Scopes + TikiDB stack

Introduction

Goal of the Scopes + TikiDB stack solarhouse WSN lab project was to establish a functional system that consinst of Contiki 2.5, Scopes and TikiDB on Z1 motes from Zolertia.

Initial assignment

- **Code cleanup**
  Most important point in assignment was code cleanup of Scopes and TikiDB. Especially code size reduction was important, since motes have very limited memory.
  First of all it was necessary to get a rough overview over the stack. Identify modules, sub modules and their responsibilities. Next step was to get them compiled. This task wasn't that easy, since it was required to run the stack on Contiki 2.5. Scopes and TikiDB were written for Contiki 2.4, so changes in 2.5 influenced the stack due to API changes in Contiki. Changes that had to be made were mostly correction of include files and adjusting changed data types. Next problem was that the linker messaged about "text segment exceeded". Such messages are exactly the reason, why code had to be cleaned up. It was simply too large.

  Scopes framework has a sub module called "repository". It is responsible to assign motes to a scope, based on predefined criteria. In original state the criterias could be almost arbitrary complex. This code was replaced by a much simpler implementation, which is less versatile. It assigns motes to a scope based only on node_id. This solution fully satisfies our needs and saves a lot of code space.

  In TikiDB was the sensortable significantly reduced. This affected not only the lookup tables and access functions, but also the long switch-case cascades in "queryresolver" submodule. In "db" and "dbms" were all aggregation functions removed, since we don't need them. This optimization also heavily affected "querymanager", which was responsible for interpreting such complex queries. Among with query optimization "querymanager" was also lightened, because it now don't have to manage queries' lifetime.

  Among those "big" optimizations a lot of other tweaks were made, like removing and inlining of one-line-functions, to save couple of ASM bytes and stack increase.

- **Query definition**
  Query definition was straight forward. Since no special requirements were formulated and only four fields are interesting/readable, all fields are packed into one query which is executed every 15 minutes.

- **Query optimization**
  Queries are optimized by means of definition/interpretation/execution. That means, that all unneeded features were removed. Namely query lifetime and aggregation support. Since we decided to use netflooding as routing protocol, no other query optimizations were made.

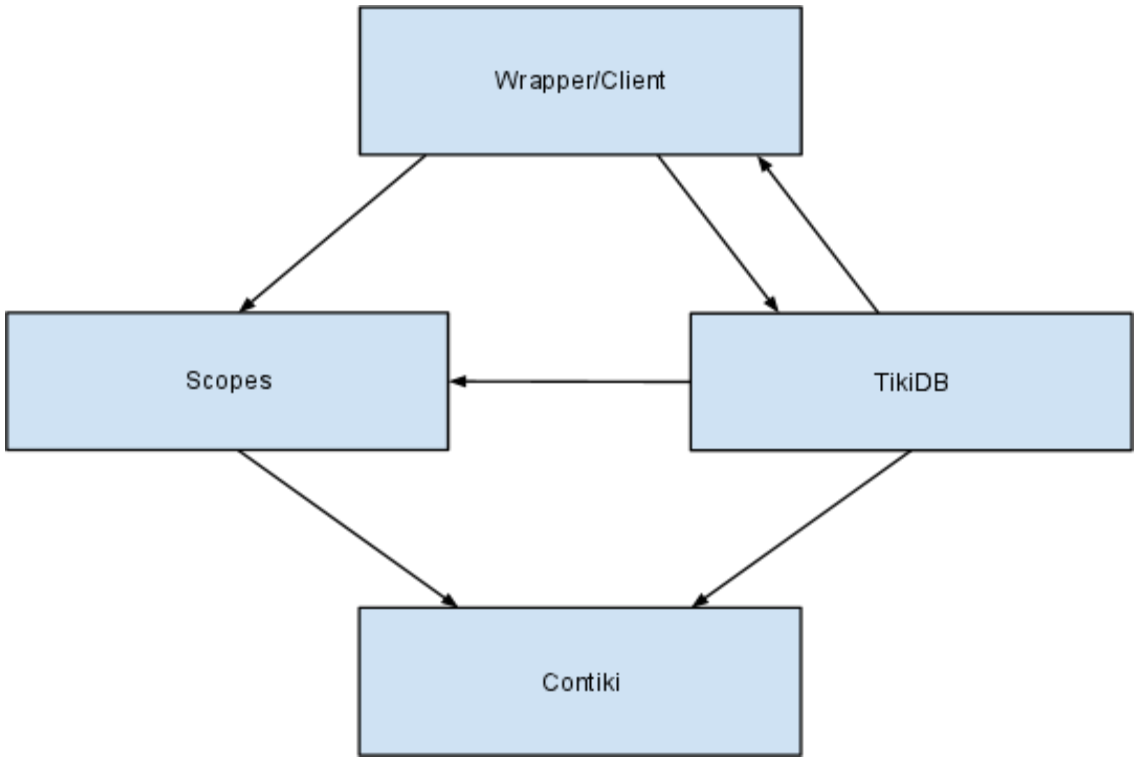
- **Query dissemination**
  Query distribution works as follows: every node starts a process when it joins a scope.

This process checks the node for a query. If none is present, it sends a special request to the base station. Base station answers this request by sending query definition to the network. Child nodes, which already have the query simply ignore the package and the child node, which sent the request accepts and stores the query definition. This algorithm ensures, that no child node “hangs around” in case it once had to be rebooted. If a child node leaves a scope, it stops the “query-watch” process and drops the query to avoid sending unneeded data and lower the power consumption. Another case is, if base station had to be restarted (i.e. watchdog). If that happens and base station is back, it waits for SCOPE_TTL amount of time to make sure, all children dropped their queries and left the old scope.

Additional assignment

- **DB-Writer**
  DB-Writer is a software piece, that collects the data delivered to the base station and stores it in a MySQL database. To accomplish this task, the “serialdump” program, which is part of Contiki, was modified. It serves now as serialdump and MySQL client at once. It buffers data incoming through serial port until it reads a newline char. Otherwise it would be quite difficult to parse the data. After a line was read, it its beeing parsed. From the parsed data a SQL statement is beeing constructed and submitted to the target database.

- **Restructuring Scopes + TikiDB stack**
  Scopes and TikiDB were originally developed by two independent development teams. It was evident, because in both modules had parts which logically belonged to other module. Scopes had a simple query system implemented while TikiDB had a simple datalink layer. This fact makes it difficult to bring this modules together. One approach called “wrapper” was initially delivered. It worked fine but it introduced undesired dependecies between those three modules. Following figure illustrates these dependencies. As can be seen, TikiDB depends on wrapper/client code. That means, if you want to write a new client that uses TikiDB, you have to make a copy of it and intrusive change the code. This circular dependency was resolved by using the Dependency Inversion Principle. TikiDB now calls “abstract” functions <[TODO: function names]> implementation of which client code has to provide.



- **Testing**
  Really a LOT of!

Result

Final system works fine on Z1 motes. Scopes and TikiDB are now compatible with Contiki 2.5. Room for extensions is there too. Measurments between first compileable version and final result give a difference of about 3KB space.