Vertical Handover Study Cluster: VHO Daemon 8.3.2005

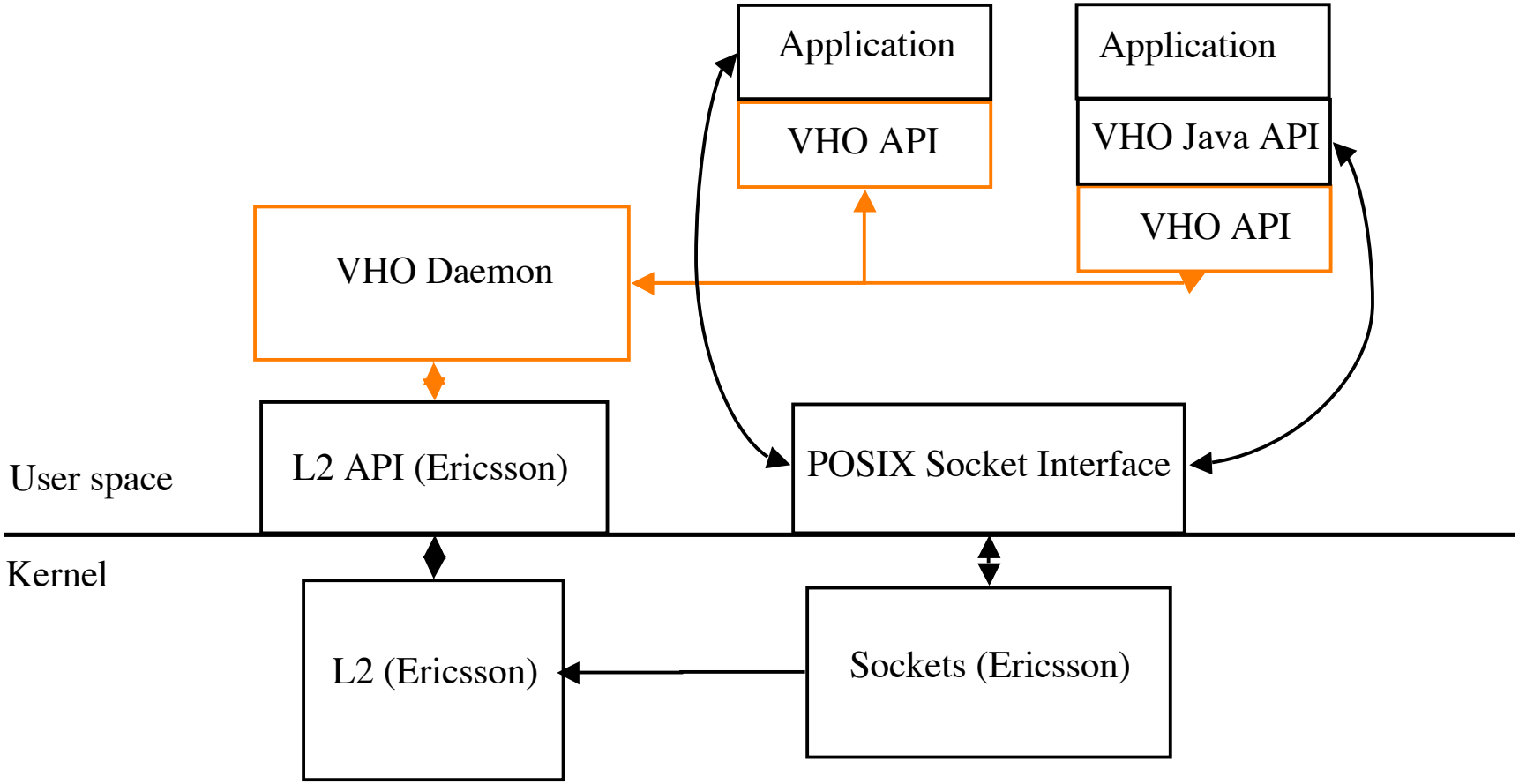
Raimo Nikkilä
PM&RG – Product Modelling and Realisation Group
Department of Computer Science and Engineering
Helsinki University of Technology
P.O.Box 5400, FIN-02015 HUT, Finland

Introduction

- A user space daemon for the VHO Platform
- Actually a daemon and an API

- Used for VHO related application communication
  - Mostly forwarded information from the OS kernel

Position in the Architecture



Implementation

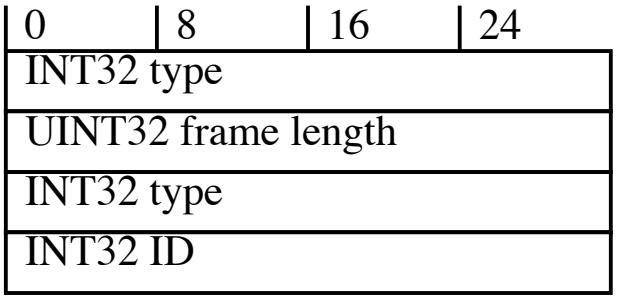
- Daemon and API Written in ISO C99
- POSIX compliant, will not work in wintendo
- A real daemon, not just something named daemon
- Written from scratch
- Requires L2 to operate (Currently Linux only)

- Java API built on the C API using JNI 1.4 exists

Daemon – API Communication

- Uses UNIX local stream sockets for the IPC
- Asynchronous server
  - Supports any practical number of applications
- Simple binary protocol is used for the communication
  - No portability issues as the daemon and API reside on the same physical machine
- Communication is mostly daemon -> API

Example of the Binary Protocol



Frame header

Connection frame

- Currently uses two frames
- 6 Different frames
  - Text, connection, L2_event, L2_raw, handover and info
  - More to come

L2 And the VHO Daemon

- VHO Daemon is the only user space process communicating directly with L2 API
- All L2 events are sent to all applications using the VHO API
- Events are available in either raw or parsed format

Using the VHO Daemon

- No configuration necessary
- Needs to be run as root
- /var/run/vhod.pid
  - Contains the daemon PID
  - Makes sure that only instance of the one daemon is running
- /var/run/vhod.sock
  - The local socket used for communication
- Logs all events to syslog
  - Usually very silent
  - If something goes wrong, read the syslog
- Can be terminated gracefully with signals such as SIGTERM

Handovers with the VHO Daemon

- Daemon learns of a handover from the L2
- Information is then forwarded to all API's
  - Everyone will know of the handover
- Currently the information consist of:
  - PID of the process receiving the handover
  - FD of the socket which is handed over
  - DID of the device the socket was previously on
  - DID of the device the socket will now be on

VHO API

- Legacy free
  - Has nothing to do with the VHO Adapter previously implemented
  - Like Daemon, written from scratch
- Easy to use
- Based entirely on callbacks
- Using one callback form
  - `void (*callback)(int id, void *ptr, void *args);`
- Provides simple enough functions for sending frames to the VHO daemon
- Comes with a few default callbacks, otherwise everything rests on the application programmer.
- Features and functionality will be appended as found necessary

Information Available Through the VHOAPI

- System device related
  - Device add and remove notification
  - Device link status
  - Device information
    - ID, type, name, hardware address, bandwidth and latency
- Handovers
- Information on other VHOAPI instances
  - Connections
- Daemon events

VHOAPI Callbacks

- Everything is based on the callbacks
  - Authorative list is at vhoapi/callbacks.h
  - Version number VHOAPI_VERSION included
- Some callbacks as examples:

```
CB_L2_LINK_UP,        /* (int did) */
CB_L2_LINK_DOWN,      /* (int did) */
CB_L2_DEVICE_ADD,     /* (int did) */
CB_L2_DEVICE_REMOVE, /* (int did) */
```

Example of a Simple Callback Function

- Actual callback function

```
void my_function(int event, void *unused, void *params)
{
    if (event == CB_L2_DEVICE_ADD)
        fprintf(stdout, "new dev: %d\n", ((int*)args)[0]);
}
```

- To add the function
  - `add_callback(CB_L2_DEVICE_ADD, NULL, &my_function);`
- The parameter type change is complex but required

VHO Java API

- Built on top of the VHO API using JNI calls
  - To the VHO API it is just another application
  - To the Java application it is an interface, similar to the VHO API in C
- Consists of a Java class and callback interface and a native (C) part with JNI calls to and from the Java side.
- Java application designers just implements the interface for callback functionality allmost as they would have in C.
- Callbacks from the VHO API passed to the native (C) part is passed up to predefined callback methods
- Calls from Java to the VHO API are passed through native Java methods and implemented in C.

VHOAPI Future

- Include future changes of L2 and platform
- Support various simulation features
  - Parameters not yet available
  - Easy debugging of applications
- Possible future development
  - Policies and network selection
  - Application to kernel messaging