medkit - Manually Embedded Device Konfiguration and Integration Technology

Name is negotiable

Motivation

Medical devices are extremely expensive, with a huge cost to the populace [cdc-medical-debt-source], including a very large population of Americans who do not seek, or adhere to, medical treatment as a consequence of prohibitive cost and the spectre of crippling medical debt [crna-paper]. In hopes of addressing this fact, many efforts have arisen to design low-cost medicaldevices [insulin pump, etc], and a whole host of low-cost sensors[wearables paper, etc]. These devices are difficult to put into production due to the high input cost often required to attain medical certification, and are often in an environment full of incompatibile propriatary medical devices and sensors that may not have the possibility of interoperation, if they even have any hardware to do so.

To solve this problem, this document sets out two designs:

- **I) The Protocol** - a specification for communication over a network of connected medical devices.
- **II) The Bridge Device** - a device that, once manually installed on an arbitrary, well-specified host device and allows it to interact with the network by implementing The Protocol (I).

(I) The Protocol

The protocol specifies the interactions between several wearable medical devices, specifically three main categories:

- **Sensor Devices**: devices that measure biometrics and transmit them to the network.
- **Delivery Devices**: devices that can be commanded to perform a set of medical tasks in accordance with a delivery policy
- **Control Devices**: devices communicating delivery policy details and managing information accumulation and exchange across the network as a (locally) central point.

The primary communication over the network falls under these general categories (with exceptions):

- devices implementing **Sensor Device** behavior communicating with a **Control Devices** to send biomedical information
- devices implementing **Control Device** behavior communicating with a **Delivery Device** to change a delivery policy.

It is also recognized that it is unreasonable for a design to be adopted broadly by manufactorers, or previous designs lacking in the hardware to communicate with the design without major change in operation, which is costly, and unlikely. Instead, the protocol is designed to be easy to implement the hardware interfaces as software modules, which will allow an easier process of building an open source database of reusable device configurations. This concept will be leveraged in section (II), where a device design is proposed that will allow a large class of button-operated devices to be integrated almost natively.

Protocol Structure

To explore the organization of the network, we will chose the example of a wearable network designed for a person with diabetes. A common configuration would be the following:

```
           glucose data
<Sensor: CGM> --------> <Control: Delivery Policy> --------> <Delivery: Insulin Pump>
                                                    delivery guidance
```

A continuous glucose monitor connected to a user reports glucose levels to a controller. The controller monitors and adjusts its delivery response and communicates its guidance to the insulin pump. The insulin pump verifies that the insulin directive makes sense and adjusts its rate of delivery accordingly.

In certain phases of this network, however, it is clear that a delivery policy might want to query actual real-time delivery in its own modelling, in which case we end up with the following relationship:

```
              insulin delivery data
<Sensor: Insulin Pump> ------> <Control: Delivery Policy>
```

With this demonstration, it is clear that a very flexible protocol is needed, with devices not fitting into neat categories.

In recognition of this, The Protocol is designed as a mechanism for communication where devices can negotiate configuration options directly with a control mechanism, from broad biometric data to sampling rates and the abstraction of wearable medical hardware.

In an ideally interoperable configuration such as this, the drug delivery polcy controller, insulin pump, and glucose monitor could all be developed seperately and interchanged, allowing a modular and easily upgradable medical device system

Specficiation

The Protocol is designed to operate in OSI Model Level 5 and 6 [osi model specification]. In a hypothetical larger, commerical stage of the project, a definitive protocol would have to be solidified. The device implemented in (II) will be done over Wi-Fi, for example, but a more realistic situation would likely use bluetooth due to the limited range of connections, wifi connection not needing to be configured manually, etc.

Each connection is split into 3 states:
- **Phase I: Authentication**: The device connects to a nearby device and exchanges initial information used for encrpyting and identifying future messages
- **Phase II: Configuration**: Period of initial communication where details such as sampling rates, feature ability and parameter options enumerated and agreed upon.
- **Phase III: Steady State Operation**: Normal mode of operaton where delivery commands are processed by the system.

Phase I Authentication

Phase II Configuration

Phase III Steady-State Operation

Once Authentication and configuration is finished, the devices communicate using packets of the following two forms:
- Requests: sent by devices to initiate a stream of communication
- Responses: sent in responses to requests

Once this exchange is made, a more domain-specific protocol could be designed on top of this, using the basic authentication and configuration operations to probe support for different communication models.

The Request has a 16-byte header:

```
+-------+-----------+-------------+------+--------------------+
| Size  |     4     |      2      |  2   |         8          |
+-------+-----------+-------------+------+--------------------+
| Field | REQ MAGIC | Message Type |  ID  | Type-Specific Data |
+-------+-----------+-------------+------+--------------------+
        0       4                 6      8                   15
```

and the Response:

```
+-------+-----------+-------------+------+--------------------+
| Size  |     4     |      2      |  2   |         8          |
+-------+-----------+-------------+------+--------------------+
| Field | RES MAGIC |  Status Code |  ID  | Type-Specific Data |
+-------+-----------+-------------+------+--------------------+
        0           4             6      8                   15
```

The fields are as follows:

REQ/S MAGIC: A 4-byte recognizable signature to identify proper starting packets, along with their type (request/response).

Message Type: One of the following:
- Reconfiguration Request - asks the device to reenter the configuration stage, either to update new hardware capabilities, or as part of an error recovery process.
- Set Parameter Value - requests a parameter change on a device implementing the delivery behaviors.
- Read Parameter Value - requests to read back a parameter on a device implementing the delivery behaviors.
- Get Device Status - requests a small informational block describing the error state of the device, for debugging.
- Notify Device - sends a regular generic data packet.
- Request Sensor Hook - requests a device implementing the Sensor to implement simple logic-based asynchronous I/O

Example implementation in a C-like syntax

```
struct Request {
  u8   magic[4];

  enum MessageType {
    RECONFIGURE,
    SET_PARAM,
    READ_PARAM,
    GET_STATUS,
    NOTIFY,
    RequestSensorHook,
  } message_type :32;

  union {
    struct Reconfigure {
      u8 must_reauthenticate;
      u56 resv;
    } reconfigure;
    struct SetParam {
      u32 param_selector;
      u32 param_value;
    };
    struct ReadParam {
      u32 param_selector;
      u32 resv;
    };
    struct GetStatus, Notify {
      u64 resv;
    };
    struct RequestSensorHook {
      enum HookType {
        SENSOR_UPDATE,
        SENSOR_LEVEL_UPPER,
        SENSOR_LEVEL_LOWER,
      } event_type :u32;
      u32 sensor_id;
    };
    struct Notify {
      u32 raw_data;
    }
  } content;
};
```

medical devices: sensors, delivery devices, and control devices the network: wearable medical devices connected to one or more other devices

Devices can choose to mark data as protected (default) or public. In the case of public data, other devices will be able to query values reported by sesnors. This could be useful in the case of devices that want to have a histograph display.

ERROR Codes
EUNSUPPORTEDACTION - A command has been requested that does not fit the listed capabilities of the device

Considerations

Before an actual release could be made with this software, a much more robust cryptographic approach would obviously have to happen. The current risk of an attack where an attacker is able to get within close range bluetooth and drastically threatening a user's health and safety, or upload malicious drivers is far too great a risk to take without extensive analysis that extends beyond the current scope of the design project.

A few major precautions to take now:
- Specifying non-volatile constant limits in rate based on a given amount of time (i.e. you can't spike

the insulin if you can only increase between safe levels without manual adjustment in edge cases).