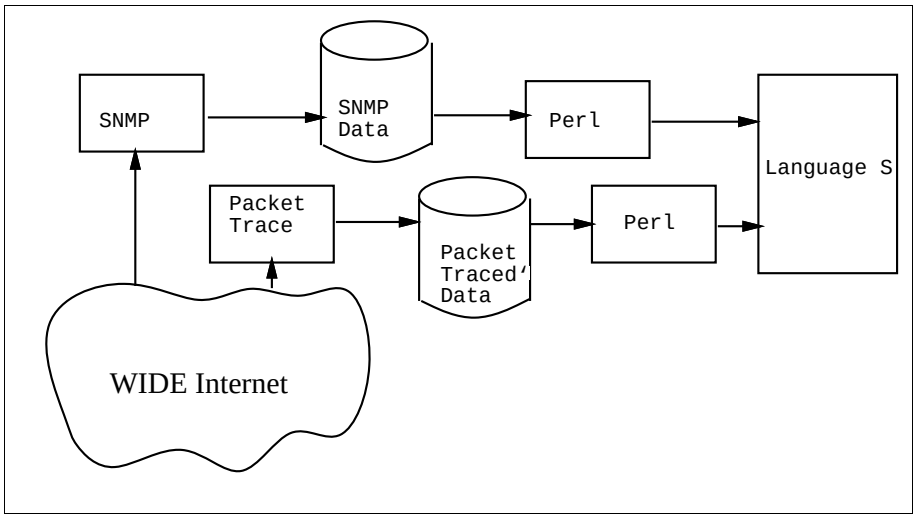


図 9.2: Processing Flow

Src to Dest	Capacity (Kbps)	Mean (Kbps)	-
PACCOM to SFC	64	37.72	58.94 %
SFC to PACCOM	64	42.24	66.00 %
SFC to TYO	192	45.15	23.52 %
TYO to SFC	192	48.15	25.08 %
TYO to KYO	64	5.02	7.84 %
KYO to TYO	64	4.48	7.00 %

表 9.1: WIDE Internet Backbone 利用率

う *colsnmp* プログラムを新たに作成した。このプログラムは引数によって指定されたディレクトリを検索し、情報を取得するゲートウェイのホスト名を理解し、各ゲートウェイごとに *snmp.config* ファイルに指定された MIB オブジェクトに関する情報を収集する。

```
#
# Configuration file for "colsnmp(collecting snmp status" of
# wnoc-tokyo-proteon.wide.ad.jp
# By Osamu N.
conf-id := 1.0
community := public
gateway := jp-gate.wide.ad.jp
# if 19: ie0: local ethernet
# if 20: ie1: SFC
# object-identifier : options
object-id := interfaces.ifTable.ifEntry.ifInOctets.19:inc
object-id := interfaces.ifTable.ifEntry.ifInOctets.20:inc
```

```
object-id := interfaces.ifTable.ifEntry.ifOutUcastPkts.19:inc
object-id := interfaces.ifTable.ifEntry.ifOutUcastPkts.20:inc
```

Figure 3: snmp.config の例

この *colsnmp* プログラムを *cron* によって 10 分間毎に呼びだし、結果をファイルに書きだす。

結果のファイルには、実際に colsnmp が実行された時間と、各オブジェクトに対応した値が 1 行として書き出される。

今回の実験では、1990 年 10 月 3 日から WIDE Internet のバックボーンホストですべての回線に対して *ifInOctets*、*ifOutOctets*、*ifInUcastPkts*、*ifOutUcastPkts* の 4 つのオブジェクトの値を収集した。また、このデータ収集は、現在も継続的に行なっている。

9.1.3　S 言語によるデータ解析

本章では、前章で説明した方法によって収集したデータの解析方法について説明し、また、解析結果についても述べる。

S 言語は、ベル研究所のベッカーとチェンバースによって開発された UNIX 上で動き、グラフィックインターフェイスが充実した統計処理用言語である。S 言語の詳細な特徴は [35][36] などを参照していただくが、特に UNIX との親和性が高く、また、グラフィック機能や時系列処理が豊富なため、データ解析はもちろん、種々のシステムの構築を行なう場合の道具として利用価値が高い。

SNMP で得られる各オブジェクトの値は、各ゲートウェイがリセットされてからの累計となる。また、データを収集しているホストが何らかの理由（システムの再起動やネットワークパーティション）で、ある時刻におけるデータの収集ができない場合もある。そこで、S 言語での処理の前処理として、*colsnmp* の出力結果を、実際のデータ収集の間隔で割って、1 分間あたりの平均トラフィック量に変換した。また、ゲートウェイが途中でリセットされるなどして、値が前の値より小さい場合には、その時間でのデータ採取ができなかったものとした。

すなわち、採取したデータとしては、

T_n:　SNMP を用いてデータを採取した時間
D_n:　採取したデータ

となり、解析を行なう時には、

$$\frac{D_n - D_{n-1}}{T_n - T_{n-1}}$$

として平均をとった解析データとした。

実際には、*colsnmp* によって収集されたデータを上記のような平均化のためのプログラムを S 言語から起動し、その処理結果のデータを S 言語を用いて種々の処理をした。

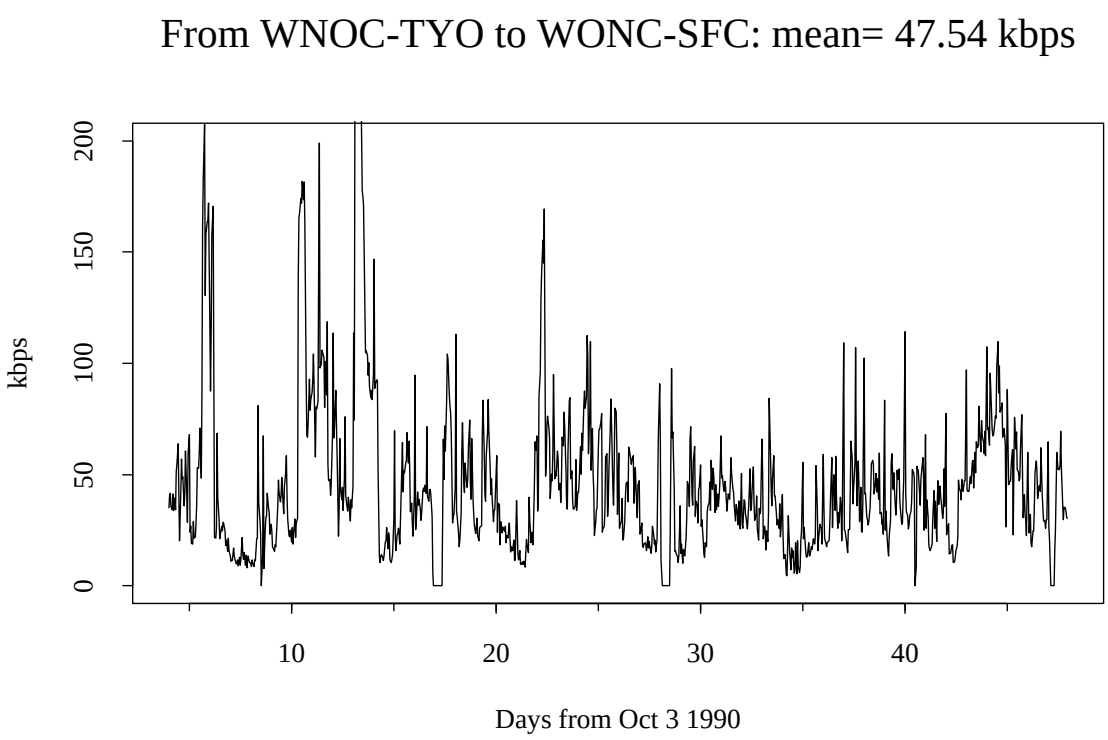


図 9.3: Traffic Data from TYO to SFC

9.1.4 回線使用率

各バックボーン間の平均回線使用率は、表 9.1の様になる。
また、実際のトラフィックをグラフィカルに表現すると図 9.3 のようになる。

9.1.5 時系列処理

S 言語が持っている時系列処理のパッケージで SABL(Seasonal Adjustment program of Bell Laboratories)[37] を用いて、データを解析すると図 9.4のようになる。
また、周期を１日とし、各時間ごとのデータを解析してみると図 9.5および図 9.6のようになる。
また、同様に周期を１週間として、各曜日についてのデータを解析してみると図 9.7、図 9.8のようになる。

9.2 トラフィック解析

SunOS の NIT を用いてネットワーク上を流れるすべてのパケットのヘッダーを読み込み、このデータを解析することによりトラフィックデータの解析をおこなった。主な解析項目を以下に示す。

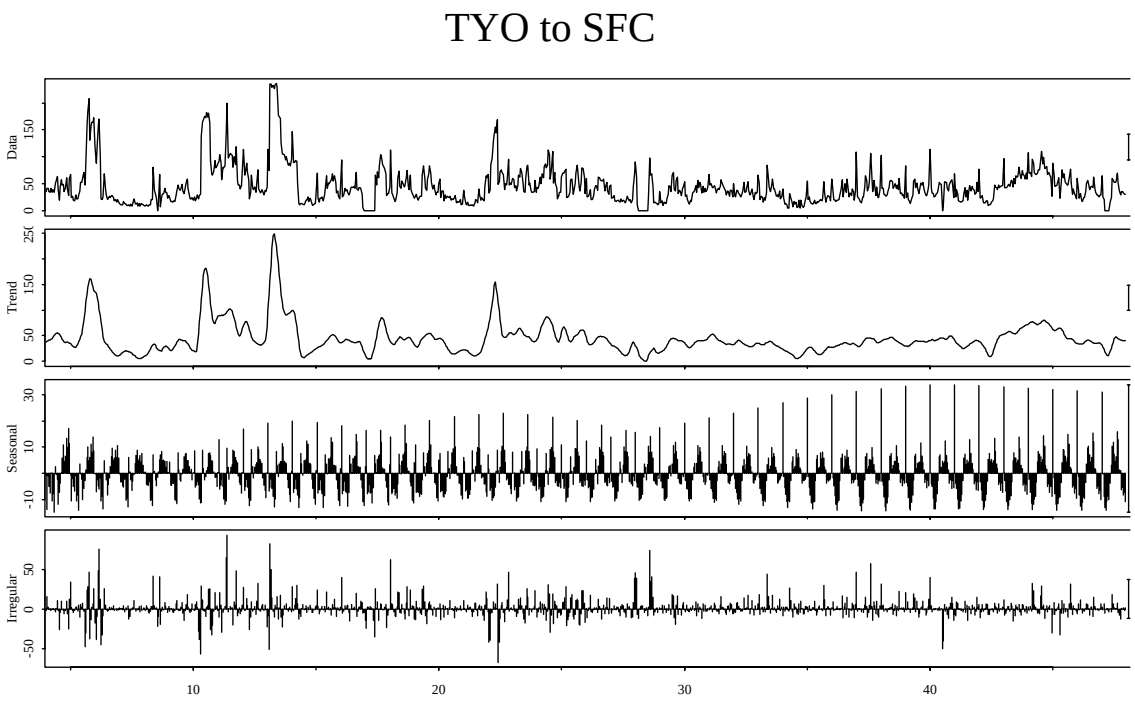


図 9.4: SABL Traffic Analize (From TYO to SFC)

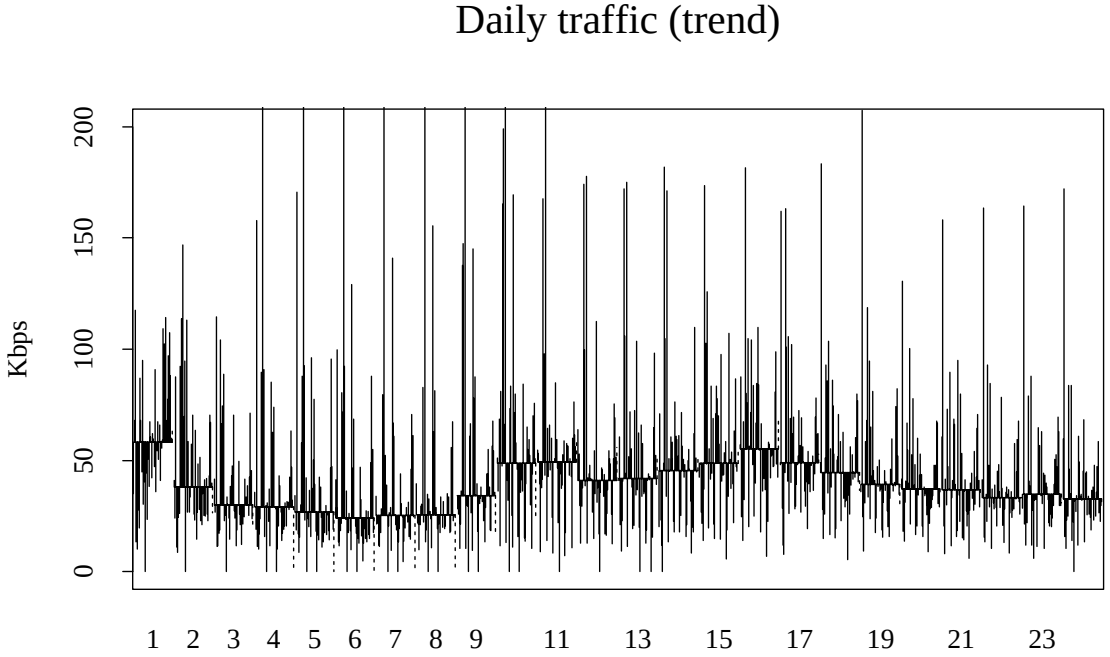


図 9.5: Daily Traffic Analize(trend)

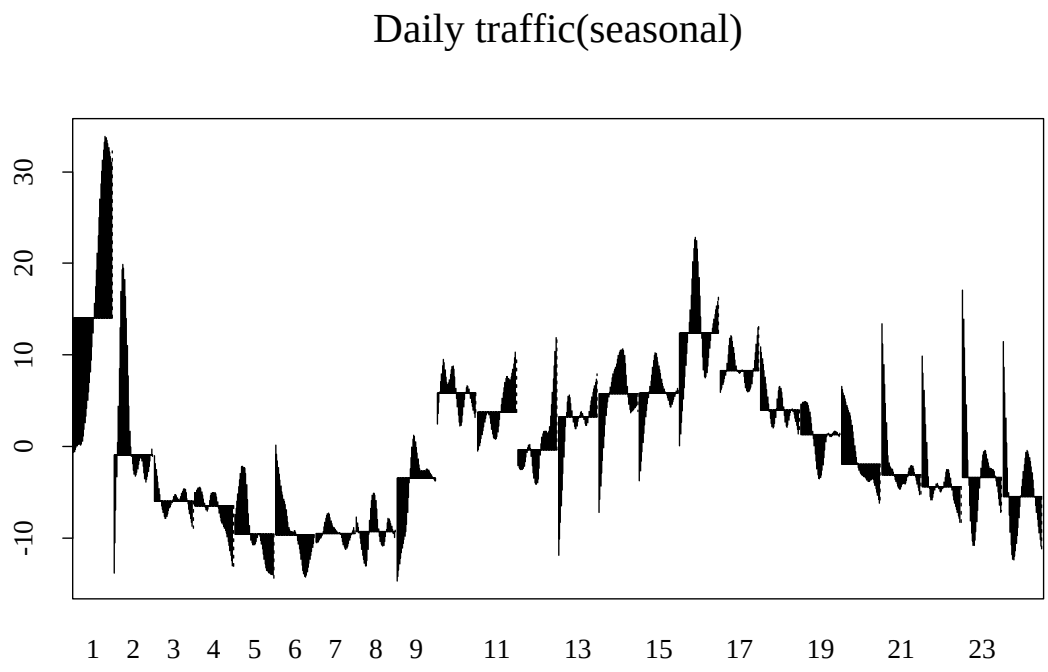


図 9.6: Daily Traffic Analize(seasonal)

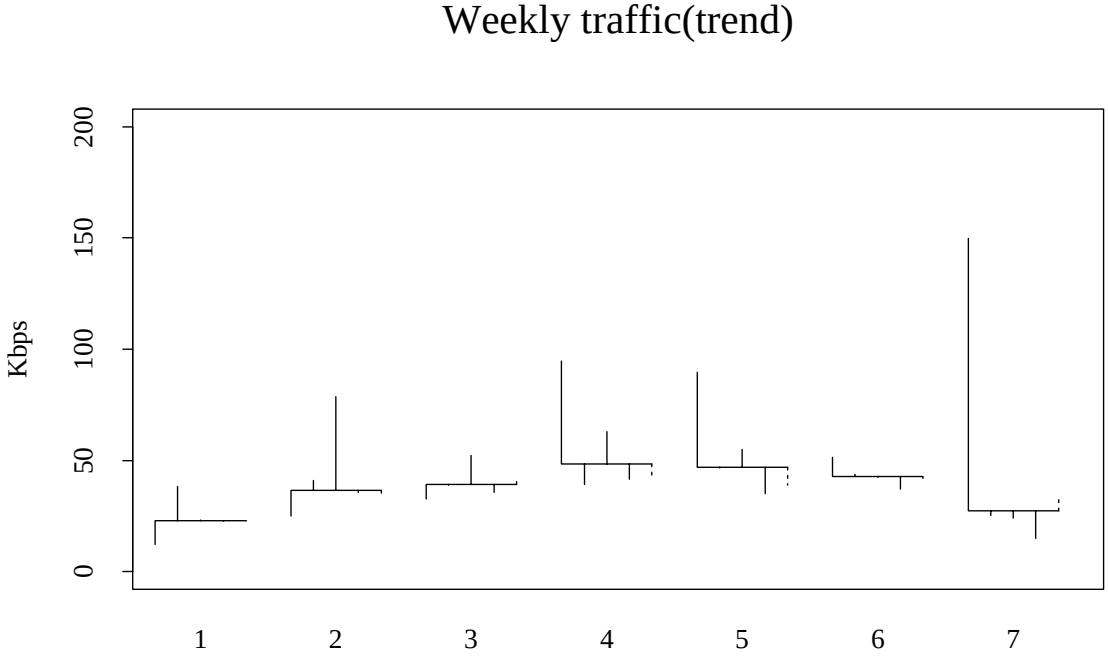


図 9.7: Weekly Traffic Analize(trend)

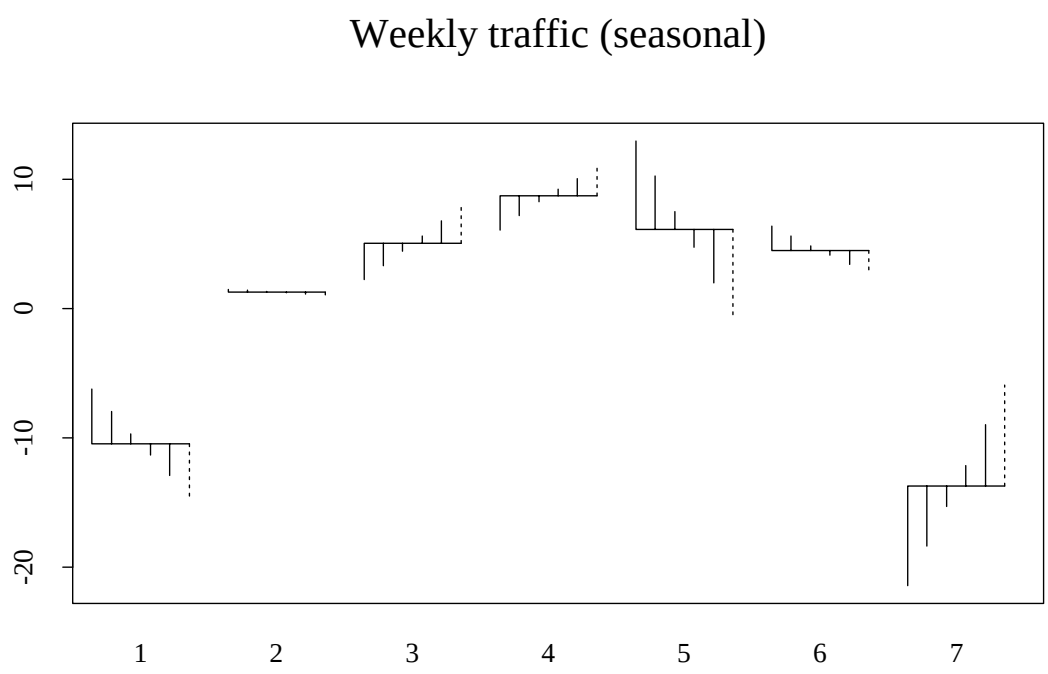


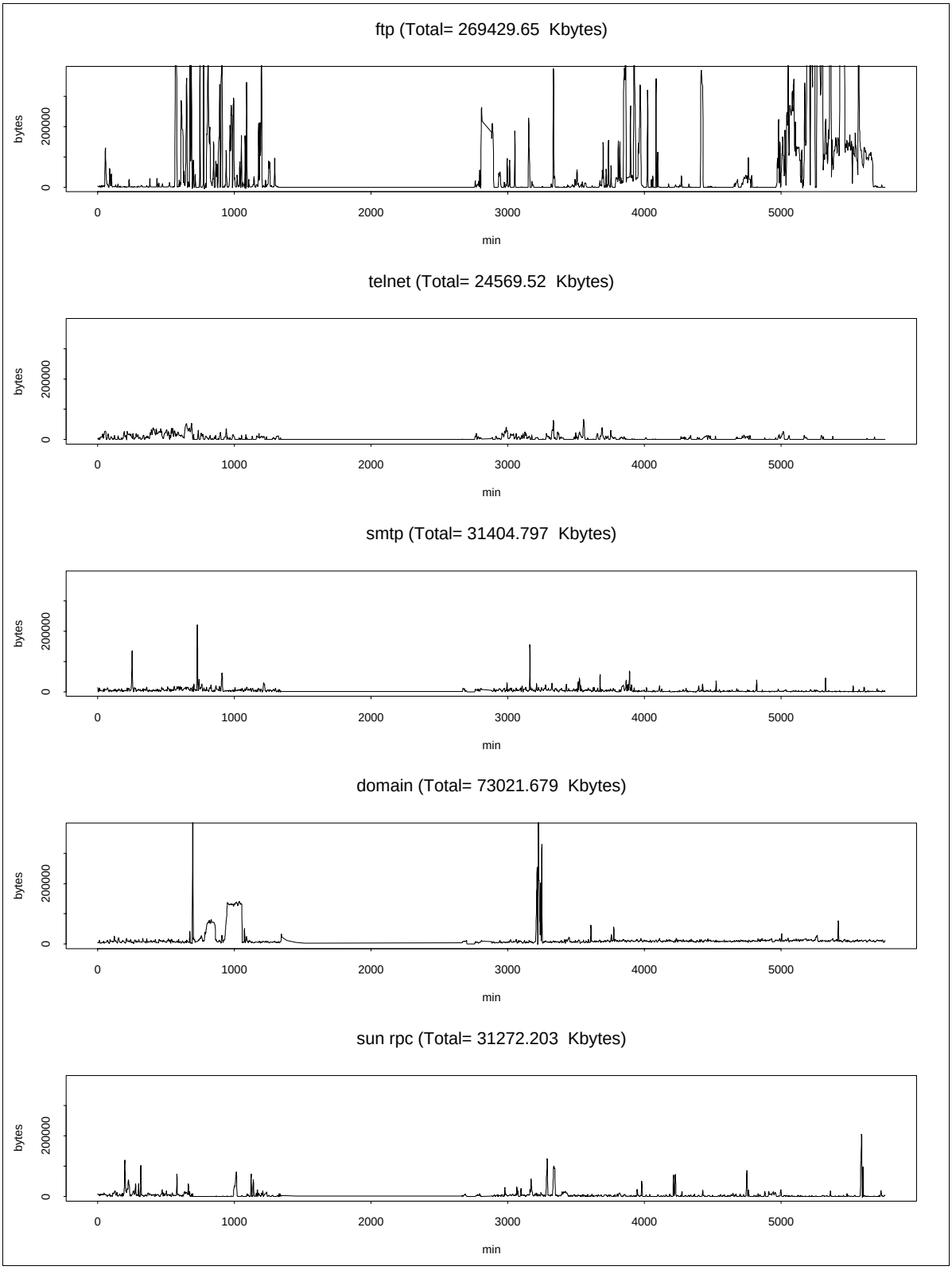
図 9.8: Weekly Traffic Analize(seasonal)

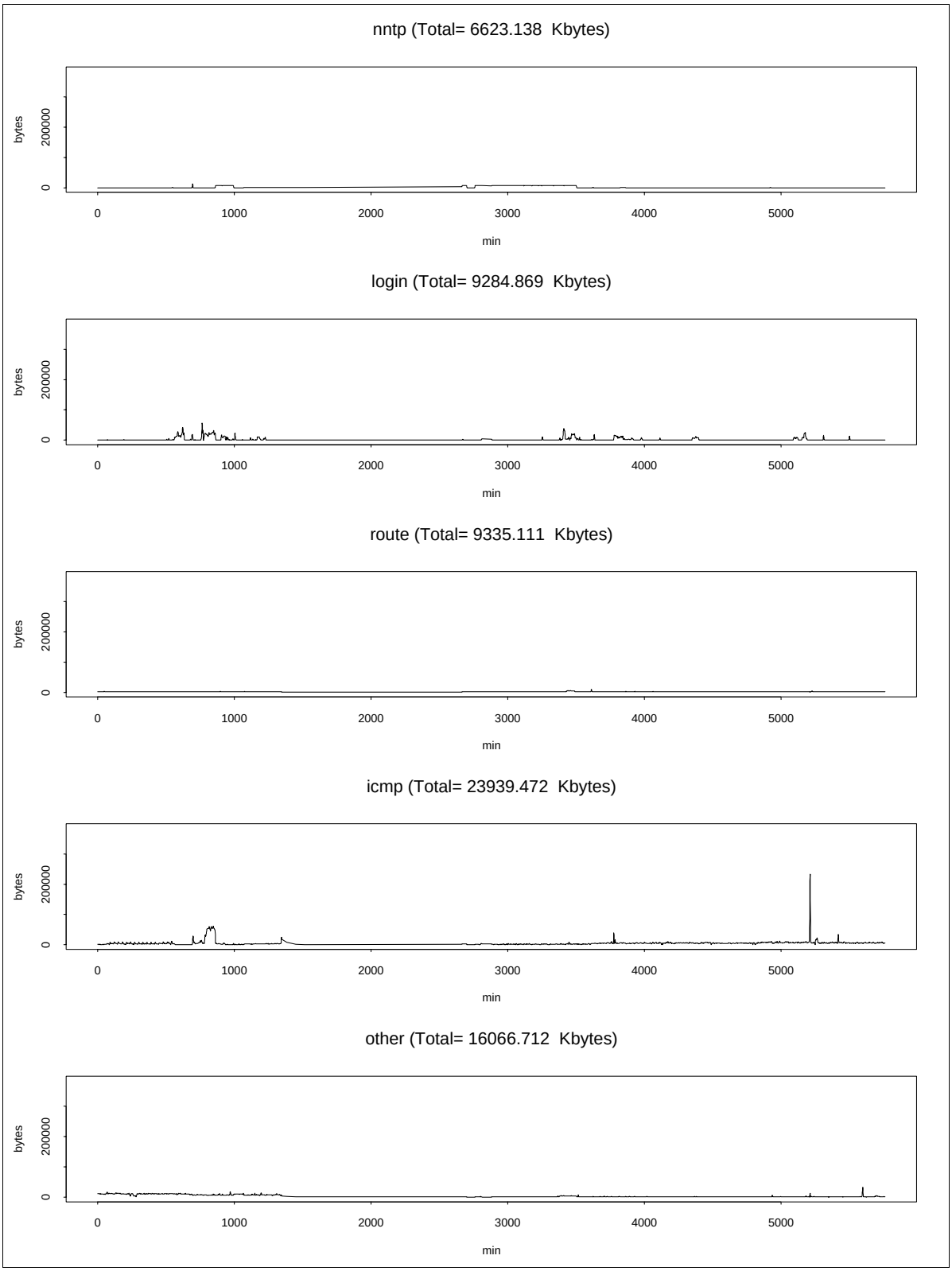
1. プロトコル（サービス）別トラフィック量
2. 関連
   (a) コネクション時間の分布
   (b) RTT の変化量

今回の解析を行なったのは、海外と国内のトラフィックについてだけである。

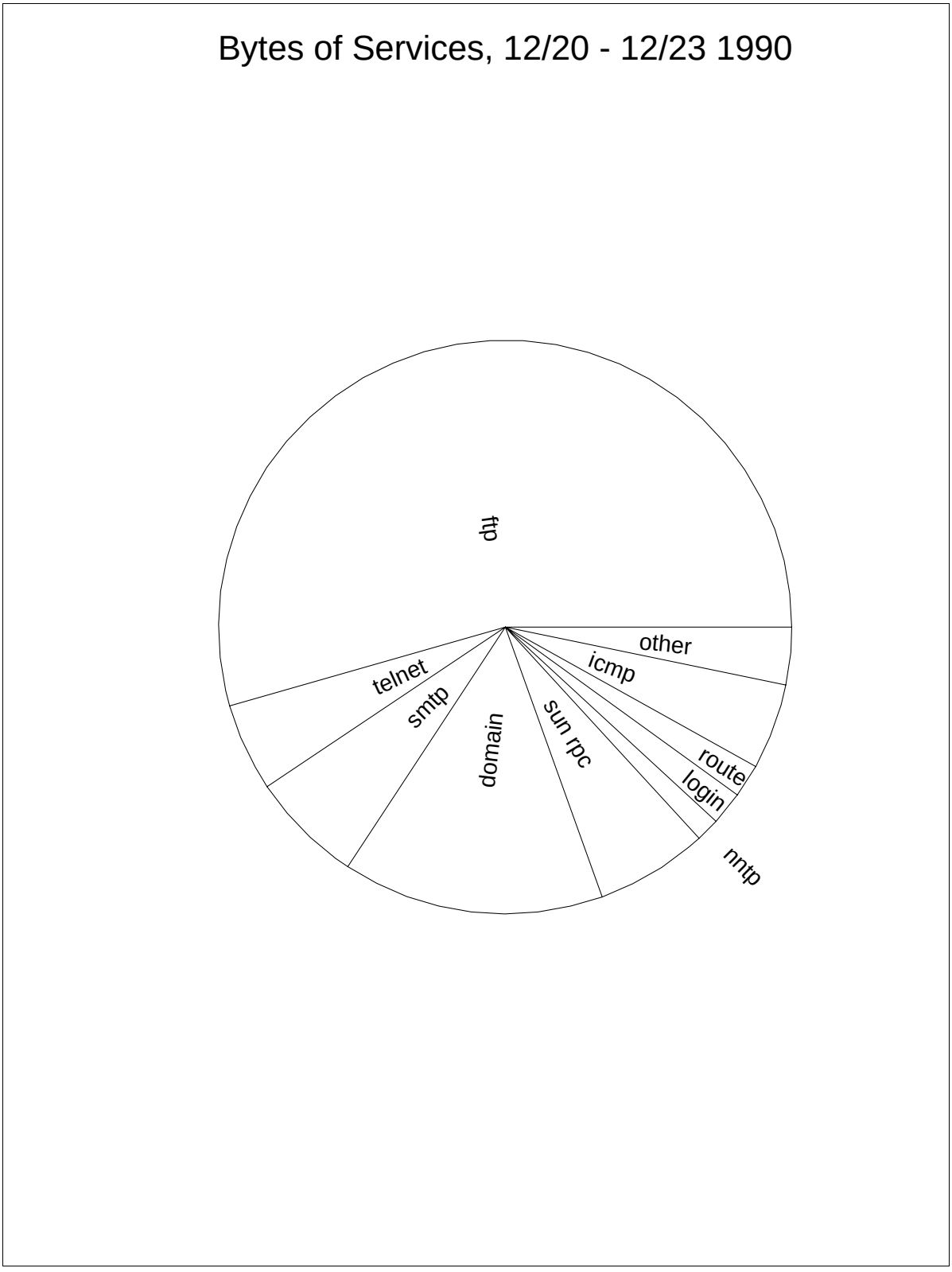
9.2.1 プロトコル別トラフィック

ftp, telnet, smtp, domain, sun rpc, nntp, login, route, icmp などそれぞれのプロトコル別トラフィックの変化を以下に示す。

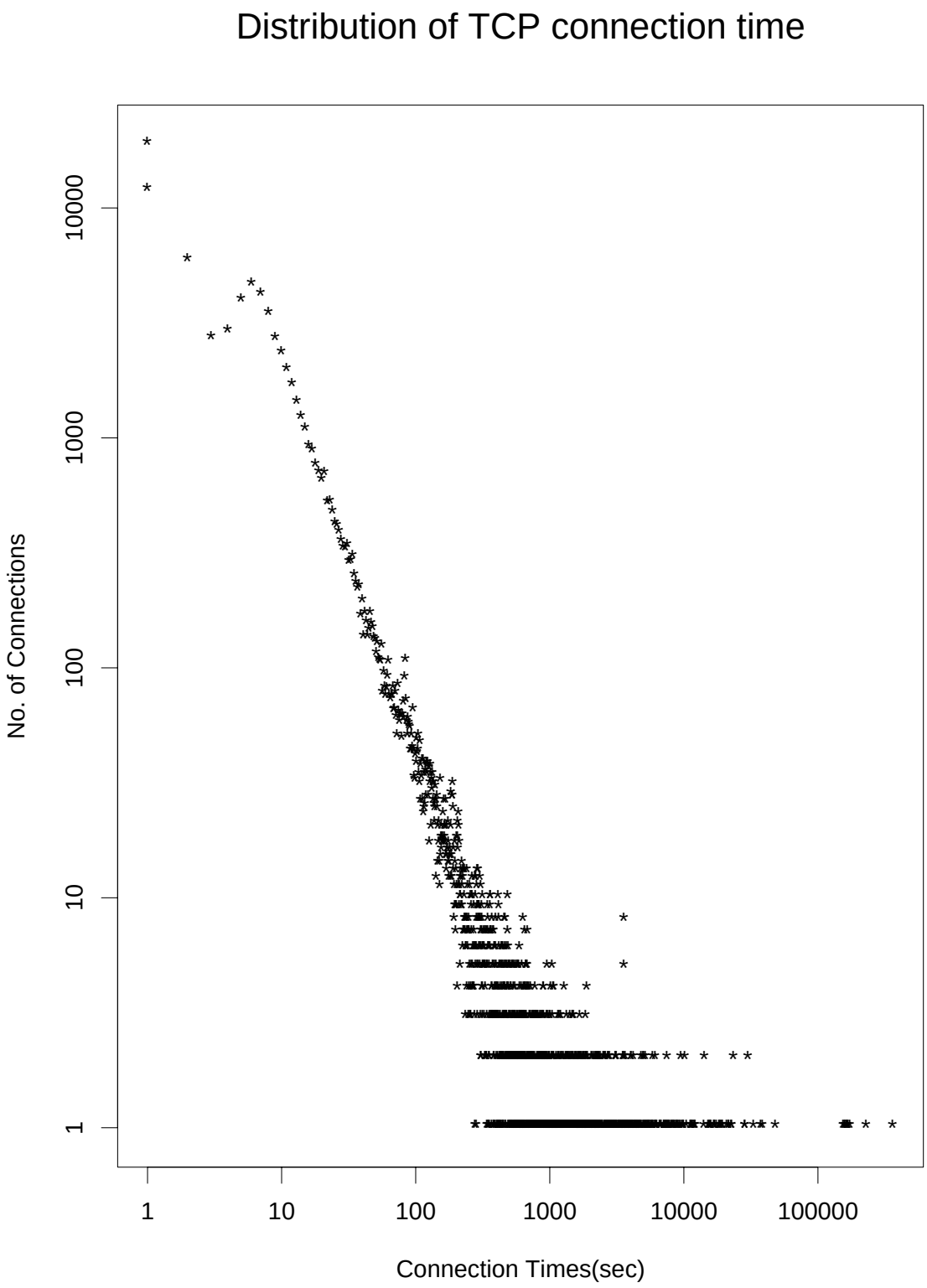



nntp (Total= 6623.138 Kbytes)
bytes
200000
0
0
1000
2000
3000
4000
5000
min
login (Total= 9284.869 Kbytes)
bytes
200000
0
0
1000
2000
3000
4000
5000
min
route (Total= 9335.111 Kbytes)
bytes
200000
0
0
1000
2000
3000
4000
5000
min
icmp (Total= 23939.472 Kbytes)
bytes
200000
0
0
1000
2000
3000
4000
5000
min
other (Total= 16066.712 Kbytes)
bytes
200000
0
0
1000
2000
3000
4000
5000
min

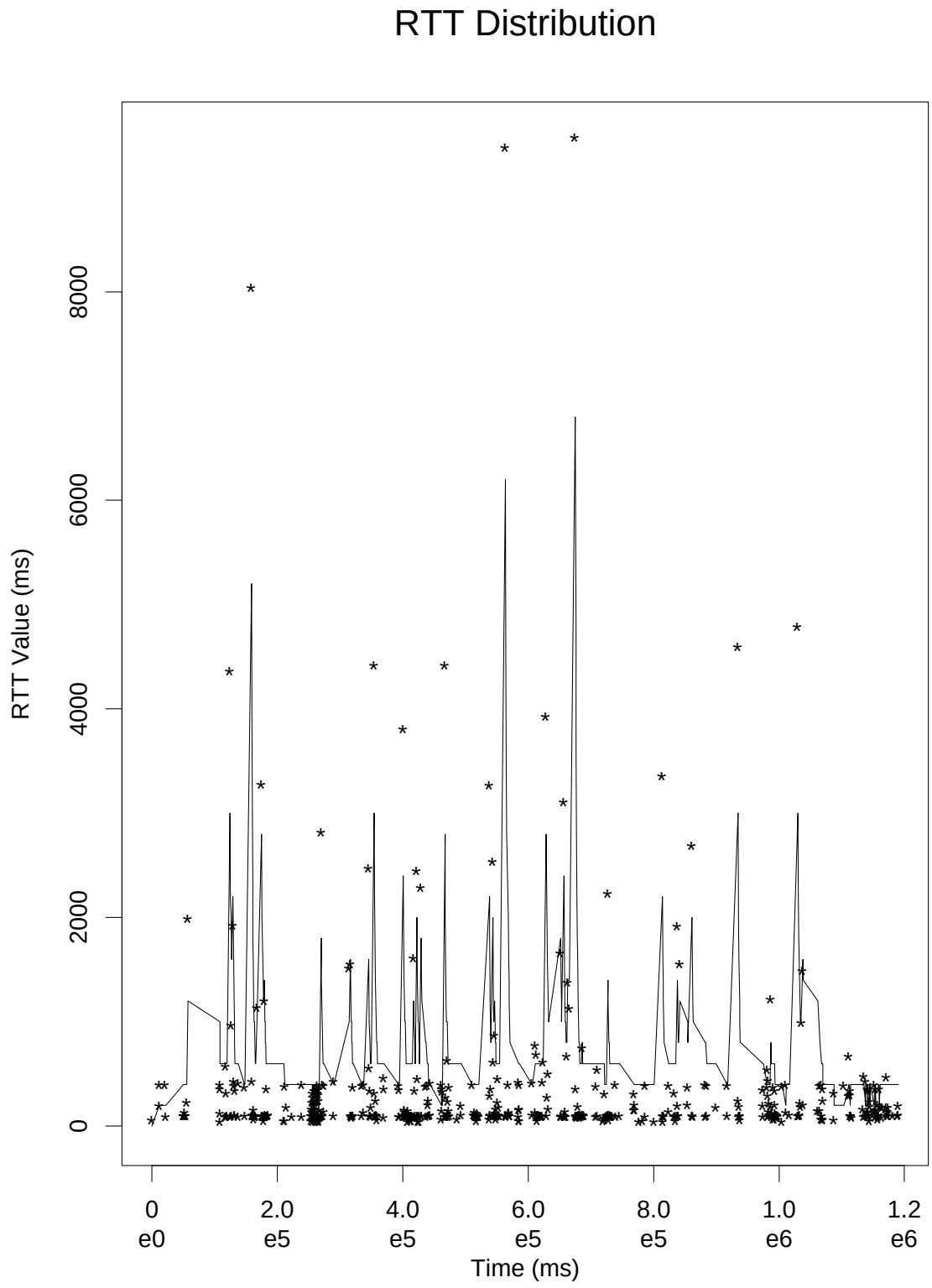
また、これらのデータ量の割合を以下に示す。



9.2.2　コネクション時間の分布



9.2.3 RTT の変化



折れ線は、TCP の retransmition のための Timeout Interval の変化である。

9.3　まとめ

本報告書では、本年度に行なったデータ収集および解析の一例を示した。今回のデータ収集及び解析は、その手法に関する調査が主な項目であったが、今後は、これらのデータを種々の目的にあった形で解析を行ない、目的に従った解析が必要となる。また、WIDE Internet の定常的なトラフィックの解析なども必要となる。NetStat WG での今後他の Working Group と協調しながら、広域分散環境におけるトラフィック解析の手法やその応用に関する研究を行なうとともに、広域分散環境におけるネットワーク管理に関する研究も行なう。

第 10 章

付録：ASN.1 概要

ASN.1(Abstract Syntax Notation 1) は、データ型を定義するために用いる記述用言語である。ASN.1 にはいくつかのプリミティブなデータ型と、構造を持った新しいデータ型を定義するための機能が備わっている。また ASN.1 で記述されたデータを OSI ネットワーク上で転送するためのエンコーディングルールも定められている。

本節では ASN.1 の syntax と BER(Basic Encoding Rule) を簡単に説明する。

10.1 ASN.1 の syntax

10.1.1 文法上の基本事項

ASN.1 では、構文上次の 4 種類の要素を使用することが出来る。

- **単語**
  upper/lowercase のアルファベット、数字、ハイフンのいずれかの文字からなる一続きの文字列。最初の文字はアルファベットでなければならない。
- **数値**
  '0'〜'9' の数字を並べた文字列。
- **文字列**
  文字列には次の三通りの記法がある。

"character string"	文字列
'0123456789ABCDEF'H	16 進数値
'01'B	2 進数値

- **区切り子**
  その他の記号。

単語間の空白文字 (スペース、タブ、行末) は無視される。コメントは空白文字が記述できるところ (任意の単語間) ならどこにでも挿入することができ、"--" で始まって同じ行の "--" もしくは行末で終る。

予約済みのキーワードは、uppercase で記述される。

オブジェクト	構文上の記法	例
型	uppercase のアルファベットで始まる単語	**Name**
値	lowercase のアルファベットで始まる単語	**id-pt-delivery**
マクロ	uppercase のアルファベットからなる単語	**SIGNED**

表 10.1: ASN.1 のオブジェクトと記法

10.1.2　モジュール

モジュールは次の形式で記述される。

```
<<module>> DEFINITIONS ::=
BEGIN

<<linkage>>

<<declarations>>
END
```

<<module>>はモジュールの名前を表す。

<<linkage>>はこのモジュールと他のモジュールとの関連を表す。このモジュールで定義した名前は “**EXPORTS**” で外部モジュールに公開し、外部モジュールで定義されている名前は “**IMPORTS**” でこのモジュールに取り込むことができる。[1]

<<declarations>>はモジュールの定義を表す。ASN.1 では次の 3 種類のオブジェクトを定義することができる。各オブジェクトはそれぞれ表 10.1の記法にしたがい区別される。

10.1.3　型

型の定義は次のように行なう。(例:型名=“**NameOfType**”、型の定義=TYPE)

```
NameOfType ::=
        TYPE
```

値の定義は次のように行なう。(例:値名=“**nameOfValue**”、型=“**NameOfType**”、値の定義=VALUE)

```
nameOfValue NameOfType ::=
        VALUE
```

[1] <<linkage>>の機能は Extensions Addendum で定められたものであり、オリジナルの ASN.1 にはこの機能はない。

ASN.1 で使用することができる型には次のようなものがある。

- SIMPLE TYPES
  - **BOOLEAN**
    “**TRUE**” か “**FALSE**” のいずれかの値を取る。
  - **INTEGER**
    通常の整数値を取る。ASN.1 としては最大値などは定めない。ラベルつきの整数値を取ることもできる。
  - **ENUMERATED**
    ラベルつきの整数値を取る。数値としての比較ができない点が INTEGER 型と異なる。
  - **REAL**
    実数値を取る。値は “mantissa”、“base”、“exponent” の三つの整数値の組で表され、式 “$mantissa \times base^{exponent}$” により実際の値が計算される。
  - **BIT STRING**
    0 ビット以上の任意長のビット列データを取る。オクテット境界 (8 ビット境界) ではないデータを表すために用いる。
  - **OCTET STRING**
    0 オクテット以上の任意長のオクテット列データを取る。各オクテットは 0〜255 の値を取る。
  - **NULL**
    NULL 値を示すデータ型である。
- OBJECT TYPES
  - **OBJECT IDENTIFIER**
    名前が割り当てられているオブジェクトを識別するためのデータ型である。非負の整数値を ‘.’ で区切った値を取り、木構造のオブジェクト空間をルートノードからたどっていずれかのノードを特定する。
  - **OBJECT DESCRIPTOR**
    オブジェクトの内容を記述するためのデータ型である。単純な文字列を値として取る。OBJECT IDENTIFIER と異なり、オブジェクトを示す一意性は保証していない。
- CONSTRUCTOR TYPES
  - **SEQUENCE**
    指定された 0 個以上の要素からなるリストを表す。要素の順序は保証される。要素には次の指定をつけることができる。

  - **OPTIONAL**
    リスト中にその要素があってもなくても良いことを表す。
  - **DEFAULT**
    値が指定されない場合の default 値を指定する。
  - **COMPONENTS OF**
    リスト中に別のサブリストを含む場合、サブリストの各要素を並べたものをリストの要素として扱うことを示す。
- **SEQUENCE OF**
  指定した要素を 0 個以上並べたリストを表す。
- **SET**
  指定した要素中の任意個の要素からなる集合を表す。要素の順序は保証されない。要素に “**COMPONENTS OF**” 指定をつけることができる。
- **SET OF**
  指定した要素を 0 個以上並べた集合を表す。

- TAGGED TYPES
  ネットワーク上で通信する際に生じ得るエンコーディング上の曖昧さをなくすために、データ型の定義にタグをつけることができる。タグにはその一意性の範囲に応じて 4 種類のクラスが用意されている。
  - **universal クラス**
    ASN.1 全体で一意性を保証する必要があるクラス。このタグは ASN.1 のドキュメントと Addenda 中でのみ定義される。表 10.2参照。クラス名は “**UNIVERSAL**”。
  - **application-wide クラス**
    あるモジュール中で一意性を保証する必要があるクラス。クラス名は “**APPLICATION**”。
  - **context-specific クラス**
    SEQUENCE などのような constructor type の定義中で、その位置で一意性を保証する必要があるクラス。
  - **private-use クラス**
    ある組織中で一意性を保証する必要があるクラス。クラス名は “**PRIVATE**”。

  context-specific 以外のクラスのタグに関しては、定義中にクラス名と非負の整数値の組で次のように表す。

```
型 ::=
    [クラス名 値]
        定義
```

context-specific クラスは定義毎に定まるクラスではないため、例えば次のように表す。

```
NotSoFuzzyThinking ::=
    SEQUENCE {
        alpha[0]        -- タグ=0
            INTEGER
            OPTIONAL,

        beta[1]         -- タグ=1
            INTEGER
            OPTIONAL
    }
```

通常タグをつけることによってネットワーク上を流れるデータの量は増えるが、これを抑えるために元々のデータにつくタグを省略する指定ができる。

- **EXPLICIT**
  元々のデータのタグを省略しない。指定しない場合は EXPLICIT が指定された場合と同じ。
- **IMPLICIT**
  元々のデータのタグを省略する。

例えば次のように定義すればネットワーク上を流れるデータには “UNIVERSAL 22” のタグと OCTET STRING(=“UNIVERSAL 4”) のタグが両方つくが、

```
IA5String ::=
    [UNIVERSAL 22]
        OCTET STRING
```

次のように定義すればデータには “UNIVERSAL 22” のタグしかつかない。

```
IA5String ::=
    [UNIVERSAL 22]
        IMPLICIT OCTET STRING
```

- META TYPES
  - **CHOICE**
    指定された要素中のいずれか一つだけを取る。
  - **ANY**
    ASN.1 で定義される全ての型のいずれかを取る。
  - **EXTERNAL**
    モジュール外で定義されている任意の型を取る。ANY と異なり、ASN.1 以外で定義されている型でも構わない。
- SUBTYPES
  型名の直後に “(制限子)” をつけることにより、より取り得る値の範囲を狭めた subtype を定義することができる。例えば次のようにして正の整数値を取る型を定義することができる。

```
PositiveNumber ::=
    INTEGER (0<..MAX)
```

10.1.4　マクロ

ASN.1 では次のようにマクロを定義することができる。

```
<<macro>> MACRO ::=
BEGIN

TYPE NOTATION ::= <<type syntax>>

VALUE NOTATION ::= <<value syntax>>

<<supporting syntax>>
END
```

<<macro>>はマクロの名前を表す。
<<type syntax>>はマクロ呼び出し時に変数名・マクロ名に続いて与えられる文法規則を定義する。
<<value syntax>>はマクロ自身が取る値を定義する。
最後に<<supporting syntax>>として、それら以外に必要な文法規則を定義することができる。

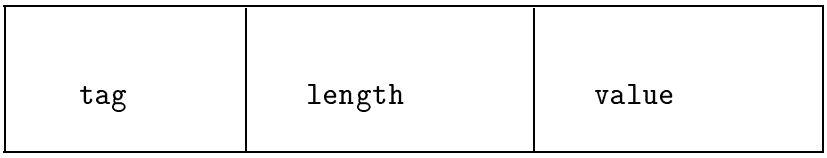


図 10.1: ASN.1 エンコーディングのフィールド

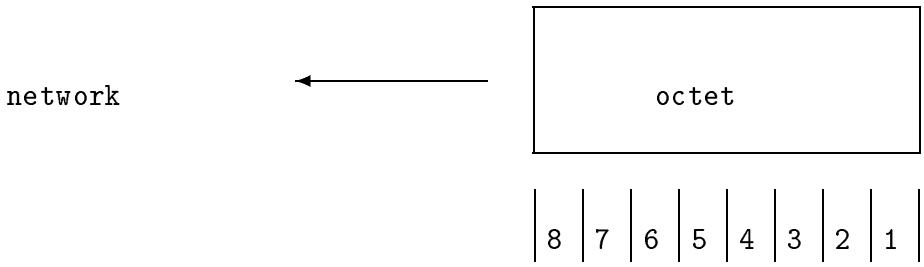


図 10.2: ASN.1 エンコーディングのビットオーダー

10.2 ASN.1 の BER

10.2.1 エンコーディングの基本事項

ASN.1 の型は、BER によって次の 3 つのフィールドに分けて扱われる。(図 10.1参照)

- **タグフィールド (tag)**
  ASN.1 の型の種類を表す。

- **長さフィールド (length)**
  エンコードされているデータ自身の長さを表す。

- **値フィールド (value)**
  値をエンコードしたデータである。

エンコードされたデータのうち、最初のオクテットのビット 8 を MSB(Most Significant Bit)、最後のオクテットのビット 1 を LSB(Least Significant Bit) と呼ぶ。データは、ネットワーク上に MSB から送られる。(図 10.2参照)

正負、および 0 を取る整数値は 2 の補数形式で表される。n ビット長のデータにおいて、MSB を $bit(n-1)$、LSB を $bit(0)$ と呼ぶことにすると、値 x は次のように求めることができる。

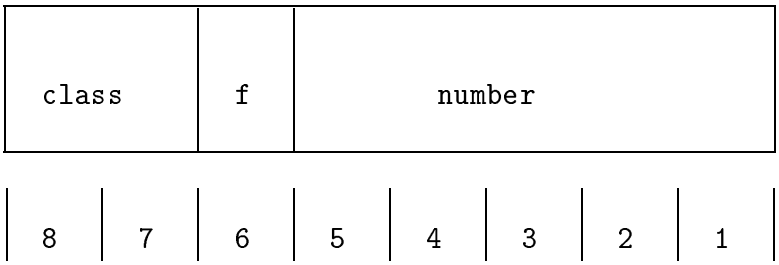


図 10.3: タグフィールドの形式

$$x = \left(\sum_{i=0}^{n-2} bit(i) \times 2^i \right) - bit(n-1) \times 2^{n-1}$$

また非負の値を取る整数値は、次のように値 x を求めることができる。

$$x = \left(\sum_{i=0}^{n-1} bit(i) \times 2^i \right)$$

10.2.2　タグフィールド

ASN.1 のタグフィールドは、1 オクテット以上のオクテット列で表される。最初のオクテットは必ず図 10.3のような形式になっている。

上位 2 ビット (ビット 8 とビット 7) は表 10.3のようにクラスを表す。

3 番目のビット (ビット 6) は primitive か constructed かを表す。

残りの 5 ビット (ビット 5〜ビット 1) は非負の値を取り、タグ番号を表す。値としては 0〜31 の 32 通りの値を取ることができるが、これだけでは足りないことが予想されるため BER ではタグ番号を次のように定義している。

- タグ番号が 31 未満の場合には、5 ビットがそのままタグ番号を表す。
- タグ番号が 31 以上の場合には 5 ビットを全部 1 にし、続くオクテットもタグ番号を表すことにする。

多くの場合にはタグ番号が 30 以下で済むためタグフィールドは 1 オクテットであるが、31 以上になる場合には最初のオクテットに続きタグ番号を表すオクテットが 1 オクテット以上続く複数オクテットのタグフィールドとなる。

後に続くオクテットのうち、最上位ビット (ビット 8) が 0 のオクテットが最後のオクテットとなる。タグ番号としては、後に続く全てのオクテットの最上位ビットを除いた残りの 7 ビットをつなげたビット列を非負の整数値として解釈した値が使用される。[2]

[2] この形式は、値フィールド中で非負の整数値を表す場合にも使用される。

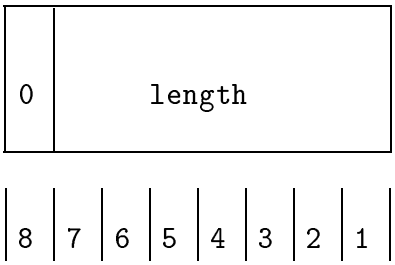


図 10.4: 長さフィールドの形式 (短い場合)

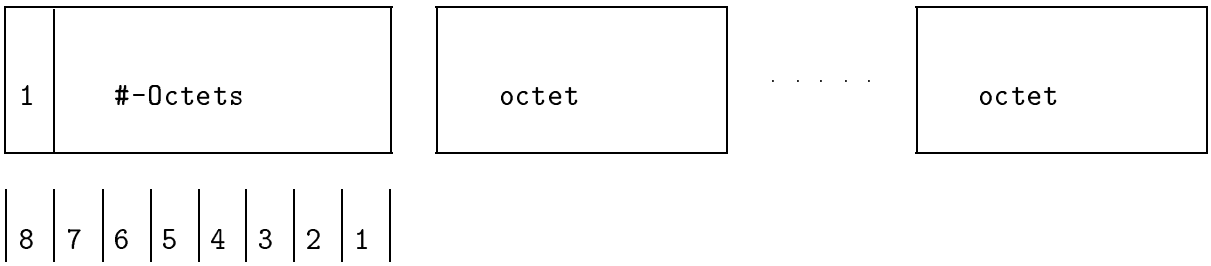


図 10.5: 長さフィールドの形式 (長い場合)

10.2.3 長さフィールド

ASN.1 の長さフィールドは 1 オクテット以上のオクテット列で表され、長さフィールドの後に続く値フィールドのオクテット数を示す。長さフィールドは図 10.4、図 10.5で表される形式になっている。

長さが 126 以下の場合には図 10.4の形式が使用され、127 以上の時には図 10.5の形式が使用される。[3]

10.2.4 値フィールド

値フィールドのエンコーディングは、値が持つ型によってそれぞれ個別に定められている。以下に、各データ型における値フィールドのエンコーディングルールを説明する。

- **BOOLEAN**
  値は 1 オクテットを占め、全ビットが 0 の時 "**FALSE**"、それ以外の場合は "**TRUE**" を表す。

[3] 7 ビットが全部 1 の場合は将来のために予約されているため、長さ 127 を表すためには使用されない。

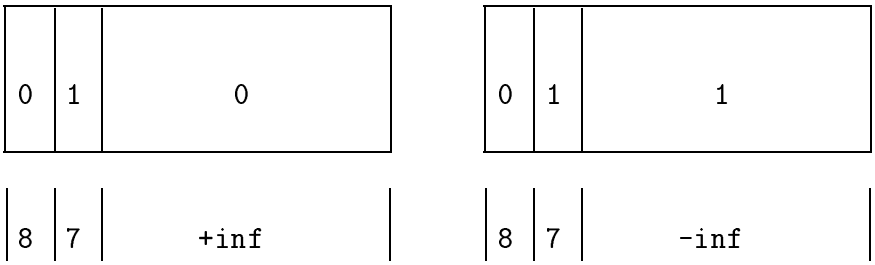


図 10.6: 定義済みの special REAL 値 ($+\infty$ と $-\infty$)

- **INTEGER**
  値は 1 オクテット以上で、全ビットを合わせて 2 の補数形式で表す。

- **REAL**
  最初のオクテットの上位 2 ビットにより、表 10.4のような意味を持つ。

  現在 special の値としては、図 10.6に示す 2 つが定義されている。

  decimal は International Standard 6093 にしたがった表記法である。最初のオクテットの下位 6 ビットで、それ以降のオクテットのフォーマットを定義する。

  binary は図 10.7のようにエンコーディングされ、$S \times N \times 2^F \times B^E$の形式で実数値を表す。

  - Sは符号を表す。
  - Fは小数点をオクテット境界に置くための補正値を表す。
  - Nは数値を表す。
  - Bは基数を表す。基数は表 10.5にしたがい 2、8、16 のいずれかとなる。
  - Eは指数を表す。指数の長さは表 10.6にしたがう。

- **BIT STRING**
  最初のオクテットは、最後のオクテットの下位何ビットが使われていないかを表す。実際の値は 2 番目以降のオクテットに収められ、最後のオクテットの有効なビットまで連続してビット列と解釈される。

- **OCTET STRING**
  オクテット列をそのまま解釈する。

- **NULL**
  長さが 0 である。

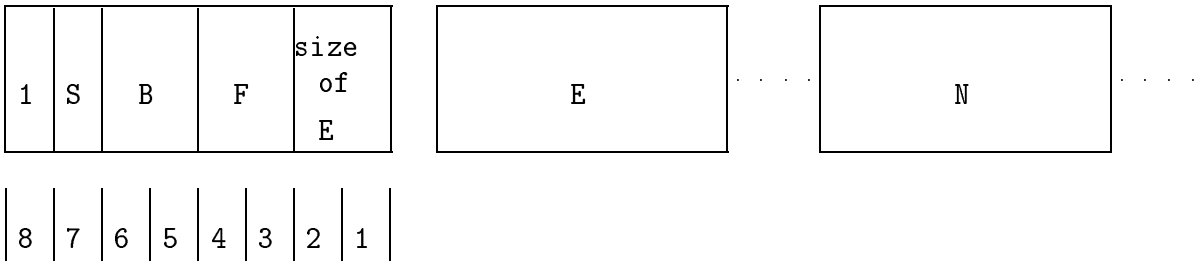


図 10.7: binary REAL 値の表現

- **OBJECT IDENTIFIER**
  最初の 2 要素を除いて、全ての要素を非負の整数値を並べて表す。最初の 2 要素だけは次の式にしたがい 1 オクテットに圧縮する。ただし Xは最初の要素、Yは 2 番目の要素を表し、“40” は十進数値である。

$$X \times 40 + Y$$

- **OBJECT DESCRIPTOR**
  次の定義にしたがって解釈される。

```
ObjectDescriptor ::=
    [UNIVERSAL 7]
        IMPLICIT GraphicString

GraphicString ::=
    [UNIVERSAL 25]
        IMPLICIT OCTET STRING
```

- **SEQUENCE**
  SEQUENCE の値フィールドは、指定された要素を順番に並べたものとなる。各要素は再帰的に図 10.3の構造を持つ。たとえば次のような SEQUENCE のエンコーディングは、図 10.8のような構造となる。

```
VarBind ::=
    SEQUENCE {
        name
            ObjectName,
```

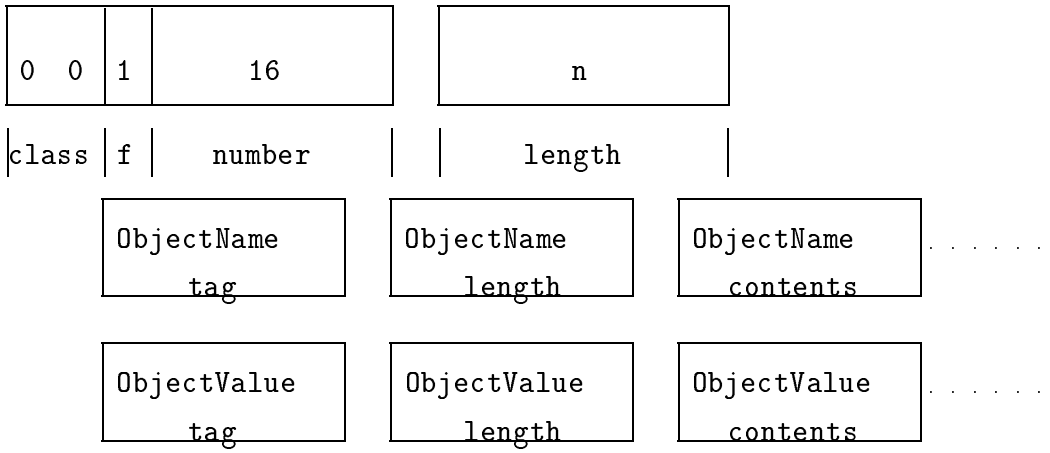


図 10.8: SEQUENCE のエンコーディング

```
        value
            ObjectValue
    }
```

ここで SEQUENCE の長さ n は、全ての要素 (この例では **ObjectName** と **ObjectValue**) のタグフィールド、長さフィールド、値フィールドの全オクテットを合計したオクテット数を表す。[4]

- **SEQUENCE OF**
  SEQUENCE OF のエンコーディングは、SEQUENCE のエンコーディングと同じである (タグ番号も同じ)。

- **SET**
  SET のエンコーディングは基本的には SEQUENCE のエンコーディングと同じであるが、タグ番号が違う点と要素の順番が意味を持たない点が異なる。

- **SET OF**
  SET OF のエンコーディングは、SET のエンコーディングと同じである (タグ番号も同じ)。

- **Tagged Types**
  次のような定義は図 10.9のようにエンコーディングされ、

[4] あらかじめ長さが分からない場合には、長さフィールドのオクテットを未定義の長さを表す'10000000'B とし、全ての要素をエンコードした後に値フィールドの終了を表す 2 オクテット'00000000'B･'00000000'B をつけてエンコードすることもできる。

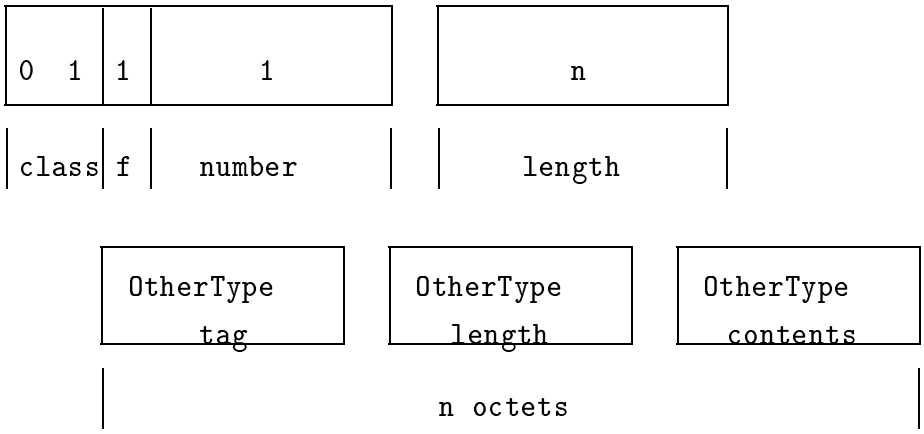


図 10.9: EXPLICIT タグのエンコーディング例

```
SomeType ::=
    [APPLICATION 1]
        OtherType
```

次のような定義は図 10.10のようにエンコーディングされる。

```
SomeType ::=
    [APPLICATION 2]
        IMPLICIT OtherType
```

- **CHOICE**
  次の例では、値 **unixEpoch** は **UTCTime** 型 (=**IMPLICIT OCTET STRING** 型) としてエンコードされ、値 **unknownTime** は **NULL** 型としてエンコードされる。

```
TimeOfDay ::=
    CHOICE {
        actual-value
            UTCTime,

        not-available
            NULL
```

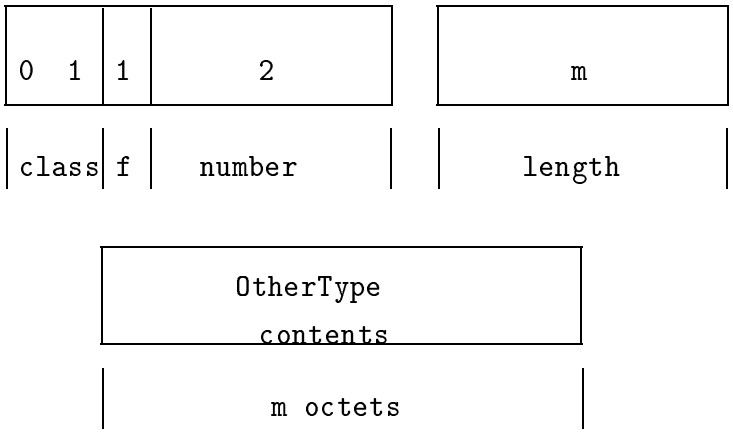


図 10.10: IMPLICIT タグのエンコーディング例

```
    }

unixEpoch TimeOfDay ::=
    { actual-value "700101000000Z" }

unknownTime TimeOfDay ::=
    { not-available NULL }
```

- **ANY**
  実際に送る値ごとに異なるが、文法的には ASN.1 にしたがっているため再帰的に ASN.1 の解釈を行なえば良い。

- **EXTERNAL**
  EXTERNAL は ASN.1 内部で次のように定義されている。

```
EXTERNAL ::=
    [UNIVERSAL 8]
        IMPLICIT SEQUENCE {
            direct-reference
                OBJECT IDENTIFIER
                OPTIONAL,

            indirect-reference
                INTEGER
```

```
        OPTIONAL,

    data-value-descriptor
        ObjectDescriptor
        OPTIONAL,

    encoding
        CHOICE {
            single-ASN1-type[0]
                ANY,

            octet-aligned[1]
                IMPLICIT OCTET STRING,

            arbitrary[2]
                IMPLICIT BIT STRING
        }
}
```

Universal Tag	ASN.1 Type
1	BOOLEAN
2	INTEGER
3	BIT STRING
4	OCTET STRING
5	NULL
6	OBJECT IDENTIFIER
7	ObjectDescriptor
8	EXTERNAL
9	REAL
10	ENUMERATED
12〜15	(予約)
16	SEQUENCE, SEQUENCE OF
17	SET, SET OF
18	NumericString
19	PrintableString
20	TeletexString
21	VideotexString
22	IA5String
23	UTCTime
24	GeneralizedTime
25	GraphicsString
26	VisibleString
27	GeneralString
28	CharacterString
29〜...	(予約)

表 10.2: ASN.1 Universal Tags

クラス	ビット 8	ビット 7
universal	0	0
application-wide	0	1
context-specific	1	0
private	1	1

表 10.3: クラスの表現

h

エンコーディング	ビット 8	ビット 7
binary	1	N/A
decimal	0	0
special	0	1

表 10.4: REAL 値の種類

基数	ビット 6	ビット 5
2	0	0
8	0	1
16	1	0

表 10.5: 基数の指定

size of E	ビット 2	ビット 1
1	0	0
2	0	1
3	1	0

表 10.6: 指数の長さ