Verteilte Sensornetzwerke

Mit Datenaggregation und Sensorfusion

PD Stefan Bosse

Universität Bremen - FB Mathematik und Informatik

LuaOS

1. Sensoren
2. Klassen und Objekte
3. Kommunikation
4. Synchronisation
5. Gruppenmanagement und Gruppenkommunikation
6. Threads und Coroutinen
7. LuaOS Router

Gruppenkommunikation

Nachrichtenbasierte Gruppenkommunikation

- Nutzt den Gruppenservice des luaosrouter Programms für die Gruppen
- Gruppenmitglieder können in einem vollständig verbundenen virtuellenNetzwerkgraph miteinander kommunizieren

Routerverbindung

1. Zuerst muss eine Verbindung zum Router (Gateway) *luasosrouter* hergestellt und ein virtueller (remote) Kommunikationport erzeugt werden:

```
myroute = route:new(url)
myport  = myroute:connect()
myid    = myroute.id
status  = myroute:ping()
```

Gruppenmanagement

2. Gruppe erzeugen (wenn sie nicht existiert)

```
status = myroute:create(groupname)
```

3. Einer Gruppe beitreten

```
status = myroute:join(groupname)
```

4. Gruppenmitglieder erfragen

```
members = myroute:ask(groupname)
```

5. Gruppe verlassen

```
members = myroute:unjoin(groupname)
```

Kommunikationskanäle

1. Verbindung zu einem Gruppenmitglied herstellen:

```
status = myport:connect(peerid,birdir?)
```

Wenn das *bidir* Argument gesetzt wird, kann auch die andere Seite Nachrichten direkt zurücksenden.

2. Nachricht senden

```
status = myport:write(datat,peerid?)
```

3. Nachrichten empfangen (Sammelpunkt)

```
data,err = myport:read()
```

4. Verbindung zu einem Peer trennen

```
status = myport:disconnect(peerid)
```

Nachrichtenempfang

Die Empfangsoperation blockiert den Prozessfluß und sollte daher in einer Koroutine in einer Schleife ausgeführt werden:

```
co = coroutine.create(function ()
  local err;
  while not err do
    local data,err = myport:read()
    ..
  end
end); coroutine.resumeCatch(co)
```

Alternative ist die Einrichtung einer Rückruffunktion und asynchroner Nachrichtenverarbeitung:

```
myport:receiver(function (data,err)
  ...
end)
```

Lua OS Router

```
lvm luasrouter [-v -h]
               [-sec keynameORport ]
               [-host IP/URL/?]
               [-port NUM]
               [-portHTTP NUM]
               [-portWS NUM]
```

- Soll der Lua OS Router außerhalb des eigenen Rechner im Netzwerk sichtbar sein, muss das Host Argument (lokale IP Adresse des Netzwerkgerätes, keine Domaine!) gesetzt werden, z.B.:

```
lvm luasrouter -host 134.102.219.1
```

Virtuelle Gruppenkommunikation VGRPC

Virtual Group Remote Procedure Call

- Nutzt den Gruppenservice des luaosrouter Programms für die Gruppen
- Gruppenmitglieder können in einem vollständig verbundenen virtuellenNetzwerkgraph miteinander kommunizieren
- Zwei Kommunikationsarchitekturen
  1. Remote Procedure call (RPC)
  2. Container mit Sende- und Empfangsfunktion

Anmeldung

- Über die Anmeldung wird:
  1. Eine Verbindung zum dem luaosrouter hergestellt (*url*, z.B. `edu-9.de:22223`);
  2. Ein Kommunikationsport mit peerid erstellt;
  3. Die Gruppe *group* wird erzeugt (wenn nicht vorhanden);
  4. Der neue Port wird der Gruppe hinzugefügt (aber nocht nicht verbunden);
  5. Für Gruppenkommunikation kann Standardzeitablauf (*timeout*) angegeben werden.

```
vgroup = vgrpc:new(url,group,verbose,timeout);
vgroup.error -- Fehlerstatus
vgroup.port.peerid -- Meine PeerID
vrgoup.router, vgroup.port -- Router/PeerPort
```

Beitreten und Verlassen von Gruppen

- Gruppen sind dynamisch. Aktuelle Gruppenmitglieder können über den Gruppenservice abgefragt werden (`vgroup:router:ask(group)`)
- Der eigene Port muss noch explizit mit den aktuellen Gruppenmitgliedern verbunden werden (oder später wieder verlassen werden):

```
vgroup:enter()  -- connect with all group members
vgroup:leave()  -- disconnect from all group members
```

Remote Procedure Calls

- Mittels der *call* Methode können bei den Gruppenmitlgiedern registrierte Funktionen aufegrufen werden. Der Aufruf wartet auf die Antwort aller Gruppenmitglieder (oder Timeout)

```
local function foo (x,y) return x*y end
vgroup:service("foo",foo);
data = vgroup:call('foo',1,2) -- table!, oder :trycall(timeout,..)
```

Senden und Empfangen von Tag Nachrichten

- Es können Gruppenmitgliedern auch explizit Nachrichten (mit Tag) gesendet werden
- Empfang muss registriert werden (*receive*) - blockiert Programmfluss!
- Es wird immer auf den Empfang von N Nachrichten von allen Gruppenmitlgiedern gewartet (oder Timeout); bei $N+1$ Mitgliedern in der Gruppe insgesamt

```
-- coroutine?
data = vgroup:receive(tag,timeout?)  -- table!
mygroup:send(tag,data)
```

Nachrichten Kontainer

- Ein Kontainer kann als Empfänger für getaggte Nachrichten **vorher** eingerichtet werden
- Nach der Einrichtung werden nachrichten gesammelt (max. *N*, oder Timeout **ab** *await* Aufruf)

```
container = vgroup:container(tag,timeout)
vgroup:send(tag,math.random())
data = container:await() -- table!
```