KNX Implementation for Plan 9

Gorka Guardiola Múzquiz
Enrique Soriano Salvador
Francisco J. Ballesteros

The problem

- Home full of devices/actuators:
  - Tv, dvd player, washing machine…
  - Thermostat, light switches
  - ADSL gateways and wifi networks
  - More and more sensors/actuators coming

We want

- Add some more sensors
- Control everything from everywhere

How

- Small computers (one? many?)
  - Gumstix/sheevaplug/beagleboard/…
- Networks (wifi when possible)
  - We need power, so sometimes a cable is ok as long as it is just one
- Sensors? Actuators?…

Sensors/Actuators

- Tried X10 (network is the power cable)
  - Doesn´t work for us
  - Power networks are bad in Spain
- We wanted to try Knx (its own bus cable/radio)
  - Write a USB driver for the coupler
  - Program the devices from the computer
  - Use them
  - How hard can it be? (Spoiler: quite)

KNX

- Used to be EIB (European installation bus)
- Defines a bus and protocols (all levels)
- The gateways available simply forward packets
- We want to export them as 9P using a small computer (gumstix, sheeva…)

KNX

- Way to program devices so that they change their behaviour, talk to each other…
- Like a kind of weird assembler java
- We are not interested in this, we have an external controller, devices: as dumb as possible
- We do want to configure the addresses
- One object per interface in a device for us (each switch, each sensor…)

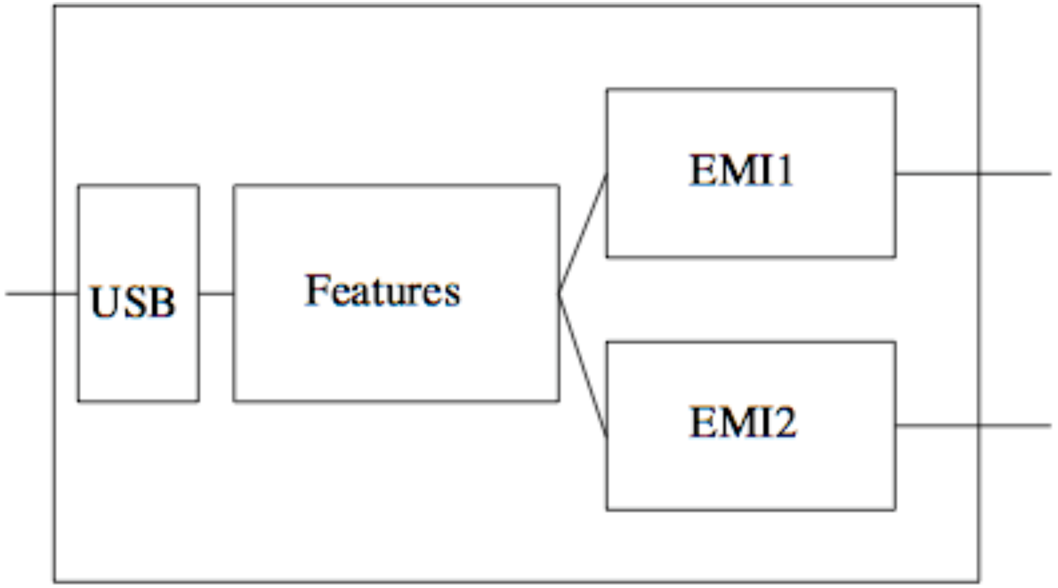
Emi

- External message interface
  - Packet definition for talking with devices
  - In the bus you get Imi (internal message interface)
  - Most of it is not byte aligned (sigh!)

KNX USB coupler

- It shows itself as an HID
- Two 64 bytes interrupt endpoints (in/out) plus ctl
- Two parts
  - Bus Access Server (Features)
  - One or more Emi servers

KNX USB



KNX USB

- Features, for configuring the device itself (layer, Emi type…)
- Emi servers for talking to the devices
  - The coupler itself has an address and is like a device

Network protocol stack

- They have everything possible
- Link/Transport/Network
- ISO request/confirmation/indication
- Confirmation is local (what is the point?)

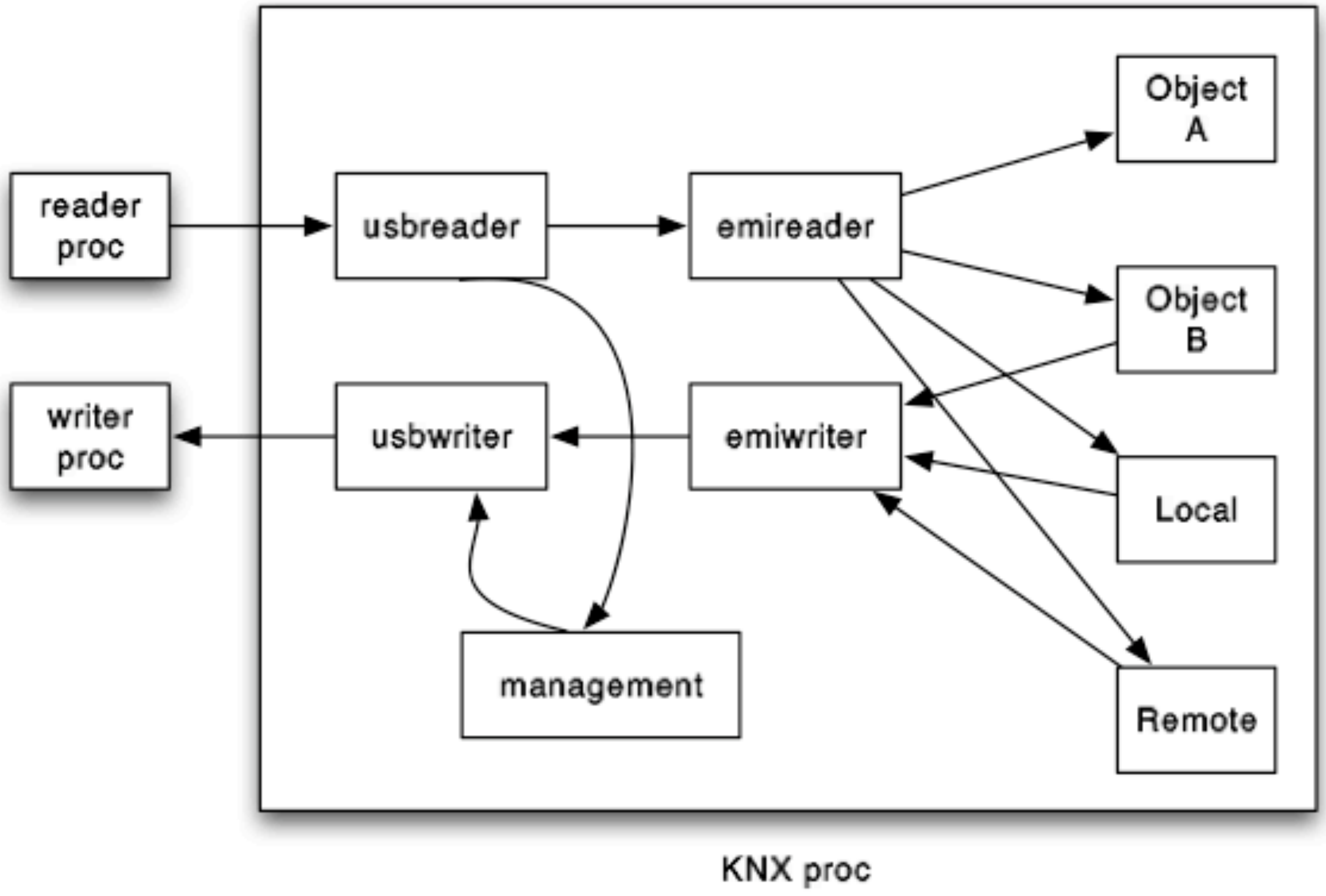
Network protocol stack

- Emi server configured as link layer (almost transparent, cannot see other connections)
- Will zero the layers under (if configured above)
- To prevent race conditions
- While configuring addresses (like an inverse dhcp), we would need to switch
- We manage everything

Network protocol stack

- We ignore confirmations
- We use one thread per device (and some extra)
- We do Stop and Wait
- We keep the temporal state in the stack

Procs/Threads



Threads

- Usbreader/usbwriter: raw HID packets
- Emireader/emiwriter: raw emi packets
- Local: emi packets with local origin
- Remote: emi packets multicast, no address
- Management: non-emi (Bus access features)
- Objects: A thread representing each configured seen object (we have a unique global address space address/object)
- One object per object + an object per device

What we can do

- 4500 LOC after
- We can configure the devices (mainly the addresses of objects and devices)
- We can use them
- We can sniff conectionless messages

What we can´t do

- Working on the filesystem (no 9P server yet)
- We cannot completely detect the available objects
- We cannot program the devices themselves (not our aim)

Q/A?