QualNet Hacks

 # 1 CBR

About QualNet Hacks

QualNet は、ほぼ全てソースコードが公開されています。

これらのソースコードには QualNet の内部を理解するために有益な情報が沢山ちりばめられています。

しかしながら、ソースコードの量は莫大であり、その内容を簡単に理解することが難しいのも事実です。

本書は、QualNet 内部の理解をより深めて頂く事を目的として作成しました。

本書を手掛かりにして、より一層 QualNet 活用して頂ければ幸いです。

このドキュメントは QualNet5.0.2 のソースコードに準拠しています。

【ソースコードに関する注意事項】

本ドキュメントには、ソースコードの一部が複写されています。ソースコードの使用に関しては、以下の開発元の制限に則りますので、ご注意下さい。

// Copyright (c) 2001-2009, Scalable Network Technologies, Inc. All Rights Reserved.
// 6100 Center Drive, Suite 1250
// Los Angeles, CA 90045 sales@scalable-networks.com
//
// This source code is licensed, not sold, and is subject to a written license agreement.
// Among other things, no portion of this source code may be copied, transmitted, disclosed,
// displayed, distributed, translated, used as the basis for a derivative work, or used,
// in whole or in part, for any program or purpose other than its intended use in compliance
// with the license agreement as part of the QualNet software.
// This source code and certain of the algorithms contained within it are confidential trade
// secrets of Scalable Network Technologies, Inc. and may not be used as the basis
// for any other software, hardware, product or service.

1 CBR

CBR(Constant Bit Rate)アプリケーションとは、CBR クライアントから CBR サーバに向けて一定サイズのデータアイテムが一定時間間隔で送られるアプリケーションである。データアイテムは UDP 上で送られるため、アイテムの到達性は保証されない。クライアント・サーバ間のセッションは、クライアントから送られるデータアイテムが初めてサーバに届いた時に開始される。

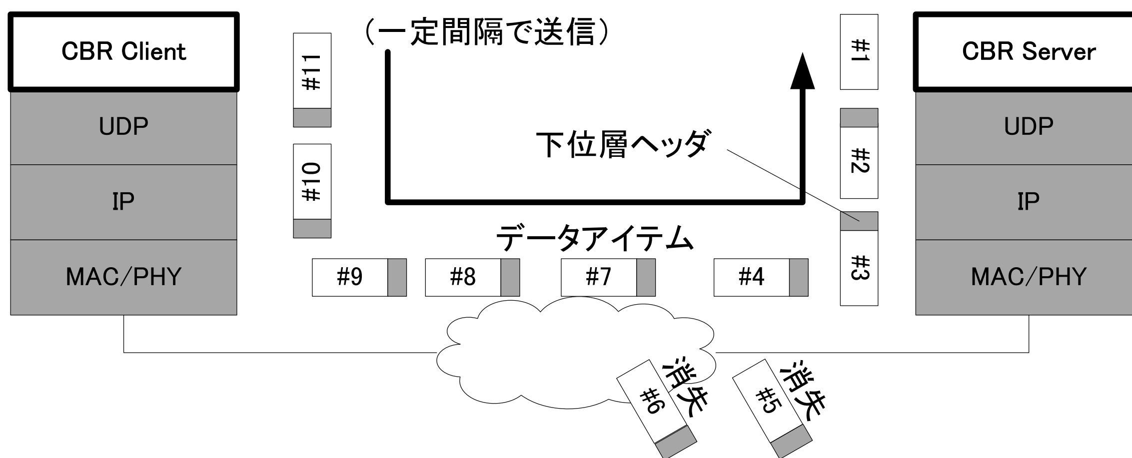

図 1 CBR アプリケーション概要

1.1 グローバル定数

CBR アプリケーションが使用するグローバル定数の一覧を表 1 に示す。

表 1 グローバル定数

定数名	値	説明(定義箇所等)
MSG_APP_TimerExpired	605	アプリケーション層の各インスタンスがタイマとして使用するイベント種別である。include/api.h に、全てのイベント種別を定義した列挙体(以下、EventType 列挙体と呼ぶ)があり、このファイルで定義している。
MSG_APP_FromTransport	610	トランスポート層からのイベント通知に使用されるイベント種別である。include/api.h の EventType 列挙体で定義している。
APP_TIMER_SEND_PKT	1	データアイテム送信を示すイベントとして使用されるサブイベント種別である。include/application.h で定義している。
APP_CBR_CLIENT	60	CBR クライアントであることを示すアプリケーション種別である。include/application.h の AppType 列挙体で定義している。
APP_CBR_SERVER	59	CBR サーバであることを示すアプリケーション種別である。include/application.h の AppType 列挙体で定義している。

定数名	値	説明(定義箇所等)
APP_LAYER	7	アプリケーション層であることを示すレイヤ種別である。include/main.h に、全てのレイヤ種別を定義した列挙体(以下、LayerType 列挙体と呼ぶ)があり、このファイルで定義している。
APP_MAX_DATA_SIZE	65023	アプリケーション層で使用できる最大データアイテムサイズ。正確には以下の値に#define されている。(IP_MAXPACKET)-(MSG_MAX_HDR_SIZE) include/application.h で定義している。
TRACE_CBR	4	CBR アプリケーションであることを示すトレース対象プロトコル種別である。include/trace.h の TraceProtocolType 列挙体で定義している。
TRACE_APPLICATION_LAYER	0	アプリケーション層であることを示すトレース対象レイヤ種別である。include/trace.h の TraceLayerType 列挙体で定義している。
CLOCKTYPE_MAX	右参照	使用できる時刻の最大値。以下の値に#define されている。TYPES_ToInt64(0x7fffffffffffffff)

1.2 データ構造

1.2.1 アプリケーションインスタンスリスト

QualNet では、1 つのノード上に複数のアプリケーションインスタンスを設定することができる。

ノード上に存在するアプリケーションインスタンスは、図 2 に示すように node->appData の appPtr から辿ることのできるリストで管理されている。このリストを以後、アプリケーションインスタンスリストと呼ぶ。

アプリケーションインスタンスリストを構成する AppInfo 構造体は、アプリケーション種別(appType)とアプリケーションインスタンスへのポインタ(appDetail)を要素に持つ。各アプリケーションインスタンスは、インスタンス生成時に AppInfo を介してこのリストの先頭に繋がれる。

トランスポート層からアプリケーション層にイベントが上がってきた際に対象となるアプリケーションインスタンスを探す際にもこのリストが辿られる。

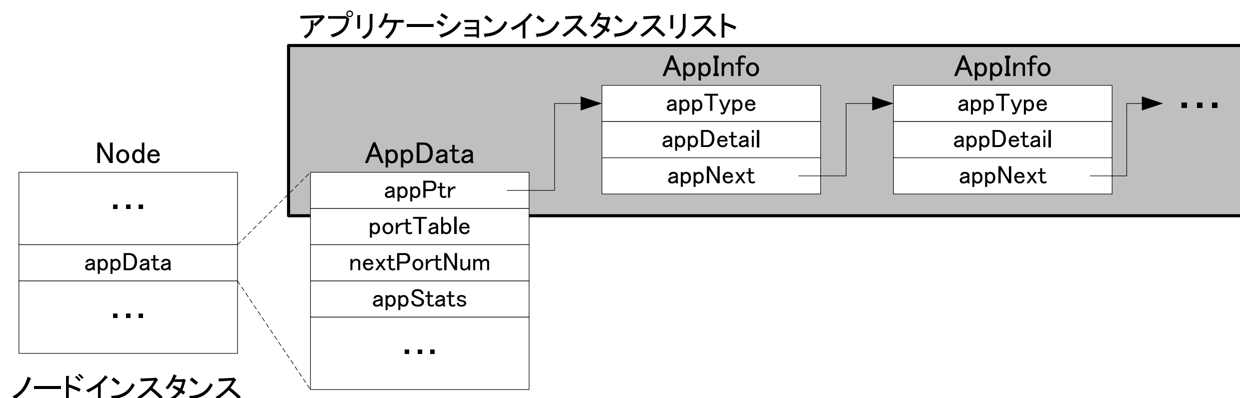

図 2 アプリケーションインスタンスリスト

1.2.2 CBR クライアント構造体(AppDataCbrClient)

CBR クライアントインスタンスは、CBR クライアント構造体(以下 AppDataCbrClient と呼ぶ)をデータ型とするインスタンスであり、ノード上に配備する際には、前述のように AppInfo を介してそのノードのアプリケーションインスタンスリストに繋ぐ。AppInfo と CBR クライアントインスタンスの関係を図 3 に示す。

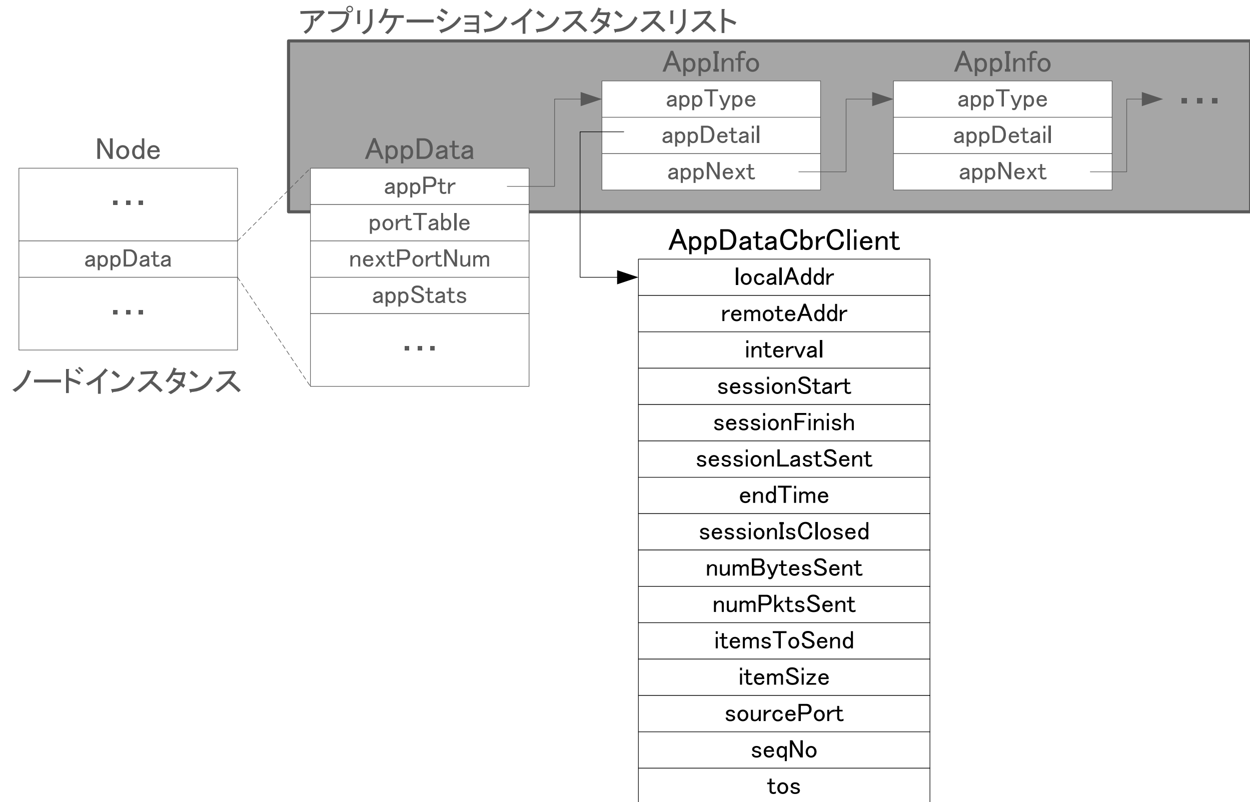

図 3 CBR クライアントインスタンス

CBR クライアントインスタンスのデータ型である AppDataCbrClient を表 2 に記載する。

表 2 CBR クライアント構造体(AppDataCbrClient)

要素名	データ型	説明
localAddr	Address	CBR クライアントのアドレス。
remoteAddr	Address	CBR サーバのアドレス。
interval	D_Clocktype	アイテム送信間隔。
sessionStart	clocktype	セッション開始時刻。初期化時に設定で与えられた値がセットされる。
sessionFinish	clocktype	セッション終了時刻。type が'c'のデータアイテムを送信する時刻あるいはシミュレーション終了時刻がセットされる。
sessionLastSent	clocktype	最後にアイテムを送信した時刻。
endTime	clocktype	アプリケーション終了時刻。初期化時に設定で与えられた CBR アプリケーション終了時刻がセットされる。
sessionIsClosed	BOOL	セッションが閉じられているかどうかを示すフラグ。type が'c'のデータアイテムを送信する際に TRUE となる。
numBytesSent	D_Int64	総送信バイト数。データアイテム送信のたびにアイテムサイズ分だけ加算される。

要素名	データ型	説明
numPktsSent	UInt32	総送信パケット数。データアイテム送信のたびに 1 だけ加算される。
itemsToSend	UInt32	残送信アイテム数。初期化時に設定で与えられたデータアイテム数がセットされ、アイテムを送信するたびに 1 だけ減算される。
itemSize	UInt32	送信アイテムサイズ。単位は Byte。初期化時に設定で与えられたサイズがセットされる。
sourcePort	short	CBR クライアントのポート番号。初期化時に、node->appData.nextPortNum の値がセットされる。(セット後、node->appData.nextPortNum 自体は 1 だけ加算される。)
seqNo	Int32	送信アイテムに割り当てるシーケンス番号。初期値 0 でデータアイテム送信のたびに 1 だけ加算される。
tos	D_Uint32	TOS 値。初期化時に設定で与えられた値がセットされる。

1.2.3 CBR サーバ構造体(AppDataCbrServer)

CBR サーバインスタンスは、CBR サーバ構造体(以下 AppDataCbrServer と呼ぶ)をデータ型とするインスタンスであり、ノード上に配備する際には、前述のように AppInfo を介してそのノードのアプリケーションインスタンスリストに繋ぐ。AppInfo と CBR サーバインスタンスの関係を図 4 に示す。

なお、CBR サーバインスタンスはシミュレーションの初期化時には生成されず、CBR クライアントから初めてデータアイテムを受信した際に動的に生成される。このため、例えば経路途中でのパケット損失などの理由でもし 1 アイテムも受信しなかった場合は、CBR サーバインスタンス事態が生成されず統計情報も得られないので注意が必要である。

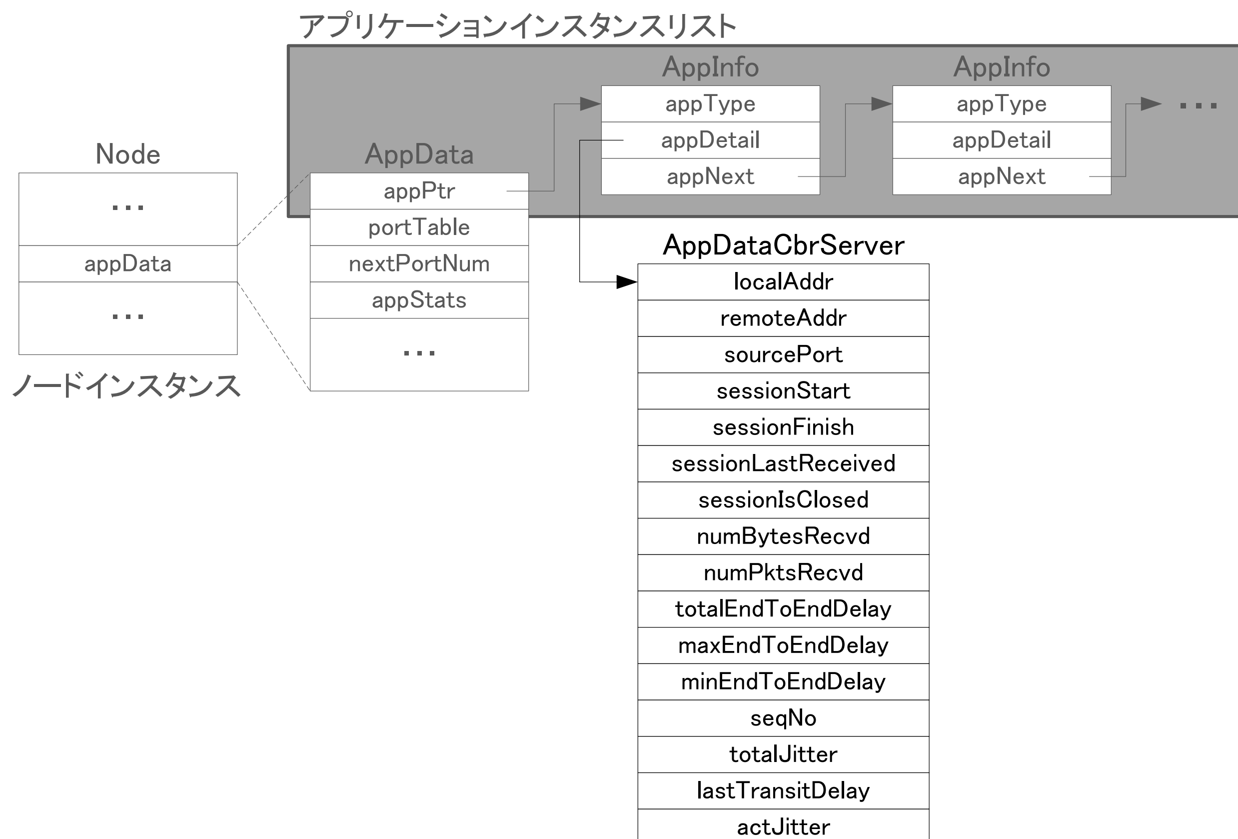

図 4 CBR サーバインスタンス

CBR クライアントインスタンスのデータ型である AppDataCbrServer を表 3 に記載する。

表 3 CBR サーバ構造体(AppDataCbrServer)

要素名	データ型	説明
localAddr	Address	CBR サーバのアドレス。CBR クライアントから最初に届いたデータアイテムから取得した値がセットされる。
remoteAddr	Address	CBR クライアントのアドレス。CBR クライアントから最初に届いたデータアイテムから取得した値がセットされる。
sourcePort	short	CBR クライアントのポート番号。CBR クライアントから最初に届いたデータアイテムから取得した値がセットされる。
sessionStart	clocktype	セッション開始時刻。CBR クライアントから最初にデータアイテムが届いた時刻がセットされる。
sessionFinish	clocktype	セッション終了時刻。type が'c'のデータアイテムを受信した時刻あるいはシミュレーション終了時刻がセットされる。
sessionLastReceived	clocktype	最後にデータアイテムを受信した時刻。順序逆転して届いたデータアイテムは対象外となる。
sessionIsClosed	BOOL	セッションが閉じられているかどうかを示すフラグ。type が'c'のデータアイテムを受信した際に TRUE となる。
numBytesRecvd	D_Int64	総受信バイト数。順序逆転して届いたデータアイテムは対象外となる。
numPktsRecvd	UInt32	総受信パケット数。順序逆転して届いたデータアイテムは対象外となる。
totalEndToEndDelay	clocktype	エンドツーエンド遅延の合計値。単位は nsec。データアイテム受信のたびに更新される。
maxEndToEndDelay	clocktype	エンドツーエンド遅延の最大値。単位は nsec。データアイテム受信のたびにチェックおよび更新される。
minEndToEndDelay	clocktype	エンドツーエンド遅延の最小値。単位は nsec。データアイテム受信のたびにチェックおよび更新される。
seqNo	Int32	次に受信を期待するデータアイテムのシーケンス番号。データアイテム受信のたびに更新される。
totalJitter	clocktype	ジッタの合計値。単位は nsec。
lastTransitDelay	clocktype	前回受信したデータアイテムの伝送遅延。単位は nsec。
actJitter	clocktype	ジッタ。単位は nsec。計算式は「RFC3550 A.8 Estimating the Interarrival Jitter」に依る。

1.2.4 CBR データアイテムと CBR データ構造体(CbrData)

CBR クライアントから CBR サーバに送信するデータアイテムとそのヘッダとして使用されるデータ構造体(以下 CbrData と呼ぶ)の関係を図 5 に示す。

CbrData は CBR アプリケーションの管理に最低限必要な情報で構成された構造体であり、アプリケーションヘッダとして CBR クライアントから送信される全てのデータアイテムに必ず含まれる。このため、CBR のデータアイテムサイズは CbrData のサイズ以上の値を設定する必要があり、それ未満の値が設定値で与えられた場合には初期化処理でエラー終了するようになっている。

CbrData のサイズ(*)は、Advanced Wireless Library および UMTS Library が無効の場合は 16Byte となる(x86/x64 系 CPU の場合)。Advanced Wireless Library あるいは UMTS Library が有効の場合は Int32 型の pktSize と clocktype 型の interval が CbrData の要素に追加されるため、CbrData のサイズ(*)は 32Byte となる(x86/x64 系 CPU の場合)。なお、データアイテムの残りの部分は、QualNet 上では実際にメ

モリ確保はされておらず、そのサイズ値だけが管理されている。(QualNet ではこれを Virtual Payload と呼んでいる。)
*(細かい話になるが)ここで言うサイズとは CbrData の正味のサイズではなく sizeof 演算子が返す値(データアイテムサイズを計算する際に使用されている値と同じで各要素のアラインメントが考慮された値)である。

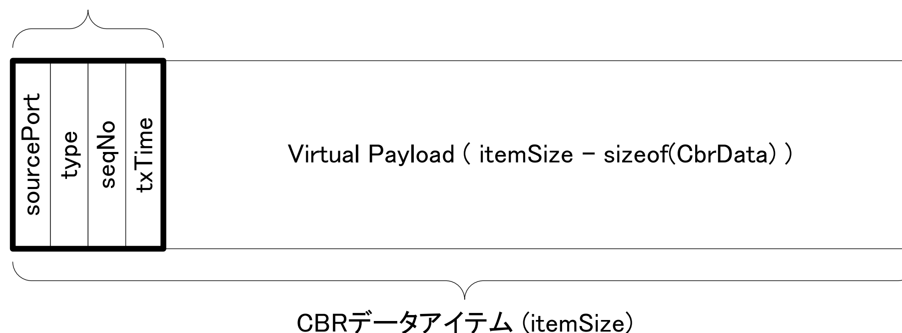

図 5 CBR データアイテム

ヘッダに使用されている CBR データ構造体(CbrData)を表 4 に記載する。

表 4 CBR データ構造体(CbrData)

要素名	データ型	説明
sourcePort	short	CBR クライアントのポート番号。
type	char	文字'd'または'c'をセットする。 'd'は通常のデータアイテムであることを意味する。 'c'はセッション終了(つまり最後のデータアイテム)を意味する。
seqNo	Int32	データアイテムを識別するためのシーケンス番号。
txTime	clocktype	CBR クライアントインスタンスが送信した時刻。

1.3 処理概要

CBR に限らず QualNet 上に実装する各種プロトコルの処理は、大きく分けて以下の 3 種類の処理で構成される。これらの処理は、ノードごとに個別に行われる。

・初期化処理(Initialize)
・イベント処理(ProcessEvent)
・終了処理(Finalize)

以下では、これら 3 種類の処理についておおまかな処理の流れを解説する。

1.3.1 初期化処理

シミュレーションの初期化時には、ノード上のアプリケーション層の初期化処理関数である APP_InitializeApplications 関数が呼び出される。

CBR クライアントが配置されているノード上では、この APP_InitializeApplications 関数から CBR クライアントの初期化処理関数である AppCbrClientInit 関数が呼び出され、その中でクライアントインスタンスの生成を行う AppCbrClientNewCbrClient 関数が呼び出される。AppCbrClientNewCbrClient 関数は APP_RegisterNewApp 関数を呼び出して生成したインスタンスを当該ノードのアプリケーションインスタンスリストに登録する。

CBR サーバが配置されているノード上では、APP_InitializeApplications 関数から CBR サーバの初期化処理関数である AppCbrServerInit 関数が呼び出されるが、この関数の中では何も処理を行っていない。(サーバインスタンスの生成および初期化は CBR クライアントからのデータアイテムが初めて届いた際に行われる。)

初期化処理に関連する関数の呼び出し関係を図 6 に示す。

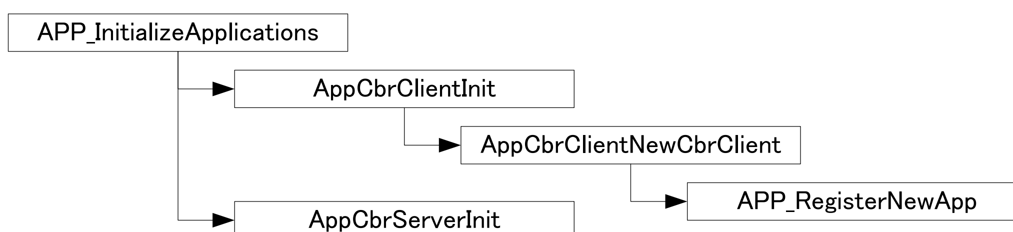

図 6　初期化処理

1.3.2 イベント処理

ノード上でアプリケーション層イベントが発生すると、APP_ProcessEvent 関数が呼び出される。その中で、当該イベントが何かを判断し、CBR クライアントのイベント処理関数である AppLayerCbrClient 関数あるいは CBR サーバのイベント処理関数である AppLayerCbrServer 関数を必要に応じて呼び出す。

AppLayerCbrClient 関数は、CBR クライアントインスタンスを取得する AppCbrClientGetCbrClient 関数、下位層プロトコルの UDP にデータアイテムを引き渡す APP_UdpSendNewHeaderVirtualDataWithPriority 関数、次のデータアイテム送信のためのタイマをセットする AppCbrClientScheduleNextPkt 関数を呼び出す。

また AppLayerCbrServer 関数は、CBR サーバインスタンスを取得する AppCbrServerGetCbrServer 関数、CBR サーバインスタンスを新規に生成する AppCbrServerNewCbrServer 関数の呼び出しを行う。AppCbrServerNewCbrServer 関数は、生成したインスタンスを APP_RegisterNewApp 関数の呼び出しにて当該ノードのアプリケーションインスタンスリストに登録する。

イベント処理に関連する関数の呼び出し関係を図 7 に示す。

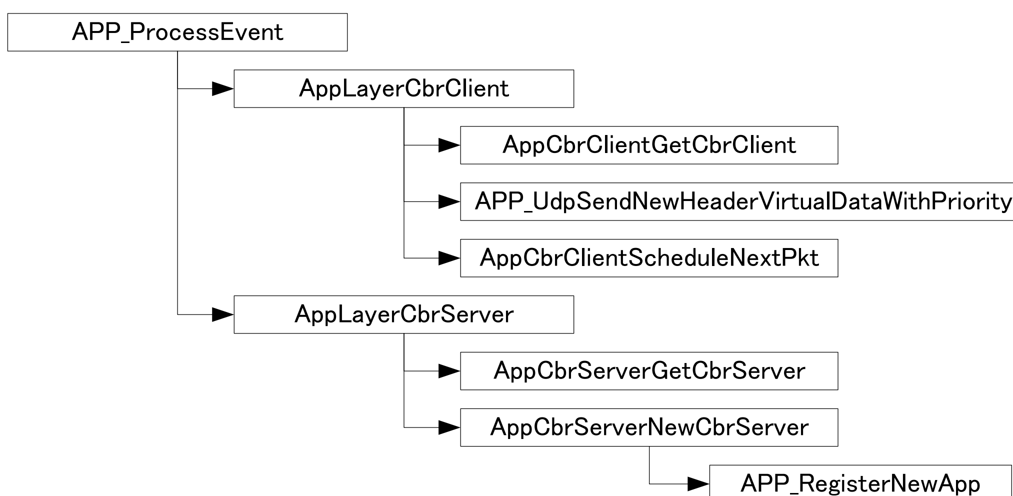

図 7　イベント処理

1.3.3 終了処理

シミュレーション終了時には、ノード上のアプリケーション層の終了処理関数である APP_Finalize 関数が呼び出される。
CBR クライアントが配置されているノード上では、この APP_Finalize 関数から CBR クライアントの終了処理関数である AppCbrClientFinalize 関数が呼び出され、その中で CBR クライアント統計情報を出力する AppCbrClientPrintStats 関数が呼び出される。
CBR サーバが配置されているノード上では、APP_Finalize 関数から CBR サーバの終了処理関数である AppCbrServerFinalize 関数が呼び出され、その中で CBR サーバ統計情報を出力する AppCbrServerPrintStats 関数が呼び出される。
終了処理に関連する関数の呼び出し関係を図 8 に示す。

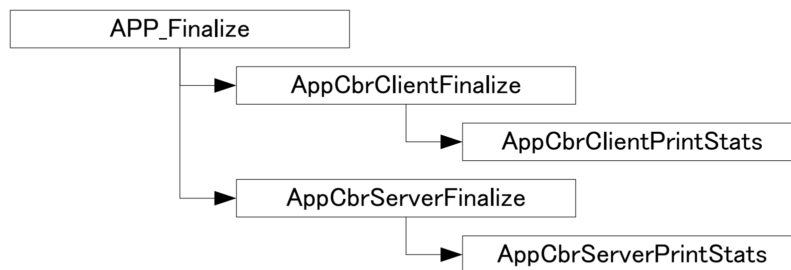

図 8 終了処理

1.4 初期化処理詳細

1.4.1 パラメータ読み込み

パラメータの読み込みは、application.cpp の 1860 行目から 2038 行目で行われる。読み込まれたパラメータは AppCbrClientInit 関数の引数として渡され、より具体的な初期化処理が実施される。

application.cpp

```
01429 /*
01430  * NAME:         APP_InitializeApplications.
01431  * PURPOSE:      start applications on nodes according to user's
01432  *               specification.
01433  * PARAMETERS:   node - pointer to the node,
01434  *               nodeInput - configuration information.
01435  * RETURN:       none.
01436  */
01437 void
01438 APP_InitializeApplications(
01439     Node *firstNode,
01440     const NodeInput *nodeInput)
01441 {
01442     NodeInput appInput;
01443     char appStr[MAX_STRING_LENGTH];
01444
01445     BOOL retVal;
01446     int  i;
01447     int  numValues;
01448     Node* node = NULL;
01449     IdToNodePtrMap* nodeHash;
01450

.... 中略 ....

01505     nodeHash = firstNode->partitionData->nodeIdHash;
```

```
01506
01507     IO_ReadCachedFile(
01508         ANY_NODEID,
01509         ANY_ADDRESS,
01510         nodeInput,
01511         "APP-CONFIG-FILE",
01512         &retVal,
01513         &appInput);
01514     // initialize appinput.numMaxLines to 0
01515     appInput.maxNumLines = 0;
01516
01517     if (retVal == FALSE)
01518     {
01519         return;
01520     }
01521
01522     for (i = 0; i < appInput.numLines; i++)
01523     {
01524         sscanf(appInput.inputStrings[i], "%s", appStr);
01525
01526         if (firstNode->networkData.networkProtocol == IPV6_ONLY)
01527         {
01528             if (strcmp(appStr, "CBR") != 0
01529                 && strcmp(appStr, "FTP/GENERIC") != 0
01530                 && strcmp(appStr, "FTP") != 0
01531                 && strcmp(appStr, "TELNET") != 0
01532                 && strcmp(appStr, "HTTP") != 0
01533                 && strcmp(appStr, "HTTPD") != 0
01534                 && strcmp(appStr, "MCBR") !=0
01535                 && strcmp(appStr, "TRAFFIC-GEN") !=0
01536                 && strcmp(appStr, "SUPER-APPLICATION") !=0)
01537             {
01538                 char buf[MAX_STRING_LENGTH];
01539
01540                 sprintf(buf,
01541                     "%s is not supported for IPv6 based Network.\n"
01542                     "Only CBR, FTP/GENERIC, FTP, TELNET ,HTTP ,MCBR,"
01543                     "SUPER-APPLICATION and TRAFFIC-GEN \n"
01544                     "applications are currently supported.\n", appStr);
01545
01546                 ERROR_ReportError(buf);
01547             }
01548         }
01549
01550 #ifdef DEBUG
01551         printf("Parsing for application type %s\n", appStr);
01552 #endif /* DEBUG */
01553
01554         if (strcmp(appStr, "FTP") == 0)
01555         {

.... 中略 ....

01858         }
01859         else
01860         if (strcmp(appStr, "CBR") == 0)
01861         {
01862             char sourceString[MAX_STRING_LENGTH];
01863             char destString[MAX_STRING_LENGTH];
```

CBR アプリケーション設定の読み込み

行番号	処理内容
1507-1513	パラメータ APP-CONFIG-FILE で指定されたアプリケーション設定ファイルの内容を NodeInput 型の変数 appInput に読み込む。アプリケーション設定ファイルは複数のアプリケー

ションの設定を記述可能(1 行が 1 つのアプリケーション定義)であり、appInput には、全ての行の情報が読み込まれている。具体的には、appInput.inputStrings[i] に i 行目の設定ファイルの内容が読み込まれている。

1522 各アプリケーション設定に関してループを回す。

1524 先頭列に位置する、アプリケーション種別を表す文字列を appStr 変数に読み込む。CBR アプリケーションの場合は、appStr は"CBR"である。

1526-1548 一部のアプリケーションは、IPv6 がサポートされていない。IPv6 を用いる設定でかつ、IPv6 非対応のアプリケーションが設定されている場合はエラーを表示してプログラムを終了する。

1544- appStr の内容に応じて、個別アプリケーションの初期化処理が行われる。CBR の場合、1860 行目の if 文のブロックが対応する初期化処理である。

application.cpp

```
01860         if (strcmp(appStr, "CBR") == 0)
01861         {
01862             char sourceString[MAX_STRING_LENGTH];
01863             char destString[MAX_STRING_LENGTH];
01864             char intervalStr[MAX_STRING_LENGTH];
01865             char startTimeStr[MAX_STRING_LENGTH];
01866             char endTimeStr[MAX_STRING_LENGTH];
01867             int itemsToSend;
01868             int itemSize;
01869             NodeAddress sourceNodeId;
01870             Address sourceAddr;
01871             NodeAddress destNodeId;
01872             Address destAddr;
01873             unsigned tos = APP_DEFAULT_TOS;
01874             BOOL isRsvpTeEnabled = FALSE;
01875             char optionToken1[MAX_STRING_LENGTH];
01876             char optionToken2[MAX_STRING_LENGTH];
01877             char optionToken3[MAX_STRING_LENGTH];
01878
01879
01880             numValues = sscanf(appInput.inputStrings[i],
01881                             "%*s %s %s %d %d %s %s %s %s %s %s",
01882                             sourceString,
01883                             destString,
01884                             &itemsToSend,
01885                             &itemSize,
01886                             intervalStr,
01887                             startTimeStr,
01888                             endTimeStr,
01889                             optionToken1,
01890                             optionToken2,
01891                             optionToken3);
01892
01893             switch (numValues)
01894             {
01895                 case 7:
01896                     break;
01897                 case 8:
01898                     if (!strcmp(optionToken1, "RSVP-TE"))
01899                     {
01900                         isRsvpTeEnabled = TRUE;
01901                         break;
01902                     } // else fall through default
01903                 case 9 :
01904                     if (APP_AssignTos(optionToken1, optionToken2, &tos))
```

```
01905                     {
01906                         break;
01907                     } // else fall through default
01908                 case 10 :
01909                     if (!strcmp(optionToken1, "RSVP-TE"))
01910                     {
01911                         isRsvpTeEnabled = TRUE;
01912
01913                         if (APP_AssignTos(optionToken2, optionToken3, &tos
))
01914                         {
01915                             break;
01916                         }// else fall through default
01917                     }
01918                     else
01919                     {
01920                         if (APP_AssignTos(optionToken1, optionToken2, &tos
))
01921                         {
01922                             if (!strcmp(optionToken3, "RSVP-TE"))
01923                             {
01924                                 isRsvpTeEnabled = TRUE;
01925                                 break;
01926                             }
01927                         }// else fall through default
01928                     }
01929                 default:
01930                 {
01931                     char errorString[MAX_STRING_LENGTH];
01932                     sprintf(errorString,
01933                         "Wrong CBR configuration format!\n"
01934                         "CBR <src><dest><items to send> "
01935                         "<item size><interval><start time> "
01936                         "<end time> [TOS <tos-value> | DSCP <dscp-value>"
01937                         " | PRECEDENCE <precedence-value>] [RSVP-TE]\n");
01938                     ERROR_ReportError(errorString);
01939                 }
01940             }
01941
01942             IO_AppParseSourceAndDestStrings(
01943                 firstNode,
01944                 appInput.inputStrings[i],
01945                 sourceString,
01946                 &sourceNodeId,
01947                 &sourceAddr,
01948                 destString,
01949                 &destNodeId,
01950                 &destAddr);
01951
```

CBR アプリケーション設定の読み込み

行番号	処理内容
1880-1891	sscanf 関数を使い、設定内容を読み込む。なお、CBR のアプリケーション設定は全部で 10 項目(1 列目の”CBR”を除き)あるが、8,9,10 番目の設定は省略可能(オプショナル)である。
1893-1940	オプショナルパラメータの設定数の違いにより、内部変数に設定する値を適切に処理している。許される設定数は 7,8,9,10 のみである。それ以外の場合は、不正なフォーマットである旨のメッセージを出力し、プログラムを終了する。
1942-1950	文字列で表現された送信元並びに送信先のノード(またはインターフェース)から、QualNet 内部で用いるノード ID および、アドレス型の変数を取得する。

application.cpp

```
01960             node = MAPPING_GetNodePtrFromHash(nodeHash, sourceNodeId);
01961             if (node != NULL)
01962             {
01963                 clocktype startTime = TIME_ConvertToClock(startTimeStr);
01964                 clocktype endTime = TIME_ConvertToClock(endTimeStr);
01965                 clocktype interval = TIME_ConvertToClock(intervalStr);
01966

.... 中略 ....

01995
01996                 AppCbrClientInit(
01997                     node,
01998                     sourceAddr,
01999                     destAddr,
02000                     itemsToSend,
02001                     itemSize,
02002                     interval,
02003                     startTime,
02004                     endTime,
02005                     tos,
02006                     isRsvpTeEnabled);
02007            }
```

CBR クライアント初期化

行番号	処理内容
1960	送信元ノード ID から、Node 構造体へのポインタを取得する。
1963-1965	文字列で表現されたセッション開始時間、終了時間、送信間隔を clocktype 型へ変換する。
1996-2006	CBR クライアントの初期化を行う。(エラー! 参照元が見つかりません。参照)

application.cpp

```
02009             if (sourceAddr.networkType != destAddr.networkType)
02010             {
02011                 char err[MAX_STRING_LENGTH];
02012
02013                 sprintf(err,
02014                         "CBR: At node %d, Source and Destination IP"
02015                         " version mismatch inside %s\n",
02016                     node->nodeId,
02017                     appInput.inputStrings[i]);
02018
02019                 ERROR_Assert(FALSE, err);
02020             }
02021
02022             // Handle Loopback Address
02023             if (node == NULL || !APP_SuccessfullyHandledLoopback(
02024                                     node,
02025                                     appInput.inputStrings[i],
02026                                     destAddr,
02027                                     destNodeId,
02028                                     sourceAddr,
02029                                     sourceNodeId))
02030             {
02031                 node = MAPPING_GetNodePtrFromHash(nodeHash, destNodeId);
02032             }
02033
02034             if (node != NULL)
02035             {
```

```
02036                 AppCbrServerInit(node);
02037             }
02038         }
02039         else
```

CBR クライアント初期化

行番号	処理内容
2009-2020	送信元と送信先アドレスのネットワークタイプが異なる場合は、エラーメッセージを出力して終了する。
2022-2032	Node が NULL の場合かつループバック設定ではない場合には node 構造体には送信先ノードの Node 構造体を格納する。
2034-2037	CBR サーバの初期化処理(1.4.3 参照)

アプリケーションは、通常、送信側をクライアント、受信側をサーバと呼ぶ。この AppCbrClientInit 関数は、クライアント側での CBR アプリケーションの初期化を行う関数である。

app_cbr.cpp

```
00333 /*
00334  * NAME:         AppCbrClientInit.
00335  * PURPOSE:      Initialize a CbrClient session.
00336  * PARAMETERS:  node - pointer to the node,
00337  *              serverAddr - address of the server,
00338  *              itemsToSend - number of items to send,
00339  *              itemSize - size of each packet,
00340  *              interval - interval of packet transmission rate.
00341  *              startTime - time until the session starts,
00342  *              endTime - time until the session ends,
00343  *              tos - the contents for the type of service field.
00344  * RETURN:      none.
00345  */
00346 void
00347 AppCbrClientInit(
00348     Node *node,
00349     Address clientAddr,
00350     Address serverAddr,
00351     Int32 itemsToSend,
00352     Int32 itemSize,
00353     clocktype interval,
00354     clocktype startTime,
00355     clocktype endTime,
00356     unsigned tos,
00357     BOOL isRsvpTeEnabled)
00358 {
00359     char error[MAX_STRING_LENGTH];
00360     AppTimer *timer;
00361     AppDataCbrClient *clientPtr;
00362     Message *timerMsg;
00363     int minSize;
00364     startTime -= getSimStartTime(node);
00365     endTime -= getSimStartTime(node);
```

CBR クライアントの開始時間、終了時間のセット

行番号	処理内容
364-265	app パラメータファイルで指定されたセッションの開始時刻、終了時刻からシミュレーションの開始時間を引く。

```
00369     /* Check to make sure the number of cbr items is a correct value. */
00370     if (itemsToSend < 0)
00371     {
00372         sprintf(error, "CBR Client: Node %d item to sends needs"
00373             " to be >= 0\n", node->nodeId);
00374
00375         ERROR_ReportError(error);
00376     }
00377
00378     /* Make sure that packet is big enough to carry cbr data information.
*/
00379     if (itemSize < minSize)
00380     {
00381         sprintf(error, "CBR Client: Node %d item size needs to be >= %d.\n
",
00382                 node->nodeId, minSize);
00383         ERROR_ReportError(error);
00384     }
00385
00386
00387     /* Make sure that packet is within max limit. */
00388     if (itemSize > APP_MAX_DATA_SIZE)
00389     {
00390         sprintf(error, "CBR Client: Node %d item size needs to be <= %d.\n
",
00391                 node->nodeId, APP_MAX_DATA_SIZE);
00392         ERROR_ReportError(error);
00393     }
00394
00395     /* Make sure interval is valid. */
00396     if (interval <= 0)
00397     {
00398         sprintf(error, "CBR Client: Node %d interval needs to be > 0.\n",
00399                 node->nodeId);
00400         ERROR_ReportError(error);
00401     }
00402
00403     /* Make sure start time is valid. */
00404     if (startTime < 0)
00405     {
00406         sprintf(error, "CBR Client: Node %d start time needs to be >= 0.\n
",
00407                 node->nodeId);
00408         ERROR_ReportError(error);
00409     }
00410
00411     /* Check to make sure the end time is a correct value. */
00412     if (!((endTime > startTime) || (endTime == 0)))
00413     {
00414         sprintf(error, "CBR Client: Node %d end time needs to be > "
00415             "start time or equal to 0.\n", node->nodeId);
00416
00417         ERROR_ReportError(error);
00418     }
00419
00420     // Validate the given tos for this application.
00421     if (/*tos < 0 || */tos > 255)
00422     {
00423         sprintf(error, "CBR Client: Node %d should have tos value "
00424             "within the range 0 to 255.\n",
00425             node->nodeId);
00426         ERROR_ReportError(error);
00427     }
```

引数のチェック

行番号	処理内容
369-427	引数で指定されきた値が有効か値かをチェックする。引数で指定されてくる値は app パラメータファイルから読み込んできた値である。無効な値を検出した場合は、ERROR_ReportError 関数を用いてエラー内容を表示しつつプログラムを終了する。

app_cbr.cpp

```
00428
.... 中略 ....
00472     clientPtr = AppCbrClientNewCbrClient(node,
00473                                          clientAddr,
00474                                          serverAddr,
00475                                          itemsToSend,
00476                                          itemSize,
00477                                          interval,
00478 #ifndef ADDON_NGCNMS
00479                                          startTime,
00480 #else
00481                                          origStart,
00482 #endif
00483                                          endTime,
00484                                          (TosType) tos);
00485
00486     if (clientPtr == NULL)
00487     {
00488         sprintf(error,
00489                 "CBR Client: Node %d cannot allocate memory for "
00490                     "new client\n", node->nodeId);
00491
00492         ERROR_ReportError(error);
00493     }
```

AppDataCbrClient データ構造作成

行番号	処理内容
472-484	AppCbrClientNewCbrClient 関数(1.4.2 参照)を呼び出し、AppDataCbrClient 構造体の作成を行う。引数として AppDataCbrClient 構造体へのポインタが得られる。
486-493	何らかの理由で AppDataCbrClient 構造体へのポインタが NULL の場合には、エラーメッセージを出力してプログラムを終了する。

app_cbr.cpp

```
00495     if (node->transportData.rsvpProtocol && isRsvpTeEnabled)
00496     {
00497         // Note: RSVP is enabled for Ipv4 networks only.
00498         Message *rsvpRegisterMsg;
00499         AppToRsvpSend *info;
00500
00501         rsvpRegisterMsg = MESSAGE_Alloc(node,
00502                                   TRANSPORT_LAYER,
00503                                   TransportProtocol_RSVP,
00504                                   MSG_TRANSPORT_RSVP_InitApp);
00505
00506         MESSAGE_InfoAlloc(node,
00507                           rsvpRegisterMsg,
00508                           sizeof(AppToRsvpSend));
00509
```

```
00510         info = (AppToRsvpSend *) MESSAGE_ReturnInfo(rsvpRegisterMsg);
00511
00512         info->sourceAddr = GetIPv4Address(clientAddr);
00513         info->destAddr = GetIPv4Address(serverAddr);
00514
00515         info->upcallFunctionPtr = NULL;
00516
00517         MESSAGE_Send(node, rsvpRegisterMsg, startTime);
00518     }
```

RSVP 関連処理

行番号	処理内容
495-518	RSVP が有効である場合には、RSVB Register メッセージがサーバ側に送信される。

app_cbr.cpp

```
00520
00521     timerMsg = MESSAGE_Alloc(node,
00522                              APP_LAYER,
00523                              APP_CBR_CLIENT,
00524                              MSG_APP_TimerExpired);
00525
00526     MESSAGE_InfoAlloc(node, timerMsg, sizeof(AppTimer));
00527
00528     timer = (AppTimer *)MESSAGE_ReturnInfo(timerMsg);
00529
00530     timer->sourcePort = clientPtr->sourcePort;
00531     timer->type = APP_TIMER_SEND_PKT;
00532
.... 中略 ....
00541
00542     MESSAGE_Send(node, timerMsg, startTime);
00543
.... 中略 ....
00549 }
```

最初のパケット送信タイミングへのタイマ設定

行番号	処理内容
520-542	最初に送信するパケットの送信タイミング(現在時刻 0 + startTime 後)へのタイマを張る。タイマが発火すると、AppLayerCbrClient 関数(1.5.1 参照)が呼び出され、パケットの送信処理が行われる。2 個目以降のパケット送信タイミングへのタイマは、AppLayerCbrClient 関数で行われる。

1.4.2 AppCbrClientNewCbrClient()

AppCbrClientNewCbrClient 関数は、AppDataCbrClient 構造体の作成およびアプリケーションの登録を行うための関数である。AppCbrClientInit 関数から呼び出される。

app_cbr.cpp

```
00724 /*
00725  * NAME:        AppCbrClientNewCbrClient.
00726  * PURPOSE:     create a new cbr client data structure, place it
00727  *              at the beginning of the application list.
00728  * PARAMETERS:  node - pointer to the node.
00729  *              remoteAddr - remote address.
00730  *              itemsToSend - number of cbr items to send in simulation.
00731  *              itemSize - size of each packet.
```

```
00732  *               interval - time between two packets.
00733  *               startTime - when the node will start sending.
00734  * RETURN:       the pointer to the created cbr client data structure,
00735  *               NULL if no data structure allocated.
00736  */
00737 AppDataCbrClient *
00738 AppCbrClientNewCbrClient(Node *node,
00739                          Address localAddr,
00740                          Address remoteAddr,
00741                          Int32 itemsToSend,
00742                          Int32 itemSize,
00743                          clocktype interval,
00744                          clocktype startTime,
00745                          clocktype endTime,
00746                          TosType tos)
00747 {
00748     AppDataCbrClient *cbrClient;
00749 
00750     cbrClient = (AppDataCbrClient *)
00751                 MEM_malloc(sizeof(AppDataCbrClient));
00752     memset(cbrClient, 0, sizeof(AppDataCbrClient));
```

CBR Client のデータ領域確保

行番号	処理内容
748-752	CBR Client アプリケーションに関するデータは、AppDataCbrClient 構造体に格納される。まずは、AppDataCbrClient 構造体変数を MEM_malloc 関数を用いてメモリ領域に確保し、cbrClient ポインタにアドレスをセットする。また、確保した領域を全て 0 でクリアする。

app_cbr.cpp

```
00753 
00754     /*
00755      * fill in cbr info.
00756      */
00757     memcpy(&(cbrClient->localAddr), &localAddr, sizeof(Address));
00758     memcpy(&(cbrClient->remoteAddr), &remoteAddr, sizeof(Address));
00759     cbrClient->interval = interval;
00760 #ifndef ADDON_NGCNMS
00761     cbrClient->sessionStart = getSimTime(node) + startTime;
00762 #else
00763     if (!NODE_IsDisabled(node))
00764     {
00765         cbrClient->sessionStart = getSimTime(node) + startTime;
00766     }
00767     else
00768     {
00769         // start time was already figured out in caller function.
00770         cbrClient->sessionStart = startTime;
00771     }
00772 #endif
00773     cbrClient->sessionIsClosed = FALSE;
00774     cbrClient->sessionLastSent = getSimTime(node);
00775     cbrClient->sessionFinish = getSimTime(node);
00776     cbrClient->endTime = endTime;
00777     cbrClient->numBytesSent = 0;
00778     cbrClient->numPktsSent = 0;
00779     cbrClient->itemsToSend = itemsToSend;
00780     cbrClient->itemSize = itemSize;
00781     cbrClient->sourcePort = node->appData.nextPortNum++;
00782     cbrClient->seqNo = 0;
00783     cbrClient->tos = tos;
```

CBR Client の初期値設定

行番号	処理内容
757-758	CbrClient のメンバ変数に、自ノードのアドレスと、対向ノード(アプリケーションパケットの送信先) のアドレスを格納する。
759	パケットの送信間隔を interval メンバ変数に格納する。
760-772	セッションのスタート時間を設定する。getSimTime で取得できる現在時刻(常に 0)から、app パラメータファイルで指定された開始時刻分経過後の時刻がスタート時間となる。
773	セッション関連の値の設定を行う。sessionIsClosed は、セッションが終了したかどうかを表す BOOL 値であり、初期化時はまだクローズしていないため、FALSE を設定する。AppLayerCbrClient 関数で最後のパケットが送信されたとき、このフラグは TRUE に設定される。
774	最後にパケットが送信された時刻。初期値として現在時刻 0 が設定されるが、時刻 0 でパケットが送信されるとは限らない点に留意する。AppLayerCbrClient 関数でパケットが送信されるたびに更新される。
775	セッションの終了時刻。初期値として時刻 0 がセットされる。AppLayerCbrClient 関数で最後のパケットが送信されたとき、その時刻がセットされる。
777-778	送信バイト数およびパケット数。初期値 0 にセットされる。
779-780	送信するパケット(アイテム)数。app パラメータファイルで指定された値がセットされる。AppLayerCbrClient 関数でパケットを送信するたびにデクリメントされてゆく。
781	ポート番号をセットする。ノード毎に保持しているカウンタ node->appDatanextPortNum の値がセットされる。この時同時に node->appDatanextPortNum の値は 1 インクリメントされ、次に別のアプリケーションが定義された場合、それに使われるポート番号になる。
782	パケットに設定されるシーケンス番号。まずは 0 に設定され、パケット送信のたびにインクリメントされる。
783	TOS(Type of Service)値を設定する。この値は app パラメータファイルで指定された値になる。

app_cbr.cpp

```
00784
00785     // Add CBR variables to hierarchy
00786     std::string path;
00787     D_Hierarchy *h = &node->partitionData->dynamicHierarchy;
00788
00789     if (h->CreateApplicationPath(
00790             node,                   // node
00791             "cbrClient",            // protocol name
00792             cbrClient->sourcePort,  // port
00793             "interval",             // object name
00794             path))                  // path (output)
00795     {
00796         h->AddObject(
00797             path,
00798             new D_ClocktypeObj(&cbrClient->interval));
00799     }
00800
```

```
00801 // The HUMAN_IN_THE_LOOP_DEMO is part of a gui user-defined command
00802 // demo.
00803 // The type of service value for this CBR application is added to
00804 // the dynamic hierarchy so that the user-defined-command can change
00805 // it during simulation.
00806 #ifdef HUMAN_IN_THE_LOOP_DEMO
00807     if (h->CreateApplicationPath(
00808             node,
00809             "cbrClient",
00810             cbrClient->sourcePort,
00811             "tos",                  // object name
00812             path))                  // path (output)
00813     {
00814         h->AddObject(
00815             path,
00816             new D_UInt32Obj(&cbrClient->tos));
00817     }
00818 #endif
00819
00820     if (h->CreateApplicationPath(
00821             node,
00822             "cbrClient",
00823             cbrClient->sourcePort,
00824             "numBytesSent",
00825             path))
00826     {
00827         h->AddObject(
00828             path,
00829             new D_Int64Obj(&cbrClient->numBytesSent));
00830     }
00831
00832 #ifdef DEBUG
00833     {
00834         char clockStr[MAX_STRING_LENGTH];
00835         char localAddrStr[MAX_STRING_LENGTH];
00836         char remoteAddrStr[MAX_STRING_LENGTH];
00837
00838         IO_ConvertIpAddressToString(&cbrClient->localAddr, localAddrStr);
00839         IO_ConvertIpAddressToString(&cbrClient->remoteAddr, remoteAddrStr)
;
00840
00841         printf("CBR Client: Node %u created new cbr client structure\n",
00842                node->nodeId);
00843         printf("    localAddr = %s\n", localAddrStr);
00844         printf("    remoteAddr = %s\n", remoteAddrStr);
00845         TIME_PrintClockInSecond(cbrClient->interval, clockStr);
00846         printf("    interval = %s\n", clockStr);
00847         TIME_PrintClockInSecond(cbrClient->sessionStart, clockStr, node);
00848         printf("    sessionStart = %sS\n", clockStr);
00849         printf("    numBytesSent = %u\n", cbrClient->numBytesSent);
00850         printf("    numPktsSent = %u\n", cbrClient->numPktsSent);
00851         printf("    itemsToSend = %u\n", cbrClient->itemsToSend);
00852         printf("    itemSize = %u\n", cbrClient->itemSize);
00853         printf("    sourcePort = %ld\n", cbrClient->sourcePort);
00854         printf("    seqNo = %ld\n", cbrClient->seqNo);
00855     }
00856 #endif /* DEBUG */
00857
00858     APP_RegisterNewApp(node, APP_CBR_CLIENT, cbrClient);
00859
00860     return cbrClient;
00861 }
```

Hierarchy に関する処理

行番号	処理内容
784-830	Dynamic Object である cbrClient->interval を Dynamic ObjectHierarchy に登録する。。
832-856	デバッグプリント。DEBUG を定義した場合、標準出力に cbrClient のメンバの値が出力される。
858	AppDataCbrClient 構造体変数へのポインタを、設定先の Node 構造体変数へのポインタおよびアプリケーションの種類を表す列挙値 APP_CBR_CLIENT と共に、APP_RegisterNewApp に渡し、アプリケーションの登録を行う。
860	AppDataCbrClient 構造体変数へのポインタを戻り値として返す。

1.4.3 AppCbrServerInit()

初期化処理に関して、サーバ側では特に行うことはない。サーバ側は、パケットを受信するまで、アプリケーションが張られていることを認識していない。
クライアント側の初期化処理に相当するサーバ側の初期化は、最初のアプリケーションパケットを受信した際に、AppLayerCbrServer 関数によって行われる。

app_cbr.cpp

```
01082 /*
01083  * NAME:         AppCbrServerInit.
01084  * PURPOSE:      listen on CbrServer server port.
01085  * PARAMETERS:  node - pointer to the node.
01086  * RETURN:       none.
01087  */
01088 void
01089 AppCbrServerInit(Node *node)
01090 {
01091 }
```

1.5 イベント処理詳細

1.5.1 AppLayerCbrClient()

本関数は CBR Client のイベント処理関数である。CBR Client のイベントは全て本関数にまで通知される。CBR Client ではタイマ発火イベントのみが実装されている。

本関数は、パケットの送信時刻毎に呼ばれ、パケットの送信処理を行う。最初のパケット送信は上述の初期化処理でスケジューリングされ、それ以降の定期送信は本関数でパケット送信後にスケジューリングされる。

app_cbr.cpp

```
00137 /*
00138  * NAME:         AppLayerCbrClient.
00139  * PURPOSE:      Models the behaviour of CbrClient Client on receiving the
00140  *               message encapsulated in msg.
00141  * PARAMETERS:  node - pointer to the node which received the message.
00142  *               msg - message received by the layer
00143  * RETURN:       none.
00144  */
00145 void
```

```
00146 AppLayerCbrClient(Node *node, Message *msg)
00147 {
.... 中略 ....
00158     switch (msg->eventType)
00159     {
00160         case MSG_APP_TimerExpired:
00161         {
.... 中略 ....
00181             switch (timer->type)
00182             {
00183                 case APP_TIMER_SEND_PKT:
00184                 {
.... 中略 ....
00278                     if (clientPtr->sessionIsClosed == FALSE)
00279                     {
00280                         AppCbrClientScheduleNextPkt(node, clientPtr);
00281                     }
.... 中略 ....
00284                 }
.... 中略 ....
00288             }
.... 中略 ....
00291         }
.... 中略 ....
00298     }
00299
00300     MESSAGE_Free(node, msg);
00301 }
```

CBR Client のイベント処理の流れ

行番号	処理内容
158	msg に設定されている eventType により処理を分岐している。CBR Client のタイマイベントは MSG_APP_TimerExpired として指定されている。送信開始時刻のタイマは初期化処理で設定しており、以降本関数において次の送信時刻のタイマを設定している。
280	AppCbrClientScheduleNextPkt において、初期化処理と同様に次の送信時刻のタイマを設定している。ただし、送信開始時間ではなく送信間隔が指定される。278 行目に関しては後述する。
300	イベント処理関数で処理を行った Message は必ず解放する。

app_cbr.cpp

```
00162             AppTimer *timer;
00163
00164             timer = (AppTimer *) MESSAGE_ReturnInfo(msg);
00165
00166 #ifdef DEBUG
00167             printf("CBR Client: Node %ld at %s got timer %d¥n",
00168                    node->nodeId, buf, timer->type);
00169 #endif /* DEBUG */
00170
00171             clientPtr = AppCbrClientGetCbrClient(node, timer->sourcePort);
00172
00173             if (clientPtr == NULL)
00174             {
00175                 sprintf(error, "CBR Client: Node %d cannot find cbr"
00176                     " client¥n", node->nodeId);
00177
00178                 ERROR_ReportError(error);
00179             }
```

CBR Client 情報の取得

行番号	処理内容
164	MESSAGE_ReturnInfo によりタイマ設定時に格納した AppTimer を取り出す。
171	初期化時に登録した AppDataCbrClient 構造体を、timer に格納する sourcePort をキーに取り出す。
173-179	エラー処理。

app_cbr.cpp

```
00181             switch (timer->type)
00182             {
00183                 case APP_TIMER_SEND_PKT:
00184                 {
.... 中略 ....
00283                     break;
00284                 }
00285
00286                 default:
00287                     assert(FALSE);
00288             }
```

CBR Client のタイマ種別による分岐

行番号	処理内容
181	本行からの switch 文では上述の AppTimer 構造体に含まれる type 変数ごとに処理を分岐している。ただし、CBR では APP_TIMER_SEND_PKT のみ使用している。 以下で、APP_TIMER_SEND_PKT タイマの処理について述べる。

app_cbr.cpp

```
00185                     CbrData data;
00186
00187 #ifdef DEBUG
00188                     printf("CBR Client: Node %u has %u items left to"
00189                         " send¥n", node->nodeId, clientPtr->itemsToSend);
00190 #endif /* DEBUG */
00191
00192                     if (((clientPtr->itemsToSend > 1) &&
00193                          (getSimTime(node) + clientPtr->interval
00194                             < clientPtr->endTime)) ||
00195                         ((clientPtr->itemsToSend == 0) &&
00196                          (getSimTime(node) + clientPtr->interval
00197                             < clientPtr->endTime)) ||
00198                         ((clientPtr->itemsToSend > 1) &&
00199                          (clientPtr->endTime == 0)) ||
00200                         ((clientPtr->itemsToSend == 0) &&
00201                          (clientPtr->endTime == 0)))
00202                     {
00203                         data.type = 'd';
00204                     }
00205                     else
00206                     {
00207                         data.type = 'c';
00208                         clientPtr->sessionIsClosed = TRUE;
00209                         clientPtr->sessionFinish = getSimTime(node);
00210                     }
00212                     data.sourcePort = clientPtr->sourcePort;
00213                     data.txTime = getSimTime(node);
```

```
00214                     data.seqNo = clientPtr->seqNo++;
```

最終パケットの判定とヘッダの設定

行番号	処理内容
192-201	本行の if 文において、本送信が最後の送信かどうかを送信済みパケット数、送信終了時刻から判定している。最後の送信でない場合、CbrData 構造体の type に'd'を設定する。最後の送信の場合は、CbrData 構造体の type に'c'を設定し、208, 209 行目で AppDataCbrClient 構造体にセッション終了情報を格納する。
212-221	CbrData 構造体の各変数に AppDataCbrClient 構造体の情報や現在時刻(アプリケーション層での送信時刻)を格納している。

app_cbr.cpp

```
00239                     // Note: An overloaded Function
00240                     APP_UdpSendNewHeaderVirtualDataWithPriority(
00241                         node,
00242                         APP_CBR_SERVER,
00243                         clientPtr->localAddr,
00244                         (short) clientPtr->sourcePort,
00245                         clientPtr->remoteAddr,
00246                         (char *) &data,
00247                         sizeof(data),
00248                         clientPtr->itemSize - sizeof(data),
00249                         clientPtr->tos,
00250                         0,
00251                         TRACE_CBR);
00252
00253                     clientPtr->numBytesSent += clientPtr->itemSize;
00254                     clientPtr->numPktsSent++;
00255                     clientPtr->sessionLastSent = getSimTime(node);
```

下位レイヤへのパケットの送信

行番号	処理内容
240-251	トランスポート層に UDP を使用してアプリケーション層のパケット送信を行う API の 1 つである。 第 8 引数(248 行目)には CBR Client のヘッダを除いたペイロードのサイズを与える。ここで CBR Client のパケットサイズは初期化処理で設定された clientPtr の itemSize であり、CbrData 構造体のサイズがヘッダサイズとなる。よってパケットサイズからヘッダサイズを引いた値がペイロードサイズである。 CBR のヘッダサイズは 32byte であるため、CBR のパケットサイズを指定する際は 32byte 以上の値を指定する必要がある。
253-255	CBR Client の統計値として送信バイト数、送信パケット数、最終送信時刻を更新する。

app_cbr.cpp

```
00273                     if (clientPtr->itemsToSend > 0)
00274                     {
00275                         clientPtr->itemsToSend--;
00276                     }
00277
00278                     if (clientPtr->sessionIsClosed == FALSE)
00279                     {
00280                         AppCbrClientScheduleNextPkt(node, clientPtr);
00281                     }
```

定期送信パケットのスケジューリング

行番号	処理内容
273-276	パケットの残り送信回数をデクリメントする。
278-281	セッションが終了していなければ、次回の定期送信をスケジューリングする。

1.5.2 AppLayerCbrServer()

app_cbr.cpp

```
00918 {
00919     char error[MAX_STRING_LENGTH];
00920     AppDataCbrServer *serverPtr;
00921
00922     switch (msg->eventType)
00923     {
00924         case MSG_APP_FromTransport:
00925         {
00926             UdpToAppRecv *info;
00927             CbrData data;
00928
00929             info = (UdpToAppRecv *) MESSAGE_ReturnInfo(msg);
00930             memcpy(&data, MESSAGE_ReturnPacket(msg), sizeof(data));
00931
00932             // trace recd pkt
00933             ActionData acnData;
00934             acnData.actionType = RECV;
00935             acnData.actionComment = NO_COMMENT;
00936             TRACE_PrintTrace(node,
00937                              msg,
00938                              TRACE_APPLICATION_LAYER,
00939                              PACKET_IN,
00940                              &acnData);
00941
```

CBR Server のイベント種別による分岐

行番号	処理内容
922-924	受け取ったメッセージの eventType がトランスポート層からのパケット受信の場合のみ、処理を行う。 それ以外の場合は、エラーを出力して処理を終了する。⇒(1067-1076)
926-930	info 部の型はメッセージ作成時に指定され、UdpToAppRecv 型である。 パケット部の型もメッセージ作成時に指定されており、CbrData 型である。 MESSAGE_ReturnInfo 関数を使用して、メッセージから Info 部へのポインタを取り出し info に代入する。 MESSAGE_ReturnPacket 関数を使用して、メッセージからパケット部へのポインタを取り出し、その内容を data へコピーする。
932-940	TRACE_PrintTrace 関数を使用して、パケット受信時の情報を出力する。

app_cbr.cpp

```
00942 #ifdef DEBUG
```

```
00943             {
00944                 char clockStr[MAX_STRING_LENGTH];
00945                 char addrStr[MAX_STRING_LENGTH];
00946                 TIME_PrintClockInSecond(data.txTime, clockStr, node);
00947                 IO_ConvertIpAddressToString(&info->sourceAddr, addrStr);
00948
00949                 printf("CBR Server %ld: packet transmitted at %sS¥n",
00950                        node->nodeId, clockStr);
00951                 TIME_PrintClockInSecond(getSimTime(node), clockStr, node);
00952                 printf("    received at %sS¥n", clockStr);
00953                 printf("    client is %s¥n", addrStr);
00954                 printf("    connection Id is %d¥n", data.sourcePort);
00955                 printf("    seqNo is %d¥n", data.seqNo);
00956             }
00957 #endif /* DEBUG */
```

受信パケットに関するデバッグ出力

行番号	処理内容
942-957	DEBUG が定義されている場合デバッグ情報を出力する。 TIME_PrintClockInSecond 関数を使用して、時刻単位を秒に変換し、文字列に変換する。 IO_ConvertIpAddressToString 関数を使用して、送信側 IP アドレスを文字列に変換する。 出力は、以下の順で行われる。 受信サーバの NodeId、パケット送信時刻(単位秒)、パケット受信時刻(単位秒)、送信クライアントの IP アドレス、送信元ポート番号、パケットのシーケンス番号。

app_cbr.cpp

```
00958
00959             serverPtr = AppCbrServerGetCbrServer(node,
00960                                                  info->sourceAddr,
00961                                                  data.sourcePort);
00962
00963             /* New connection, so create new cbr server to handle client. */
00964             if (serverPtr == NULL)
00965             {
00966                 serverPtr = AppCbrServerNewCbrServer(node,
00967                                                      info->destAddr,
00968                                                      info->sourceAddr,
00969                                                      data.sourcePort);
00970             }
00971
00972             if (serverPtr == NULL)
00973             {
00974                 sprintf(error, "CBR Server: Node %d unable to "
00975                         "allocation server¥n", node->nodeId);
00976
00977                 ERROR_ReportError(error);
00978             }
00979
```

CBR サーバ構造体の作成または取得

行番号	処理内容
959-961	AppCbrServerGetCbrServer 関数(後述)を呼び出し、すでに作成済みの CBR サーバのデータのポインタを取得する。
963-970	取得できない場合は初めてのパケットの到着と判断する。 この場合には、AppCbrServerNewCbrServer 関数(後述)を呼び出し、新規に CBR データ構造を作成し、そのポインタを取得する。

972-979　既存の CBR サーバ構造体が取り出せなかった場合か、新規に CBR サーバ構造体を作成できなかった場合には、エラーメッセージを作成し、ERROR_ReportError 関数を呼び出しエラーとする。ERROR_ReportError 関数は、QualNet を終了させる。

app_cbr.cpp

```
00980 #ifdef ADDON_BOEINGFCS
00981             if ((serverPtr->useSeqNoCheck && data.seqNo >= serverPtr->seqNo)
||
00982                 (serverPtr->useSeqNoCheck == FALSE))
00983 #else
00984             if (data.seqNo >= serverPtr->seqNo)
00985 #endif
00986             {
00987                 serverPtr->numBytesRecvd += MESSAGE_ReturnPacketSize(msg);
00988                 serverPtr->sessionLastReceived = getSimTime(node);
00989
00990                 clocktype delay = getSimTime(node) - data.txTime;
00991
00992                 serverPtr->maxEndToEndDelay = MAX( delay, serverPtr-
>maxEndToEndDelay);
00993                 serverPtr->minEndToEndDelay = MIN( delay, serverPtr-
>minEndToEndDelay);
00994                 serverPtr->totalEndToEndDelay += delay;
00995                 serverPtr->numPktsRecvd++;
00996
00997                 serverPtr->seqNo = data.seqNo + 1;
00998
00999
```

CBR Server での順序制御

行番号	処理内容
984-999	受け取ったパケットのシーケンス番号と、CBR サーバ側で管理しているシーケンス番号を比較し、未受信であれば以下の受信処理を行う。すでに受信していれば何もしない。

- MESSAGE_ReturnPacketSize で受信バイト数を取り出した後に、送受信バイト数に加算する。
- 現在の時刻で、最後にパケットを受信した時刻を更新する。
- 現在の時刻と、パケットの送信時刻の差から遅延時間を計算する。
- 現在の最大遅延時間と比較し、最大遅延時間を更新する。
- 現在の最小遅延時間と比較し、最小遅延時間を更新する。
- これまでの累積遅延時間に加算する。
- 受信パケット数を 1 増やす。
- 次に受信すべきシーケンス番号を受信したシーケンス番号の次(+1)とする。欠落したパケットが発生する可能性があるため、シーケンス番号の連続性をチェックしているわけではないことに注意。984 行目の if 文で、過去のパケット(シーケンス番号が若いもの)は処理されない。

app_cbr.cpp

```
01000 #ifdef STK_INTERFACE
01001                     //totalReceived++;
01002
01003                     StkUpdateCbrStats(
01004                            node,
01005                            STK_STAT_CBR_RECEIVED,
01006                            serverPtr->numPktsRecvd,
01007                            getSimTime(node));
```

```
01008
01009                      //StkUpdateCbrStats(
01010                      //       node,
01011                      //       STK_STAT_CBR_PDR,
01012                      //       (double)totalReceived / (double)totalSent,
01013                      //       getSimTime(node));
01014 #endif /* STK_INTERFACE */
01015
01016            // Calculate jitter.
01017
01018            clocktype transitDelay = getSimTime(node)- data.txTime;
01019            // Jitter can only be measured after receiving
01020            // two packets.
01021            if ( serverPtr->numPktsRecvd > 1)
01022               {
01023                 clocktype tempJitter = transitDelay -
01024                 serverPtr->lastTransitDelay;
01025
01026                if (tempJitter < 0)
01027                {
01028                  tempJitter = -tempJitter;
01029                }
01030                serverPtr->actJitter += ((tempJitter - serverPtr-
>actJitter)/16);
01031                serverPtr->totalJitter += serverPtr->actJitter;
01032              }
01033
01034            serverPtr->lastTransitDelay
01035              = transitDelay;
01036
01037            //          serverPtr->lastPacketReceptionTime =
getSimTime(node);
01038
```

★ジッタの計算

行番号	処理内容
1018-1035	引き続き、受信処理である。ジッタの計算を行う。 CBR のジッタは、以下の定義で計算されている。 現在の遅延(transitDelay)と直前のパケット遅延(serverPtr->lastTransitDelay)の差の絶対値を tempJitter とする。 動的なジッタ(serverPtr->actJitter)は、現在のジッタから動的なジッタの 1/16 を減算したものとしている。ここで使われている Jitter 計算式は「RFC3550 A.8 Estimating the Interarrival Jitter」で定義されいる計算式である。 総合的なジッタ(serverPtr->totalJitter)に動的なジッタを加算する。 現在の遅延を直前のパケット遅延として記憶する。

app_cbr.cpp

```
01039 #ifdef DEBUG
01040                {
01041                    char clockStr[24];
01042                    TIME_PrintClockInSecond(
01043                        serverPtr->totalEndToEndDelay, clockStr);
01044                    printf("    total end-to-end delay so far is %sS¥n",
01045                           clockStr);
01046                }
01047 #endif /* DEBUG */
01048
01049                if (data.type == 'd')
01050                {
01051                    /* Do nothing. */
01052                }
```

```
01053                 else if (data.type == 'c')
01054                 {
01055                     serverPtr->sessionFinish = getSimTime(node);
01056                     serverPtr->sessionIsClosed = TRUE;
01057                 }
01058                 else
01059                 {
01060                     assert(FALSE);
01061                 }
01062             }
01063 
01064             break;
01065         }
```

セッションクローズの判断

行番号	処理内容
1039-1047	DEBUG が定義されている場合デバッグ情報を出力する。 TIME_PrintClockInSecond 関数を使用して、ジッタの値を秒に変換して出力する。
1049-1061	パケットの種別を判別し、'd'の場合はデータパケットを表すため処理しない。'c'の場合は、セッションのクローズを表すため、現在の時刻をセッション終了時刻として記憶し、さらにクローズフラグを TRUE に設定する。 それ以外の種別の場合はデータエラーと見なして assert させる。

app_cbr.cpp

```
01067         default:
01068         {
01069             char buf[MAX_STRING_LENGTH];
01070 
01071             TIME_PrintClockInSecond(getSimTime(node), buf, node);
01072             sprintf(error, "CBR Server: At time %sS, node %d received "
01073                     "message of unknown type "
01074                     "%d¥n", buf, node->nodeId, msg->eventType);
01075             ERROR_ReportError(error);
01076         }
01077     }
01078 
01079     MESSAGE_Free(node, msg);
01080 }
```

★不足

行番号	処理内容
1067-1076	CBR のサーバ側では処理すべきメッセージは MSG_APP_FromTransport のみであるため、それ以外の場合はエラーとする。
1079	メッセージ受信処理は全て処理されたため、MESSAGE_Free 関数を使用して、受信メッセージを解放する。処理済みのメッセージは必ず解放しなければならない。未開放のままだと、メモリ使用量が増加しメモリ不足を引き起こすことがある。

app_cbr.cpp

```
01275 AppDataCbrServer *
01276 AppCbrServerGetCbrServer(
01277     Node *node, Address remoteAddr, short sourcePort)
01278 {
01279     AppInfo *appList = node->appData.appPtr;
01280     AppDataCbrServer *cbrServer;
```

```
01281
01282     for (; appList != NULL; appList = appList->appNext)
01283     {
01284         if (appList->appType == APP_CBR_SERVER)
01285         {
01286             cbrServer = (AppDataCbrServer *) appList->appDetail;
01287
01288             if ((cbrServer->sourcePort == sourcePort) &&
01289                 IO_CheckIsSameAddress(
01290                     cbrServer->remoteAddr, remoteAddr))
01291             {
01292                 return cbrServer;
01293             }
01294        }
01295    }
01296
01297     return NULL;
01298 }
```

★不足

行番号	処理内容
1279-1297	ノードが管理しているすべて全てのアプリケーション情報調べて、引数で指定された送信側 IP アドレスとポート番号を持つものがあるか調べる。 ノードには CBR 以外の他のアプリケーション情報もリスト形式で管理されている。 まず、最初にアプリケーションタイプ(appList->appType)が APP_CBR_SERVER であるかをチェックする。 次に、アプリケーション詳細情報(appList->appDetail)を取り出す。ここに CBR サーバ固有の詳細情報が管理されている。 最後に、取り出した CBR サーバ詳細情報の IP アドレスと、ポート番号を比較し、それぞれが一致した時のみ、CBR サーバ詳細情報へのポインタを返す。 見つからなかった場合は、NULL を返す。

app_cbr.cpp

```
01311 AppDataCbrServer *
01312 AppCbrServerNewCbrServer(Node *node,
01313                          Address localAddr,
01314                          Address remoteAddr,
01315                          short sourcePort)
01316 {
01317     AppDataCbrServer *cbrServer;
01318
01319     cbrServer = (AppDataCbrServer *)
01320                 MEM_malloc(sizeof(AppDataCbrServer));
01321     memset(cbrServer, 0, sizeof(AppDataCbrServer));
01322     /*
01323      * Fill in cbr info.
01324      */
01325     memcpy(&(cbrServer->localAddr), &localAddr, sizeof(Address));
01326     memcpy(&(cbrServer->remoteAddr), &remoteAddr, sizeof(Address));
01327     cbrServer->sourcePort = sourcePort;
01328     cbrServer->sessionStart = getSimTime(node);
01329     cbrServer->sessionFinish = getSimTime(node);
01330     cbrServer->sessionLastReceived = getSimTime(node);
01331     cbrServer->sessionIsClosed = FALSE;
01332     cbrServer->numBytesRecvd = 0;
01333     cbrServer->numPktsRecvd = 0;
01334     cbrServer->totalEndToEndDelay = 0;
01335     cbrServer->maxEndToEndDelay = 0;
01336     cbrServer->minEndToEndDelay = CLOCKTYPE_MAX;
01337     cbrServer->seqNo = 0;
```

```
01338     //    cbrServer->lastInterArrivalInterval = 0;
01339     //    cbrServer->lastPacketReceptionTime = 0;
01340     cbrServer->totalJitter = 0;
01341     cbrServer->actJitter = 0;
01342
01343 #ifdef ADDON_BOEINGFCS
01344     BOOL retVal = FALSE;
01345
01346     cbrServer->useSeqNoCheck = TRUE;
01347
01348     IO_ReadBool(
01349         node->nodeId,
01350         ANY_ADDRESS,
01351         node->partitionData->nodeInput,
01352         "APP-CBR-USE-SEQUENCE-NUMBER-CHECK",
01353         &retVal,
01354         &cbrServer->useSeqNoCheck);
01355
01356 #endif
01357
01358     // Add CBR variables to hierarchy
01359     std::string path;
01360     D_Hierarchy *h = &node->partitionData->dynamicHierarchy;
01361
01362     if (h->CreateApplicationPath(
01363             node,
01364             "cbrServer",
01365             cbrServer->sourcePort,
01366             "numBytesRecvd",
01367             path))
01368     {
01369         h->AddObject(
01370             path,
01371             new D_Int64Obj(&cbrServer->numBytesRecvd));
01372     }
01373
01374     APP_RegisterNewApp(node, APP_CBR_SERVER, cbrServer);
01375
01376     return cbrServer;
01377 }
```

★不足

行番号	処理内容
1317-1321	新しい AppDataCbrServer 構造体を確保する。 MEM_malloc 関数を使用して、AppDataCbrServer 構造体領域のメモリを確保する。確保した領域は 0 で初期化しておく。
1325-1341	CBR サーバ情報の以下のパラメータに初期値を設定する。

- 自ノードのアドレス
- 送信元ノードのアドレス
- 送信元ポート
- セッション開始時刻
- セッション終了時刻
- 最後にパケットを受信した時刻
- セッションが閉じているか判定するフラグ
- 受信したバイト数
- 受信したパケット数
- 遅延の合計
- 最大遅延

- 最小遅延
- シーケンス番号
- ジッタの合計
- ジッタ

1358-1372	**Dynamic Object** である cbrServer->numBytesRecvd を Dynamic Object Hierarchy に登録する。
1374	作成した SBR サーバ詳細情報を APP_RegisterNewApp 関数を使用してノードに登録する。この時、APP 識別として APP_CBR_SERVER を指定する。

1.6 終了処理詳細

終了処理の主な役割は、シミュレーション中に取った統計情報を.stat ファイルに書き出すことである。実際のファイルへの書き出しはパーティション毎に別々のファイルに対して行われ、最後にそれらファイルをマージして一つの.stat ファイルが作りだされる。

CBR の終了処理は application.cpp 内の関数 App_Finalize からクライアントとサーバ、それぞれのインスタンスが別々に呼び出され、実行される。ノードは同じアプリケーションのインスタンスを複数持つことができるため、終了関数も同じノードに対して複数回実行されるという点に注意が必要である。

1.6.1 AppCbrClientFinalize()

CBR クライアント(送信側)の終了処理関数。統計情報ファイル出力処理を行っている。引数は以下のとおり。

1) ノード構造体へのポインタ
2) AppInfo 構造体へのポインタ(メンバの appDetail に CBR サーバのアプリケーション構造体へのポインタが含まれる)

app_cbr.cpp

```
00683 void
00684 AppCbrClientFinalize(Node *node, AppInfo* appInfo)
00685 {
00686     AppDataCbrClient *clientPtr = (AppDataCbrClient*)appInfo->appDetail;
00687
00688     if (node->appData.appStats == TRUE)
00689     {
00690         AppCbrClientPrintStats(node, clientPtr);
00691     }
00692 }
```

CBR クライアントの終了処理

行番号	処理内容
686	AppInfo 構造体ポインタ(引数)から AppDataCbrClient 構造体ポインタを取り出す。
688-691	統計値出力フラグが TRUE の場合、AppCbrClientPrintStats を呼び出して統計値を.stat ファイルに出力する。

1.6.2 AppCbrServerFinalize()

CBR サーバ(受信側)の終了処理関数。統計情報ファイル出力処理を行っている。引数は以下のとおり。

1) ノード構造体へのポインタ

2) AppInfo 構造体へのポインタ(メンバの appDetail に CBR サーバのアプリケーション構造体へのポインタが含まれる)

app_cbr.cpp

```
01256 AppCbrServerFinalize(Node *node, AppInfo* appInfo)
01257 {
01258     AppDataCbrServer *serverPtr = (AppDataCbrServer*)appInfo->appDetail;
01259
01260     if (node->appData.appStats == TRUE)
01261     {
01262         AppCbrServerPrintStats(node, serverPtr);
01263     }
01264 }
```

CBR サーバの終了処理

行番号	処理内容
1258	AppInfo 構造体ポインタ(引数)から AppDataCbrServer 構造体ポインタを取り出す。
1260-1263	統計値出力フラグが TRUE の場合、AppCbrServerPrintStats を呼び出して統計値を.stat ファイルに出力する。 統計値出力フラグ appStats は通常、GUI からレイヤ毎に ON/OFF 設定することができる。デフォルトでは ON(つまり TRUE)になっている。コードの上ではプロトコル毎に ON/OFF できる作りになっているが、.config ファイル中で実際に設定可能なのは、アプリケーションレイヤ全体の統計情報出力を制御するパラメータ APPLICATION-STATISTICS のみである。 この値を YES に設定(デフォルト)すれば、CBR クライアント、CBR サーバどちらも統計情報をファイル出力する仕様となっている。以下に application.cpp の該当箇所のソースを示す。

app_cbr.cpp

```
00525     /* Check if statistics needs to be printed. */
00526
00527     IO_ReadString(
00528         node->nodeId,
00529         ANY_ADDRESS,
00530         nodeInput,
00531         "APPLICATION-STATISTICS",
00532         &retVal,
00533         buf);
00534
00535     if (retVal == FALSE || strcmp(buf, "NO") == 0)
00536     {
00537         node->appData.appStats = FALSE;
00538     }
00539     else
00540     if (strcmp(buf, "YES") == 0)
00541     {
00542         node->appData.appStats = TRUE;
00543     }
00544     else
00545     {
00546         fprintf(stderr,
00547                 "Expecting YES or NO for APPLICATION-STATISTICS
parameter¥n");
00548         abort();
00549     }
```

アプリケーションの統計値情報出力要不要の読み込み

行番号	処理内容
527-533	IO_ReadString 関数を用いて.config ファイルより、当該ノードに対するパラメータ

APPLICATION-STATISTICS の値(YES or NO)を読み出している。パラメータの値はノード単位で指定されている場合もあれば、グローバルな設定、あるいはサブネット単位(アプリケーションの場合には関係無いが)で設定されている場合もある。この API 関数を呼び出すことにより、実際の.config ファイル中での設定をがどのように行われているか気にすることなく、当該ノードに対する設定値だけを取り出すことができる。

535-549　APPLICATION-STATISTICS の設定値が YES か NO かによって、ノード構造体に含まれるアプリケーションデータ構造体(ノード上に設定されたアプリケーションの数だけ存在する)のメンバ appStats の値をそれぞれ TRUE あるいは FALSE に設定する。
YES でも NO でもない値が設定されていたらエラー処理を行う(544 行-549 行)。
繰り返すが、この処理はアプリケーションに依存しない共通の処理として実装されている。もしアプリケーション毎に統計情報出力の ON/OFF 設定を変えたい場合には、この構造体メンバの値がアプリケーションの種類(インスタンス)毎に設定できるようにコードを改変する。

1.7 統計値処理

前述のとおり、QualNet のプロトコルモデルは一般に、終了処理の中で統計値のファイル出力を行う。CBR の例も含め、多くのプロトコルモデルの実装において、統計情報出力処理を別関数として実装し、終了処理関数から呼び出す形にしているものが多い。

1.7.1 AppCbrClientPrintStats

CBR クライアントの統計情報ファイル出力処理関数である。引数は以下のとおり。

1) ノード構造体へのポインタ
2) CBR クライアントのアプリケーションデータ構造体へのポインタ

app_cbr.cpp

```
00558 void AppCbrClientPrintStats(Node *node, AppDataCbrClient *clientPtr) {
00559     clocktype throughput;
00560
00561     char addrStr[MAX_STRING_LENGTH];
00562     char startStr[MAX_STRING_LENGTH];
00563     char closeStr[MAX_STRING_LENGTH];
00564     char sessionStatusStr[MAX_STRING_LENGTH];
00565     char throughputStr[MAX_STRING_LENGTH];
00566
00567     char buf[MAX_STRING_LENGTH];
00568     char buf1[MAX_STRING_LENGTH];
00569
00570     TIME_PrintClockInSecond(clientPtr->sessionStart, startStr, node);
00571     TIME_PrintClockInSecond(clientPtr->sessionLastSent, closeStr, node);
```

時刻を表す文字列の準備

行番号　処理内容

570-571　TIME_PrintClockInSecond は clocktype 型(64 ビット整数でナノ秒単位の時刻を表す)のシミュレーション時間を秒単位の文字列(SS.ssss)形式に変換する関数。CBR セッション開始時刻と終了時刻をそれぞれ文字列に変換する。

app_cbr.cpp

```
00573     if (clientPtr->sessionIsClosed == FALSE) {
00574         clientPtr->sessionFinish = getSimTime(node);
00575         sprintf(sessionStatusStr, "Not closed");
```

```
00576     }
00577     else {
00578         sprintf(sessionStatusStr, "Closed");
00579     }
```

セッション状態を表す文字列の準備

行番号	処理内容
573-574	CBR は CBR クライアントが予め決められた数のデータパケットを送信し終えるとセッションをクローズする。この if 文では、終了処理時点でセッションが既にクローズしているかどうかを判断する。 セッションが終了していなかった場合、セッション終了時刻を現在時刻(シミュレーション終了時刻)に設定する。セッションがクローズしていないことを統計値出力用の文字列変数に設定し、後の処理でファイル出力する。
578	セッションが終了していた場合、セッションがクローズしていることを統計値出力用の文字列変数に設定し、後の処理でファイル出力する。

app_cbr.cpp

```
00580
00581     if (clientPtr->sessionFinish <= clientPtr->sessionStart) {
00582         throughput = 0;
00583     }
00584     else {
00585         throughput = (clocktype)
00586                      ((clientPtr->numBytesSent * 8.0 * SECOND)
00587                       / (clientPtr->sessionFinish
00588                       - clientPtr->sessionStart));
00589     }
00590
00591     ctoa(throughput, throughputStr);
```

スループットを表す文字列の準備

行番号	処理内容
581-583	セッション終了時刻がセッション開始時刻と同じかあるいはそれ以前(実際にはあり得ないだろうが)ということは、セッションが開始されていないことを意味する。その場合、スループットはゼロになる。
584-589	そうでなければ、(CBR クライアント、つまり送信側の)スループットは送信したバイト数*8 を(セッション終了時刻 - セッション開始時刻)で割った値となる。

app_cbr.cpp

```
00593     if(clientPtr->remoteAddr.networkType == NETWORK_ATM)
00594     {
00595         const LogicalSubnet* dstLogicalSubnet =
00596             AtmGetLogicalSubnetFromNodeId(
00597             node,
00598             clientPtr->remoteAddr.interfaceAddr.atm.ESI_pt1,
00599             0);
00600         IO_ConvertIpAddressToString(
00601             dstLogicalSubnet->ipAddress,
00602             addrStr);
00603     }
00604     else
00605     {
00606         IO_ConvertIpAddressToString(&clientPtr->remoteAddr, addrStr);
00607
```

```
00608     }
```

IP アドレスを表す文字列の準備

行番号	処理内容
593-603	ネットワークタイプが ATM の時の処理。ノード ID から ATM の論理サブネット情報を得て、その IP アドレスを文字列に変換する。
604-608	ネットワークタイプが ATM ではない時、宛先 IP アドレスを文字列に変換する。

app_cbr.cpp

```
00610     sprintf(buf, "Server Address = %s", addrStr);
00611     IO_PrintStat(
00612         node,
00613         "Application",
00614         "CBR Client",
00615         ANY_DEST,
00616         clientPtr->sourcePort,
00617         buf);
00618
00619     sprintf(buf, "First Packet Sent at (s) = %s", startStr);
00620     IO_PrintStat(
00621         node,
00622         "Application",
00623         "CBR Client",
00624         ANY_DEST,
00625         clientPtr->sourcePort,
00626         buf);
00627
00628     sprintf(buf, "Last Packet Sent at (s) = %s", closeStr);
00629     IO_PrintStat(
00630         node,
00631         "Application",
00632         "CBR Client",
00633         ANY_DEST,
00634         clientPtr->sourcePort,
00635         buf);
00636
00637     sprintf(buf, "Session Status = %s", sessionStatusStr);
00638     IO_PrintStat(
00639         node,
00640         "Application",
00641         "CBR Client",
00642         ANY_DEST,
00643         clientPtr->sourcePort,
00644         buf);
00645
00646     ctoa((Int64) clientPtr->numBytesSent, buf1);
00647     sprintf(buf, "Total Bytes Sent = %s", buf1);
00648     IO_PrintStat(
00649         node,
00650         "Application",
00651         "CBR Client",
00652         ANY_DEST,
00653         clientPtr->sourcePort,
00654         buf);
00655
00656     sprintf(buf, "Total Packets Sent = %u", clientPtr->numPktsSent);
00657     IO_PrintStat(
00658         node,
00659         "Application",
00660         "CBR Client",
00661         ANY_DEST,
```

```
00662         clientPtr->sourcePort,
00663         buf);
00664
00665     sprintf(buf, "Throughput (bits/s) = %s", throughputStr);
00666     IO_PrintStat(
00667         node,
00668         "Application",
00669         "CBR Client",
00670         ANY_DEST,
00671         clientPtr->sourcePort,
00672         buf);
00673
00674 }
```

CBR クライアントの各種統計情報をファイルへ出力

行番号	処理内容
611-672	IO_PrintStat 関数を繰り返し呼び出して、CBR クライアントの各種統計値をファイル出力する。

1.7.2 AppCbrServerPrintStats

CBR サーバ側の統計情報出力関数である。引数は以下のとおり。

1) ノード構造体へのポインタ
2) CBR サーバのアプリケーションデータ構造体へのポインタ

app_cbr.cpp

```
01100 void
01101 AppCbrServerPrintStats(Node *node, AppDataCbrServer *serverPtr) {
01102     clocktype throughput;
01103     clocktype avgJitter;
01104
01105     char addrStr[MAX_STRING_LENGTH];
01106     char jitterStr[MAX_STRING_LENGTH];
01107     char clockStr[MAX_STRING_LENGTH];
01108     char startStr[MAX_STRING_LENGTH];
01109     char closeStr[MAX_STRING_LENGTH];
01110     char sessionStatusStr[MAX_STRING_LENGTH];
01111     char throughputStr[MAX_STRING_LENGTH];
01112     char buf[MAX_STRING_LENGTH];
01113     char buf1[MAX_STRING_LENGTH];
01114
01115     TIME_PrintClockInSecond(serverPtr->sessionStart, startStr, node);
01116     TIME_PrintClockInSecond(serverPtr->sessionLastReceived, closeStr, node);
```

時刻を表す文字列の準備

行番号	処理内容
1115-1116	TIME_PrintClockInSecond は clocktype 型(64 ビット整数でナノ秒単位の時刻を表す)のシミュレーション時間を秒単位の文字列(SS.ssss)形式に変換する関数。CBR セッション開始時刻と最終パケット受信時刻をそれぞれ文字列に変換する。

app_cbr.cpp

```
01117
01118     if (serverPtr->sessionIsClosed == FALSE) {
01119         serverPtr->sessionFinish = getSimTime(node);
```

```
01120         sprintf(sessionStatusStr, "Not closed");
01121     }
01122     else {
01123         sprintf(sessionStatusStr, "Closed");
01124     }
01125
```

セッション状態を表す文字列の準備

行番号	処理内容
1118-1124	セッションの状態とセッション終了時刻を設定する、CBR クライアントとほぼ同様の処理。CBR クライアントの説明を参照。

app_cbr.cpp

```
01126     if (serverPtr->numPktsRecvd == 0) {
01127         TIME_PrintClockInSecond(0, clockStr);
01128     }
01129     else {
01130         TIME_PrintClockInSecond(
01131             serverPtr->totalEndToEndDelay / serverPtr->numPktsRecvd,
01132             clockStr);
01133     }
```

遅延時間を表す文字列の準備

行番号	処理内容
1126-1133	受信したパケット単位の遅延時間を clockStr に文字列として保存。受信したパケット数がゼロであれば遅延時間もゼロである。

app_cbr.cpp

```
01135     if (serverPtr->sessionFinish <= serverPtr->sessionStart) {
01136         throughput = 0;
01137     }
01138     else {
01139         throughput = (clocktype)
01140                      ((serverPtr->numBytesRecvd * 8.0 * SECOND)
01141                       / (serverPtr->sessionFinish
01142                       - serverPtr->sessionStart));
01143     }
01145     ctoa(throughput, throughputStr);
```

スループットを表す文字列の準備

行番号	処理内容
1135-1145	スループットの計算。セッションが成立していない状況ではスループットはゼロ(1136 行)。そうでなければ、スループットは受信した総バイト数×8 を、セッション継続時間で割った値(1139 行-1142 行)として計算される。

app_cbr.cpp

```
01147     IO_ConvertIpAddressToString(&serverPtr->remoteAddr, addrStr);
01148
01149     sprintf(buf, "Client address = %s", addrStr);
01150     IO_PrintStat(
01151         node,
01152         "Application",
01153         "CBR Server",
01154         ANY_DEST,
01155         serverPtr->sourcePort,
```

```
01156         buf);
01157
01158     sprintf(buf, "First Packet Received at (s) = %s", startStr);
01159     IO_PrintStat(
01160         node,
01161         "Application",
01162         "CBR Server",
01163         ANY_DEST,
01164         serverPtr->sourcePort,
01165         buf);
01166
01167     sprintf(buf, "Last Packet Received at (s) = %s", closeStr);
01168     IO_PrintStat(
01169         node,
01170         "Application",
01171         "CBR Server",
01172         ANY_DEST,
01173         serverPtr->sourcePort,
01174         buf);
01175
01176     sprintf(buf, "Session Status = %s", sessionStatusStr);
01177     IO_PrintStat(
01178         node,
01179         "Application",
01180         "CBR Server",
01181         ANY_DEST,
01182         serverPtr->sourcePort,
01183         buf);
01184
01185
01186     ctoa((Int64) serverPtr->numBytesRecvd, buf1);
01187     sprintf(buf, "Total Bytes Received = %s", buf1);
01188     IO_PrintStat(
01189         node,
01190         "Application",
01191         "CBR Server",
01192         ANY_DEST,
01193         serverPtr->sourcePort,
01194         buf);
01195
01196     sprintf(buf, "Total Packets Received = %u", serverPtr->numPktsRecvd);
01197     IO_PrintStat(
01198         node,
01199         "Application",
01200         "CBR Server",
01201         ANY_DEST,
01202         serverPtr->sourcePort,
01203         buf);
01204
01205     sprintf(buf, "Throughput (bits/s) = %s", throughputStr);
01206     IO_PrintStat(
01207         node,
01208         "Application",
01209         "CBR Server",
01210         ANY_DEST,
01211         serverPtr->sourcePort,
01212         buf);
01213
01214     sprintf(buf, "Average End-to-End Delay (s) = %s", clockStr);
01215     IO_PrintStat(
01216         node,
01217         "Application",
01218         "CBR Server",
01219         ANY_DEST,
01220         serverPtr->sourcePort,
01221         buf);
```

CBR サーバの各種統計情報をファイルへ出力

行番号	処理内容
1147-1221	IO_PrintStat 関数を繰り返し呼び出して、CBR サーバの各種統計値をファイル出力する。

app_cbr.cpp

```
01223
01224     // Jitter can only be measured after receiving two packets.
01225
01226
01227     if (serverPtr->numPktsRecvd - 1 <= 0)
01228       {
01229         avgJitter = 0;
01230       }
01231     else
01232       {
01233         avgJitter = serverPtr->totalJitter / (serverPtr->numPktsRecvd-1) ;
01234       }
01235
01236     TIME_PrintClockInSecond(avgJitter, jitterStr);
01237
01238     sprintf(buf, "Average Jitter (s) = %s", jitterStr);
01239     IO_PrintStat(
01240         node,
01241         "Application",
01242         "CBR Server",
01243         ANY_DEST,
01244         serverPtr->sourcePort,
01245         buf);
01246 }
```

ジッタ情報をファイルへ出力

行番号	処理内容
1127-1234	ジッタの計算を行う。
1236-1238	ジッタの計算結果を秒単位の文字列に変換する。
1239-1245	ファイルへ出力する。

1.8 GUI 設定

GUI での各プロトコルのパラメータ設定画面は、XML 記述形式であり、ユーザがカスタマイズ可能である。CBR の GUI パラメータ設定用のソースコードは以下のファイルである。

QUALNET_HOME\gui\settings\components\cbr.cmp

CBR の GUI パラメータ設定画面を図 9 に示す。

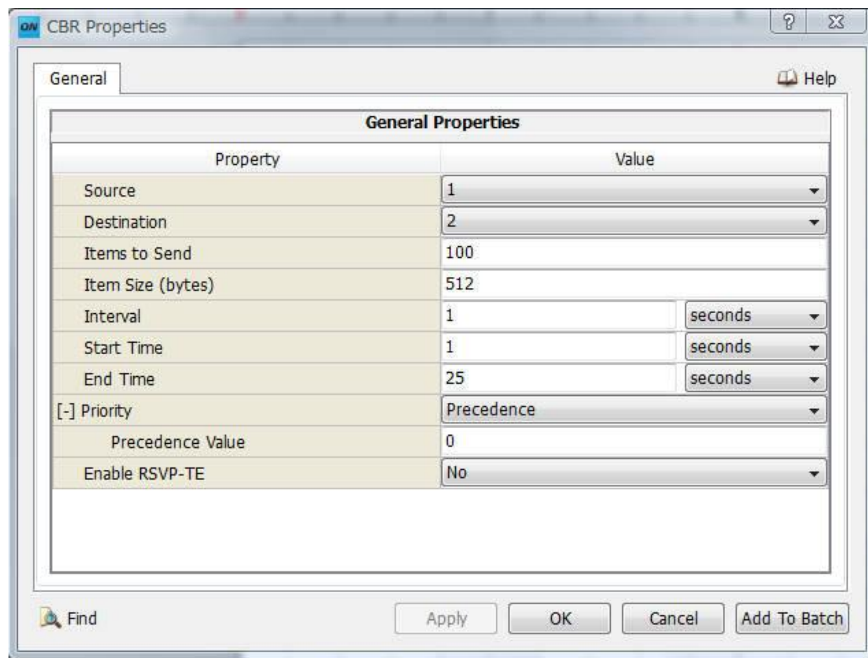

図 9 GUI パラメータ設定画面

cbr.cmp

```
00001 <?xml version="1.0" encoding="ISO-8859-1"?>
00002 <root version="1.0">
00003   <category name="CBR Properties" singlehost="false" loopback="enabled"
      propertytype="CBR">
```

バージョン情報とGUI 設定のカテゴリに関する設定

行番号	処理内容
1-3	category name は GUI 設定画面のタイトルバーに表記される。propertytype はプロトコルを含めた全てのコンポーネントの中でユニークな名称の必要がある。singlehost は single-host アプリケーションか client-server アプリケーションかを規定する。loopback は loop-back を可能とするかを規定する。

cbr.cmp

```
00004 <variable name="Source" key="SOURCE" type="SelectionDynamic"
      keyvisible="false" optional="false" />
00005 <variable name="Destination" key="DESTINATION" type="SelectionDynamic"
      keyvisible="false" optional="false" />
00006     <variable name="Items to Send" key="ITEM-TO-SEND" type="Integer"
      default="100" min="0" keyvisible="false" help="Number of items to send"
      optional="false" />
00007     <variable key="ITEM-SIZE" type="Integer" name="Item Size (bytes)"
      default="512" min="24" max="65023" keyvisible="false" help="Item size in
      bytes" optional="false" />
00008     <variable name="Interval" key="INTERVAL" type="Time" default="1S"
      keyvisible="false" optional="false" />
00009     <variable name="Start Time" key="START-TIME" type="Time" default="1S"
      keyvisible="false" optional="false" />
00010     <variable name="End Time" key="END-TIME" type="Time" default="25S"
      keyvisible="false" optional="false" />
```

各種パラメータ設定

行番号	処理内容
4-10	階層や選択のないパラメータ項目は、それぞれ 1 行に属性を記述する。 name は、GUI 設定画面に表記される。 Key は、シナリオ設定ファイルに記述されるユニークな名称である。 Type は属性値の種類である。SelectionDynamic は既に設定されているノードを示し、選択リストの内容はノード ID か IP アドレスである。 Keyvisible はシナリオ設定ファイルに key を表示するかどうかを示す。 Optional は必須かオプションを示す。 なお、"Items to Send"、End Time を 0 に設定すると、シナリオが終了するまでトラフィックが発生する。

cbr.cmp

```
00011 <variable name="Priority" key="PRIORITY" type="Selection"
      default="PRECEDENCE" keyvisible="false">
00012 <option value="TOS" name="TOS" help="value (0-255) of the TOS bits in the IP
      header">
00013         <variable name="TOS Value" key="TOS-BITS" type="Integer" default="0"
      min="0" max="255" keyvisible="false" optional="false" />
00014       </option>
00015       <option value="DSCP" name="DSCP" help="value (0-63) of the DSCP bits in
      the IP header">
00016         <variable name="DSCP Value" key="DSCP-BITS" type="Integer"
      default="0" min="0" max="63" keyvisible="false" optional="false" />
00017       </option>
00018       <option value="PRECEDENCE" name="Precedence" help="value (0-7) of the
      Precedence bits in the IP header">
00019         <variable name="Precedence Value" key="PRECEDENCE-BITS"
      type="Integer" default="0" min="0" max="7" keyvisible="false"
      optional="false" />
00020       </option>
00021     </variable>
```

各種パラメータ設定

行番号	処理内容
11-21	選択型であり、選択値によって更に値の設定が必要な場合は、<option>により各選択値を記述する。ここでは 11 行目で type を”Selection”として、12-14 行目、15-17 行目、18-20 行目にてそれぞれの選択項目と詳細な設定値を定義する。

cbr.cmp

```
00022     <variable name="Enable RSVP-TE" key="ENABLE-RSVP-TE" type="Selection"
      default="N/A" keyvisible="false" optional="true">
00023       <option value="N/A" name="No" />
00024       <option value="RSVP-TE" name="Yes" />
00025     </variable>
00026 </category>
00027 </root>
```

各種パラメータ設定

行番号	処理内容
22-27	22 行目で Enable RSVP-TE の型を設定する。選択型を設定しているので、選択値を 23-24 行目で指定する。

GUI のパラメータ設定画面で保存した値は、シナリオフォルダ内の.app ファイルに保存される。
CBR アプリケーションは以下のような 1 行で設定される。

```
CBR 1 3 0 512 0.05S 5S 315S PRECEDENCE 0
```