Native SHIM APIs

Miika Komu <miika@iki.fi>
Helsinki Institute for Information Technology

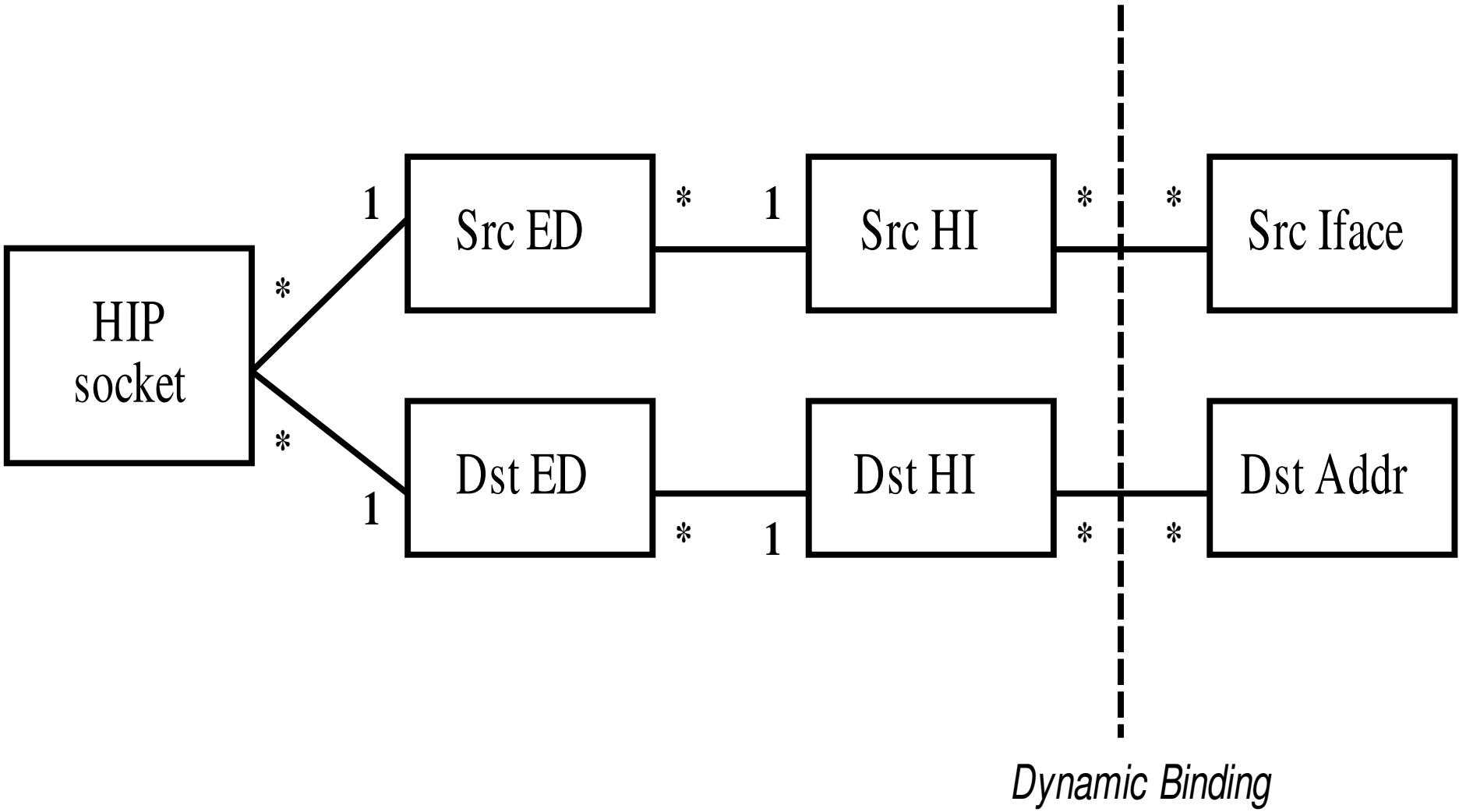
About the Draft

- The draft has been earlier on HIP RG
- Differences to the earlier draft
  - Generalized the text (could be used also by BTNS, and maybe SHIM6?)
  - Used a separate resolver function call earlier, now uses the standard getaddrinfo
- Contributions
  - A new indirection/abstraction called endpoint descriptor
  - Applications gain more control of SHIM Layer
  - Application can specify its own public keys

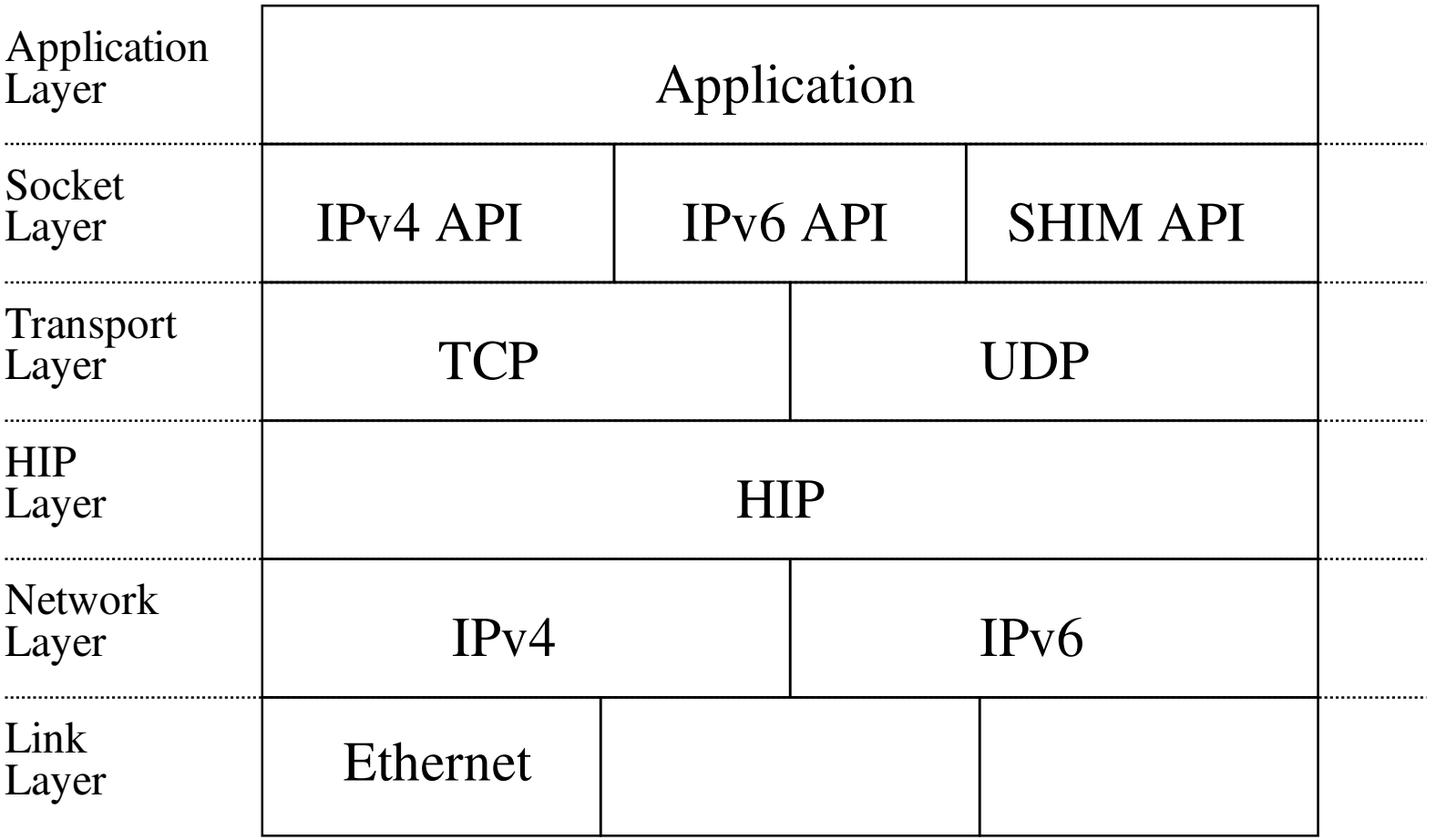
End-point Descriptors

- ED = End-point Descriptor
  - 32 bit value
  - Acts as a token or handle to a HI
- Why not HI or HIT?
  - ED hides the presentation of HIs from most apps
  - Deployment of new identifiers/layers (service/session?) is easier because applications do not have to be modified
  - Implementing opportunistic mode and application specified identifiers (variable sized public keys) becomes easier

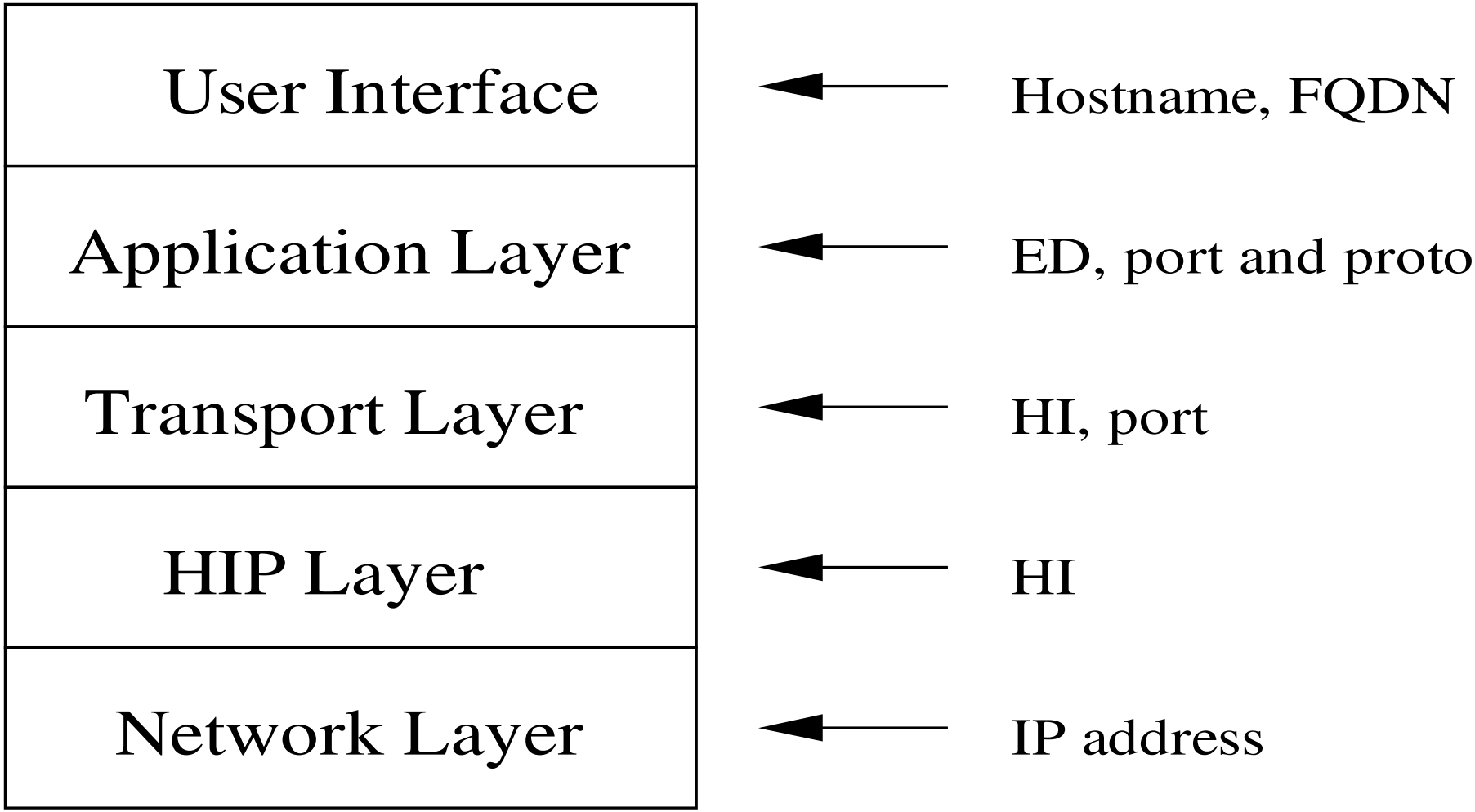
ED Bindings


HIP
socket
*
*
1
Src ED
*
1
Src HI
*
*
Src Iface
1
Dst ED
*
1
Dst HI
*
*
Dst Addr
Dynamic Binding

API Layering



Naming


User Interface
Application Layer
Transport Layer
HIP Layer
Network Layer
Hostname, FQDN
ED, port and proto
HI, port
HI
IP address

Legacy APIs

- Suitable for non-SHIM aware applications
  - Applications need no changes
  - Easier deployment path
- Applications use HITs or LSIs to establish connections using HIP
  - No new socket handler required
  - The use of IP addresses is also possible
- http://www.ietf.org/internet-drafts/draft-henderson-hip-applications-03.txt

Native SHIM API

- Suitable for new SHIM aware applications
  - Applications can control the SHIM layer better
- Introduces a new socket family: PF_SHIM
  - Detection of SHIM support in the localhost
  - Can be used for communicating user or application specified Host Identifiers
- Introduces a new socket address structure with new identifier: Endpoint Descriptor (ED)
  - Similar to file descriptor, only local significance

Legacy vs. Native SHIM API

```
struct addrinfo hints, *res, *try;

char *hello = "hello";

int err, bytes, sock;

memset(hints, 0, sizeof(hints));

//hints.ai_family = AF_INET6;

hints.ai_socktype = SOCK_STREAM;

err   = getaddrinfo("www.host.org", "echo",
                              &hints, &res);

sock  = socket(res->ai_family,res->ai_socktype,
      res->protocol); // family=AF_INET6

for (try = res; try; try = try->ai_next)
           err = connect(sock, try->ai_addr,
                              try->ai_addrlen);
      bytes = send(sock, hello, strlen(hello), 0);
      bytes = recv(sock, hello, strlen(hello), 0);

err   = close(sock);
      err   = freeaddrinfo(res);
```

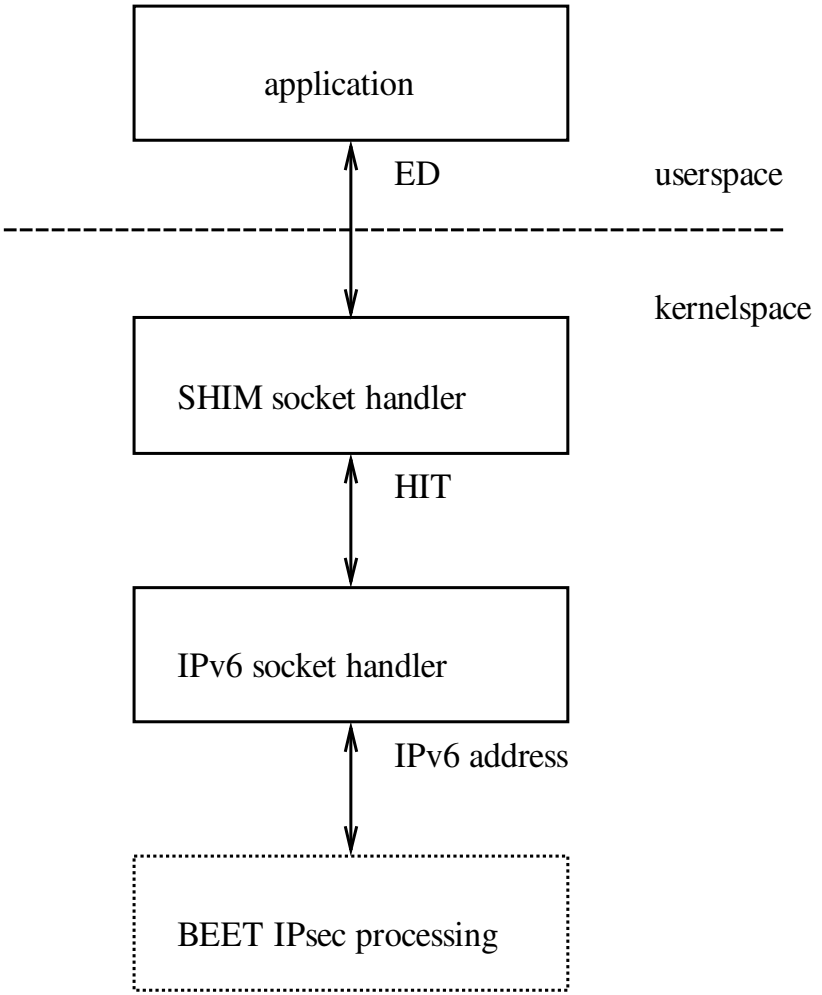
```
struct addrinfo hints, *res, *try;

char *hello = "hello";

int err,  bytes, sock;

memset(hints, 0, sizeof(hints));

hints.flags = AI_ED;

//hints.family = AF_SHIM;

hints.ai_socktype = SOCK_STREAM;

err   = getaddrinfo("www.host.org", "echo",
                              &hints, &res);

sock  = socket(res->ai_family, res->ai_socktype
       res->protocol); // family=PF_SHIM

for (try = res; try; try = try->ai_next)
           err = connect(sock, try->ai_addr,
                              try->ai_addrlen);
      bytes = send(sock, hello, strlen(hello), 0);
      bytes = recv(sock, hello, strlen(hello), 0);

err   = close(sock);
      err   = freeendpointinfo(res);
```

Application Specified Idenfiers

```
int sockfd, err, family = PF_SHIM, type = SOCK_STREAM;

char *user_priv_key = "/home/mk/hip_host_dsa_key";
struct endpoint *endpoint;
struct sockaddr_ed my_ed;
struct addrinfoinfo hints, *res = NULL;

err = load_hip_endpoint_pem(user_priv_key, &endpoint);
err = setmyeid(&my_ed, "", endpoint, NULL);
sockfd = socket(family, type, 0);
err = bind(sockfd, (struct sockaddr *) &my_ed,
     sizeof(my_ed));

memset(&hints, 0, sizeof(&hints));
hints.ai_socktype = type;
hints.ai_family = family;
hints.flags = AI_ED;
err = getendpointinfo("www.host.org", "echo", &hints, &res);

/* connect, send and recv normally */
```

Implementation

- http://infrahip.hiit.fi/
- Linux based implementation
  - the resolver extensions
  - application specified identifiers
  - kernelspace socket handler



Summary

- ED hides the presentation of HITs and HIs
  - Opportunistic HIP
  - Deployment of new identifier layers
  - Size of HIT may change in the future
- PF_SHIM family can be used e.g. for detecting SHIM support in the localhost
- Native SHIM API extends HIP architecture by allowing apps to have their own id

Open Issues

- Use EDs or HITs?
- Semi-legacy APIs required?
  - Applications use HITs but are able to set HIP specific (system-wide) socket options
- Applicability to BTNS
- Accept as a official WG item?

Related Work

- Applying a Cryptographic Namespace to Applications [Komu et al]
- draft-mkomu-hip-native-api-00
- Application Programming Interfaces for Host Identity Protocol [Komu]
- draft-henderson-hip-applications
- draft-sugimoto-multihome-shim-api

Questions / Feedback

Miika Komu <miika@iki.fi>
Helsinki Institute for Information Technology