

# AKARI ネットワークアーキテクチャにおける L5-API の提案

遠藤 誠<sup>†</sup> 金丸 翔<sup>†</sup> 寺岡 文男<sup>†</sup>

<sup>†</sup> 慶應義塾大学大学院理工学研究科

〒 223-8522 横浜市港北区日吉 3-14-1

E-mail: †{done,satsuma}@tera.ics.keio.ac.jp, ††tera@ics.keio.ac.jp

あらまし インターネットを白紙から作り直そうという考え方、いわゆる clean slate approach が日米欧で盛んになっている。その中で、NICT が進めている AKARI プロジェクトでは 6 階層のネットワークアーキテクチャを提案しており、アプリケーション層とトランスポート層の間に第 5 層を導入している。第 5 層では従来のトランスポート層が提供するパスよりも抽象的なパス (L5-path) をアプリケーション層に提供する。L5-path では、複数の L4-path を束ねて L5-path を構成することによって帯域集約や耐故障性を実現したり、空間的または時間的に分断された L4-path をつなぎ合わせて L5-path を構成することによって、ポリシーを適用した L5-path や DTN を実現する。本論文では L5-path を利用するための API を提案する。

キーワード セッション層, API

## The proposal of L5-API for AKARI Network Architecture

Makoto ENDO<sup>†</sup>, Sho KANEMARU<sup>†</sup>, and Fumio TERAOKA<sup>†</sup>

<sup>†</sup> Graduate School of Science and Technology, Keio University

Hiyoshi 3-14-1, Kohoku-ku, Yokohama-shi, 223-8522 Japan

E-mail: †{done,satsuma}@tera.ics.keio.ac.jp, ††tera@ics.keio.ac.jp

**Abstract** The ideas that redesigning the Internet from scratch, so-called "clean slate approach" has become popular in the world. Among them, AKARI project progressed by NICT proposes 6-layered network architecture, in which the 5th layer is introduced between application layer and transport layer. The 5th layer will provide more abstract communication path (L5-path) than traditional transport layer for application layer. In the L5-path, bundling multiple L4-paths and constructing the L5-path will realize the bandwidth aggregation and fault tolerant, or splicing the spatially- or temporally- split L4-paths and constructing the L5-path will realize the policy-applied L5-path or DTN. This paper proposes the APIs to utilize the L5-paths.

**Key words** Session layer, API

### 1. はじめに

近年、インターネットのアーキテクチャを白紙から構築し直すべきだという考えが広まりつつある。このような考え方は "clean slate approach" と呼ばれている。clean slate approach では非階層型や階層型のネットワークアーキテクチャが提案されている。非階層型ネットワークアーキテクチャとしては、NewArch Project [1] で提案されている RBA [2] や FIND [3] で提案されている SILO [4] がある。一方、階層型ネットワークアーキテクチャとしては、RNA [5] や CCN [6] がある。階層構造は、ネットワークを構成する機能を階層に分け、各階層は下位層の構造を隠蔽し、上位層には抽象化されたサービスを提供するという考え方である。階層化により機能がモジュール化で

き、各階層間の独立性が高くなるなどの利点がある。検討の結果、AKARI [7] では今後追加される機能や今までの機能を整理するために、ZNA (*Z Network Architecture*) と呼ばれる 6 階層の階層型ネットワークアーキテクチャを提案している。ZNA の特徴として、第 5 層 (L5: Session Layer) を新たに導入している点が挙げられる。L5 の導入によって、より抽象度の高い通信路 (L5-path) をアプリケーション層に提供することができる。本論文ではアプリケーションが L5-path を利用するための L5-API を提案する。

### 2. L5-path の導入

ZNA は、現在のインターネットには存在しない L5 を導入している [8]。なお、各階層の名称は未定だが、便宜上 OSI 参照

L6: Application	Data Plane	Abstract Entity	Inter-Layer System	Control Plane
L5: Session	Data Plane	Abstract Entity		Control Plane
L4: Transport	Data Plane	Abstract Entity		Control Plane
L3: Network	Data Plane	Abstract Entity		Control Plane
L2: Link	Data Plane	Abstract Entity		Control Plane
L1: Physical	Data Plane	Abstract Entity		Control Plane

図 1 提案アーキテクチャの階層構造

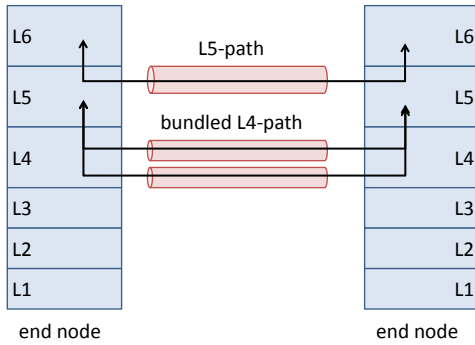


図 2 Bundled path

モデルにおける階層名を利用することとする (図 1) . L5 の導入目的は、現在のインターネットのトランスポート層が提供する通信路 (L4-path) よりさらに抽象的な通信路をアプリケーション層に提供することである . L4-path を複数束ねた、あるいは空間的、時間的に結合した、より抽象的な通信路を L5-path として導入する . L5-path としては以下のようなものがあげられる .

### 2.1 bundled path

複数の L4-path を束ねて L5-path を構成する (図 2 参照) . 束ねた L4-path を同時に使用することにより、帯域を集約した L5-path を提供できる . あるいは、複数の L4-path のうち 1 つを通信に使用し、他を予備としておく . 通信に使用している L4-path に障害が発生したら、予備の L4-path を通信に使用する . これにより耐故障性をもつ L5-path を提供できる . bundled path は SCTP (Stream Control Transport Protocol) [9] のマルチパス機能をモデル化したものである .

### 2.2 spatially-spliced path

空間的に複数の L4-path を中間ノードの L5 でつなぎ合わせて L5-path を構成する (図 3 参照) . L4-path をつなぎ合わせるノードを splicer と呼ぶこととする . splicer はパケットを中継する際、設定されたポリシーに従ってパケットの中継の可否を判断したり、必要によってはヘッダの内容を書き換える . これにより、ポリシーを適用した L5-path を提供できる . spatially-spliced path は NAT (Network Address Translation) のような middle box を介した通信をモデル化したものである .

### 2.3 temporally-spliced path

時間的に複数の L4-path を中間ノードの L5 でつなぎ合わせて L5-path を構成する (図 4 参照) . 図 4(a) では、左端のエンドノードが中央の splicer の通信可能範囲にあり、両ノード間に L4-path が確立されて通信し、splicer は通信内容をバッファ

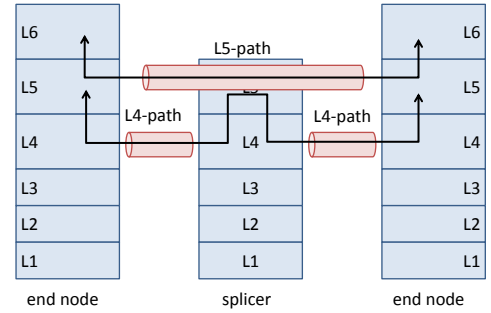


図 3 Spatially-spliced path

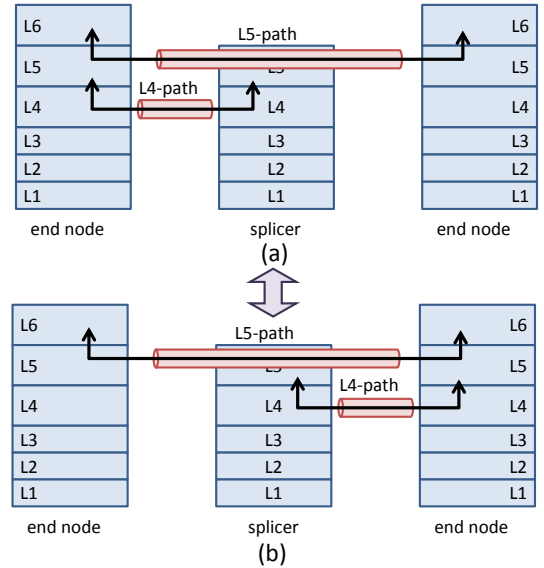


図 4 Temporally-spliced path

リングする . 図 4(b) では splicer は右端のエンドノードとの通信可能範囲に入り、右端のエンドノードとの間に L4-path を確立し、バッファリングしていたデータを送信する . このように、時間的には 2 つの L4-path は同時には確立されていないが、L5-path は常に利用可能のように見える . 以上のように、時間的に L4-path をつなぎ合わせるにより DTN (Delay Tolerant Network) を実現できる .

## 3. L5 socket の定義

現在のインターネットではトランスポート層がアプリケーション層に提供する通信機能の端点を socket と呼び、IP アドレスとポート番号の対で表される . これにない、ZNA ではアプリケーション層が L5-path を利用するための端点を L5-socket と呼ぶこととする . ZNA では ID/Locator split を採用しているので、L5-socket はノード ID とポート番号の対で表されることになる .

現在の socket API では struct sockaddr\_in という構造体で定義されている . これにない、L5-socket として図 5 に示す struct sockaddr\_z という構造体を定義した . sin\_family としては ZNA を表す PF\_ZNET を用いる . ZNA では 1 つのノード ID は複数の Locator を持つことができるが、sockaddr\_z には複数ある Locator のうち、代表的な 1 つのみを登録する

```

struct sockaddr_z {
    uint16_t  sin_family; /* ソケットの protocol ファミリー */
    uint16_t  sin_port;   /* ソケットのポート番号 */
    Z_ID      sin_id;     /* ノード ID */
    Z_LOC     sin_loc;    /* ノードの locator */
    void      *cert;      /* 証明書など */
};

typedef struct z_id {
    uint8_t   registry[2];
    uint8_t   organization[4];
    uint8_t   value[10];
} Z_ID;

typedef struct z_loc {
    uint32_t  loc_family;
    uint8_t   loc[16];
} Z_LOC;

```

図 5 ソケット構造体

こととする。ノード ID を表す構造体としては `struct z_id` (`Z_ID`) を定義した。ZNA では複数のネットワーク層プロトコルの混在をサポートするので、Locator を表す構造体として `struct z_loc` (`Z_LOC`) を定義した。たとえば、Locator として IPv4 address を使用する場合は、`loc_family` には `AF_INET` を指定する。`void *cert` は、spatially-spliced path で使用される (4.2 節参照)。

## 4. API の設計

2. 章で説明した L5-path を実現するために、各パスを確立するためのシグナリングプロトコル、アプリケーションが L5-path を利用するための API が必要となる。そこで、従来のソケット API を基本として、bundled path と spatially-spliced path、temporally-spliced path を利用するための拡張 API (L5-API) を設計する。

L5-API として表 1 のような API を定義した。API は、L5-path 管理、L4-path 管理、送受信という 3 つのグループに分類される。

### 4.1 bundled path シーケンス

図 6 に bundled path で帯域集約する場合のシーケンス図を示す。便宜上、L5-path を能動的に確立するノードを initiator と呼び、受動的に確立するノードを responder と呼ぶこととする。はじめに、両エンドノードは L5-socket を

```
socket_z(PF_ZNET, SOCK_RELIABLE, 0);
```

によって開く。帯域集約の L5-path を確立するために、プロトコルファミリには L5-socket であることを表す `PF_ZNET`、ソケットタイプには信頼性を保証した通信を表す `SOCK_RELIABLE` を指定する。ソケットタイプとしては、この他に UDP のようなデータグラム通信を表す `SOCK_DGRAM` などがある。initiator 側では

```
connect_z(s, r_sock, r_socklen);
```

によって L5-path 接続要求メッセージを送信する。`s` は `socket_z` で開いた L5-socket のソケット記述子、`r_sock`、`r_socklen` はそれぞれ相手ノード (remote) の L5-socket の構造体とそのサイズを表す。responder 側では

```
bind_z(s, r_sock, r_socklen);
listen_z(s);
accept_z(s, l_sock, l_socklen);
```

によってその要求メッセージを受け取り応答メッセージを返す。`s` は `socket_z` で開いた L5-socket のソケット記述子、`r_sock`、`r_socklen`、`l_sock`、`l_socklen` はそれぞれ相手ノード (remote) の L5-socket の構造体とそのサイズ、自ノード (local) の L5-socket の構造体とそのサイズを表す。このメッセージのやりとりによって、両エンドノード間に L5-path を確立することを合意する。その結果、L4-path を 1 つ持った L5-path を確立する。複数の L4-path が同じリンクやノードを通るとスループットの向上が見込めないため、disjoint なパスを見つける必要がある。そこで、両エンドノード間に存在する disjoint な複数の L4-path を見つけるために、

```
getsockopt_z(s, SOL_L5, SO_ISDISJOINT, l_r_path,
             l_r_pathlen);
```

を呼ぶ。ソケットレベルには `SOL_L5`、オプションの名前には disjoint なパスの情報を取得することを指定するために `SO_ISDISJOINT` をそれぞれ指定し、`l_r_path` には両エンドノード間の disjoint な L4-path のソケット構造体の組をリストで取得し、`l_r_pathlen` にはそのサイズが設定される。なお、実際に disjoint なパスを見つける方法については本論文の範囲外とする。両エンドノードのアプリケーションは、

```
send_z(s, msg, msg_len, 0);
recv_z(s, msg, msg_len, 0);
```

を使用して互いに利用できる L4-path を通知し合ったり、利用する L4-path の数などを決定するためにネゴシエーションする。`msg`、`msg_len` は送受信するデータやそのサイズである。そして、

```
connect_L4(s, l_r_path, l_r_pathlen);
listen_L4(s, l_r_path, l_r_pathlen);
close_L4(s, l_r_path, l_r_pathlen);
```

によって L4-path を張り替えることで、disjoint な複数の L4-path を持った L5-path を確立する。`l_r_path`、`l_r_pathlen` には操作したい L4-path のソケット構造体の組を指定する。L5-path を閉じる場合、

```
close_z(s);
```

によって終了メッセージをやりとりし、L5-path を終了させることを両エンドノードが合意するとパスを閉じる。

図 7 に bundled path で耐故障性を向上させる場合のシーケンス図を示す。帯域集約の場合と同様にして、L4-path を 1 つ持った L5-path を確立し、両エンドノード間の disjoint な複数の L4-path を発見する。次に、アプリケーションがネゴシエーションすることで、disjoint な L4-path の中から通信に使用する primary path や予備とする secondary path を決定す

表 1 L5-API の一覧

分類	プロトタイプ	機能
L5-path 管理	<code>int socket_z(int domain, int type, int proto);</code>	L5-path のソケットを開く
	<code>int bind_z(int s, const struct sockaddr_z *addr, socklen_t addrlen);</code>	L5-path のソケットをバインド
	<code>int listen_z(int s);</code>	L5-path 接続キューの作成
	<code>int accept_z(int s, struct sockaddr_z *restrict addr, socklen_t * restrict addrlen);</code>	L5-path 接続の受付
	<code>int connect_z(int s, const struct sockaddr_z *name, socklen_t namelen);</code>	サーバへの L5-path 接続
	<code>int close_z(int s);</code>	L5-path を閉じる
	<code>int cat_conn(int s, struct path *local_remote, void *cert);</code>	L5-path を結合
	<code>int getsockopt_z(int s, int layer, int opt_name, void *opt_value, int *opt_len);</code>	ソケットのオプションを取得
L4-path 管理	<code>int setsockopt_z(int s, int layer, int opt_name, void *opt_value, const int opt_len);</code>	ソケットにオプションを設定
	<code>int connect_L4(int s, struct path *local_remote, int pathlen);</code>	L5-path に L4-path を追加して接続
	<code>int listen_L4(int s, struct path *local_remote, int pathlen);</code>	L5-path に L4-path の追加を受付
送受信	<code>int close_L4(int s, struct path *local_remote, int pathlen);</code>	L4-path を閉じる
	<code>ssize_t send_z(int s, const void *msg, size_t len, int flag);</code>	L5-path にパケットを送信
	<code>ssize_t recv_z(int s, void *buf, size_t len, int flags);</code>	L5-path からパケットを受信
	<code>ssize_t sendmsg_z(int s, const struct msghdr *msg, int flags);</code>	L5-path にパケットを送信 (DTN)
	<code>ssize_t recvmsg_z(int s, struct msghdr *msg, int flags);</code>	L5-path からパケットを受信 (DTN)

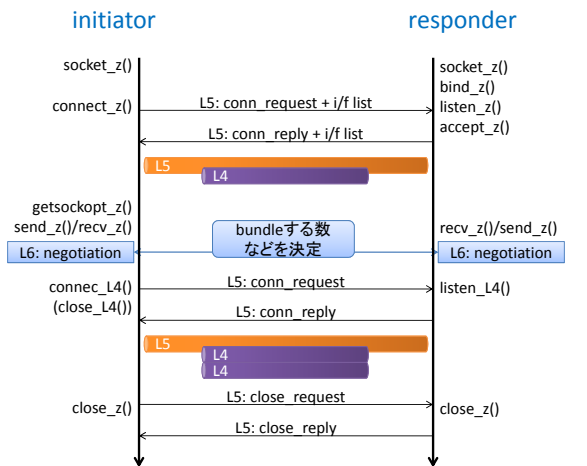


図 6 Bundled path (bandwidth aggregation) のシーケンス図

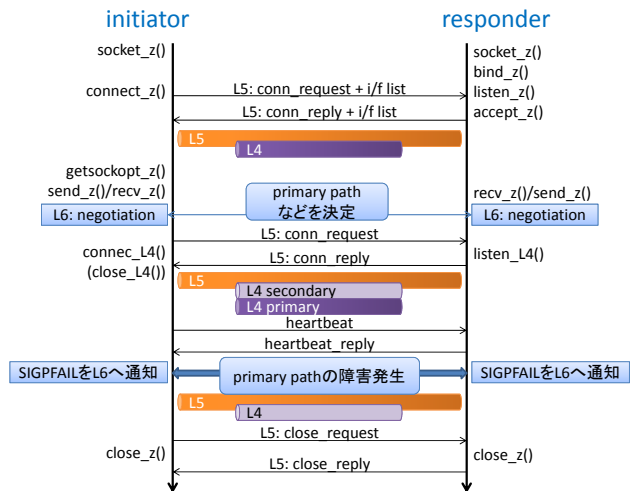


図 7 Bundled path (fault tolerant) のシーケンス図

る。そして、`connect_L4()` などによって L4-path の追加や張り替えを行うと、耐故障性を持った L5-path を確立することができる。各 L4-path 上では heartbeat メッセージをやりとりし、生存を確認する。primary path に障害が発生し通信ができなくなると、L5 は primary path の障害を検知し、待機していた secondary path への切り替えを行う。同時に、L5 はアプリケーションにシグナルを通知することでアプリケーションへパスが切り替えられたことを知らせる。切り替えが行われた結果として、通信を途切れさせることなく維持することができる。L5-path を閉じる場合、`close_z()` によって終了メッセージをやりとりしてパスを閉じる。

#### 4.2 spatially-spliced path シーケンス

前述したように、spatially-spliced path は NAT のような middle box をモデル化したものである。middle box が現在の NAT のようにエンドノードにその存在を気付かせないようなもの場合は、`connect_z()`、`listen_z()`、`accept_z()` によって L5-path が確立される。

middle box がエンドノードとのネゴシエーションを必要と

する場合のシーケンス図を図 8 に示す。はじめに、initiator は responder に接続要求メッセージを送る。しかし、経路の間に存在する splicer によってメッセージは捕捉され、splicer は接続要求に対してエラーメッセージを両エンドノードに送信する。エラーメッセージは splicer が自身の存在を両エンドノードに知らせるために送信される。すると両エンドノードは splicer と接続要求メッセージをやりとりする。これは予め splicer との間に両エンドノードが L5-path を確立するためである。次に両エンドノードと splicer は `send_z()` や `recv_z()` を使用してネゴシエーションする。ネゴシエーションの結果、splicer が両エンドノードのデータを通過させると判断した場合、splicer から両エンドノードに証明書が発行されると考える。そして両エンドノードは受け取った証明書をい用い、

```
cat_conn(s, l_sock, r_sock, cert);
```

によって initiator と splicer の間と responder と splicer の間にある 2 つの L5-path を結合する要求メッセージを両エンド

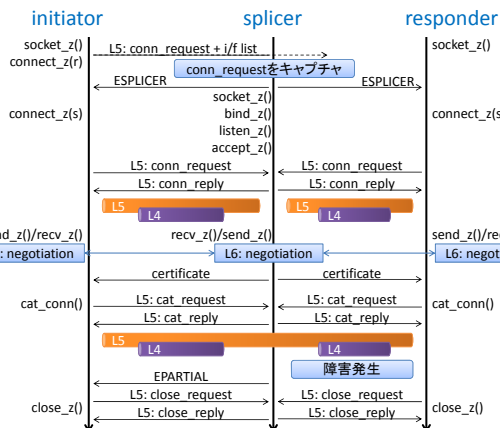


図 8 Spatially-spliced path のシーケンス図

ノード側からそれぞれ送信し, splicer が応答メッセージを送信する.  $s$  はソケット記述子,  $l\_sock$ ,  $s\_sock$  はそれぞれのソケット構造体,  $cert$  は splicer から発行された証明書をそれぞれ表す. 以上のメッセージのやりとりによって 2 つの L5-path を結合することが合意されることで, 両エンドノード間に 1 つの L5-path が確立される. もし片方のエンドノードと splicer の間にある L4-path に障害が発生した場合には, splicer から他方のエンドノードにエラーメッセージを送信することで他方のエンドノードに L5-path が途切れてしまったことを知らせる. L5-path を閉じる場合, 両エンドノード間でアプリケーションが L5-path を閉じることに合意し, 両エンドノードが  $close\_z()$  によって splicer と終了メッセージをやりとりしてパスを閉じる.

### 4.3 temporally-spliced path シーケンス

図 9 に temporally-spliced path を確立する場合のシーケンス図を示す. はじめに, initiator は responder への到達性がない状態で, responder に接続要求メッセージを送る. しかし responder までメッセージは到達せず, 接続要求メッセージは到達可能な splicer までしか進まない. すると splicer は responder が現在到達不可能であることをエラーメッセージによって initiator に通知する. そこで, initiator は接続要求メッセージを splicer に送信し, splicer と L5-path を確立する. splicer から initiator ヘデータのバッファが可能であることを知らされると, initiator は  $sendmsg\_z()$  によって responder 宛てのデータを送信し, splicer は  $sendmsg\_z()$  によって受信しバッファリングする. 次に, splicer が responder へ向かうより最適な splicer と通信可能になると, 同様に 2 つの splicer の間で L5-path を確立し, より最適な splicer ヘデータを渡しバッファリングする. responder へ近づくような splicer と通信可能になると, データを次々と中継していく. データをバッファリングしている splicer が responder と通信可能になると, その splicer と responder の間で L5-path を確立し, responder ヘデータを渡す. 結果として, initiator から responder ヘデータを送信することができる.

### 4.4 L5-path 管理用構造体の定義

図 10 に, L5-path の情報を管理するためにエンドノード

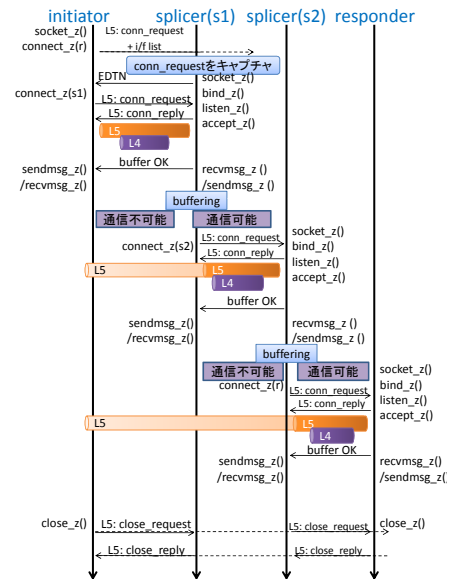


図 9 Temporally-spliced path のシーケンス図

```

struct L5_path { /* end node が保持 */
    int    L5_sock_desc; /* L5 の skt desc */
    uint16_t L5_local_port; /* local の L5 のポート番号 */
    uint16_t L5_remote_port; /* remote の L5 のポート番号 */
    int    L4_sock_desc[8]; /* L4 の skt_desc */
    int    L4_p_sock_desc_ix; /* primary L4 の index */
    int    path_type; /* path のタイプ */
    int    L4_bundle_ix[4]; /* bundled L4 パスの index */
    int    no_s_L4_sock; /* secondary L4 の数 */
    int    stat; /* ステータス */
    void   *cert; /* 証明書 */
    void   *snd_buf; /* bundle 用の送信バッファ */
    void   *rcv_buf; /* bundle 用の受信バッファ */
};

struct spliced_path { /* splicer が保持 */
    int    path_id; /* パスを識別するため */
    int    L4_sock_desc1; /* L4 の socket descriptor */
    int    L4_sock_desc2; /* L4 の socket descriptor */
    int    stat; /* ステータス */
    void   *cert1; /* 証明書 */
    void   *cert2; /* 証明書 */
};

struct L4_path { /* L5-path 中の L4-path の情報 */
    struct sockaddr_z local_sock;
    struct sockaddr_z remote_sock;
};

```

図 10 パス管理構造体

や splicer が保持する構造体を定義した. L5-path 構造体はエンドノードが L5-path の情報を管理するために保持する構造体である.  $L5\_sock\_desc$  は L5-socket のソケット記述子である. L5-path を識別する ID の役割も持つ.  $L4\_sock\_desc[8]$  は, L5-path がもつ複数の L4-path を示す配列である.  $L4\_p\_sock\_desc\_ix$  は,  $L4\_sock\_desc[8]$  の中で primary path である L4-path のインデックスである.  $L4\_bundle\_ix[4]$  は,  $L4\_sock\_desc[8]$  の中で bundled path として利用されている L4-path のインデックスである.  $spliced\_path$  構造体は splicer が L5-path の情報を管理するために保持する構造体である. また L4-path 構造体は 1 つの L4-path の両端にある L5-socket のソケット構造体を持つ. L4-path を指定するために利用する.

type (1byte)	code (1byte)
src_port (2byte)	
dst_port (2byte)	
...	
(type dependent)	

図 11 L5 パケットフォーマット

## 5. L5 プロトコル

L5 プロトコルで使用されるメッセージのパケットフォーマット (L5 パケットフォーマット) を図 11 に示す。図 11 において、type は 1byte のパケットタイプ (機能コード)、code は 1byte のコード (補助的な機能コード)、src\_port は 2byte の始点ポート番号、dst\_port は 2byte の終点ポート番号をそれぞれ表し、残りはパケットタイプごとにそれぞれ異なる (type dependent)。以下に L5 プロトコルで使用されるメッセージの種類を示す。

- CONN Request/Reply  
L5-path を確立するための接続要求および応答メッセージ。ノードの持つインターフェース情報を含む。
- CLOSE Request/Reply  
L5-path を閉じるための終了要求メッセージおよび応答メッセージ。
- HEARTBEAT Request/Reply  
bundled path における primary path の生存確認および応答メッセージ。
- CAT Request/Reply  
spatially-spliced path における 2 つの L5-path を 1 つのパスに結合するための結合要求および応答メッセージ。2 つの L5-path を指定するために、ポート番号の情報をもう 1 組含む。
- STATUS REPORT  
パスやノードの状態を他のノードへ通知するためのレポートメッセージ。
- ERROR REPORT  
パスやノードに発生したエラーの情報を他のノードへ通知するためのレポートメッセージ。

## 6. 実装

ZNA の L5 が実現可能であるかどうか確認するために、第 1 ステップとして現在のスタック上に L5 を実装する予定である。libL5 と呼ばれるライブラリとして L5-API を実装し、アプリケーションプログラムにリンクする。libL5 ライブラリの中で現在のソケット API を呼び出す。実装のモジュール図を図 12 に示す。図 12 において、application が libL5 ライブラリの中の L5-API を利用する。libL5 ライブラリの L5-API を呼び出すと、L4 のシステムコールが呼び出される。

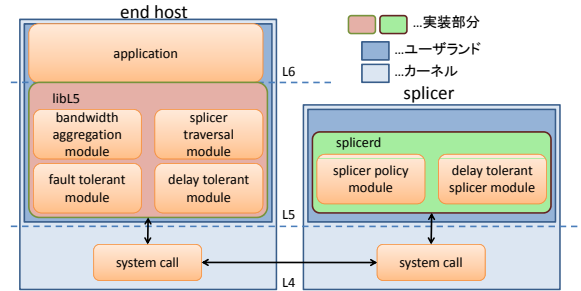


図 12 実装モジュール図

## 7. まとめ

本論文では、AKARI ネットワークアーキテクチャにおいて導入されている L5 によって提供される L5-path を利用するための API を設計した。L5 では帯域集約や耐故障性の向上を実現する bundled path や、ポリシーを適用したパスの提供や DTN を実現する spatially-spliced path, temporally-spliced path の機能を提供する。各 L5-path を利用するために L5 プロトコルを設計し、パスを確立し終了するまでのシーケンス図によって利用方法を示した。また、L5-path を扱うためのソケット構造体やノードが L5-path の情報を管理するためのパス管理構造体を設計した。今後は、L5 が実現可能かどうか確認するために現在のスタックに L5 を実装する。ライブラリとして L5 を実装し、ライブラリの中でソケット API を呼び出して現在のソケットインターフェースを使った実装を考えている。

## 文 献

- [1] “NewArch project home page”. <http://www.isi.edu/newarch/>.
- [2] R. Braden, T. Faber, and M. Handley, “From Protocol Stack to Protocol Heap – Role Based Architecture,” ACM SIGCOMM Computer Communications Review, vol.33, no.1, pp.17–22, Jan. 2003.
- [3] “FIND (future internet design) home page”. <http://www.netsfind.net/>.
- [4] R. Dutta, G.N. Rouskas, I. Baldine, A. Bragg, and D. Stevenson, “The SILO Architecture for Services Integration, control, and Optimization for the Future Internet,” Proceedings of IEEE International Conference on Communications 2007 (ICC 2007), pp.1899–1904, June 2007.
- [5] J.D. Touch, W.Y. S., and V. Pingali, “A Recursive Network Architecture,” 2006. ISI-TR-2006-626.
- [6] V. Jacobson, D.K. Smetters, J.D. Thornton, M.F. Plass, N.H. Briggs, and R.L. Braynard, “Networking Named Content,” Proceedings of CoNEXT’09, pp.1–12, Dec. 2009.
- [7] “AKARI Project Home Page”. <http://akari-project.nict.go.jp/>.
- [8] F. Teraoka, “Redesigning Layered Network Architecture for New Generation Networks,” Proceedings of 2nd IEEE Workshop on the Network of the Future, Dec. 2009.
- [9] R. Stewart, Q. Xie, K. Morneault, C. Sharp, H. Schwarzbauer, T. Taylor, I. Rytina, M. Kalla, L. Zhang, and V. Paxson, “Stream control transmission protocol,” RFC2960, Oct. 2000.