RESTful Services für 6LoWPAN Home Automation Networks

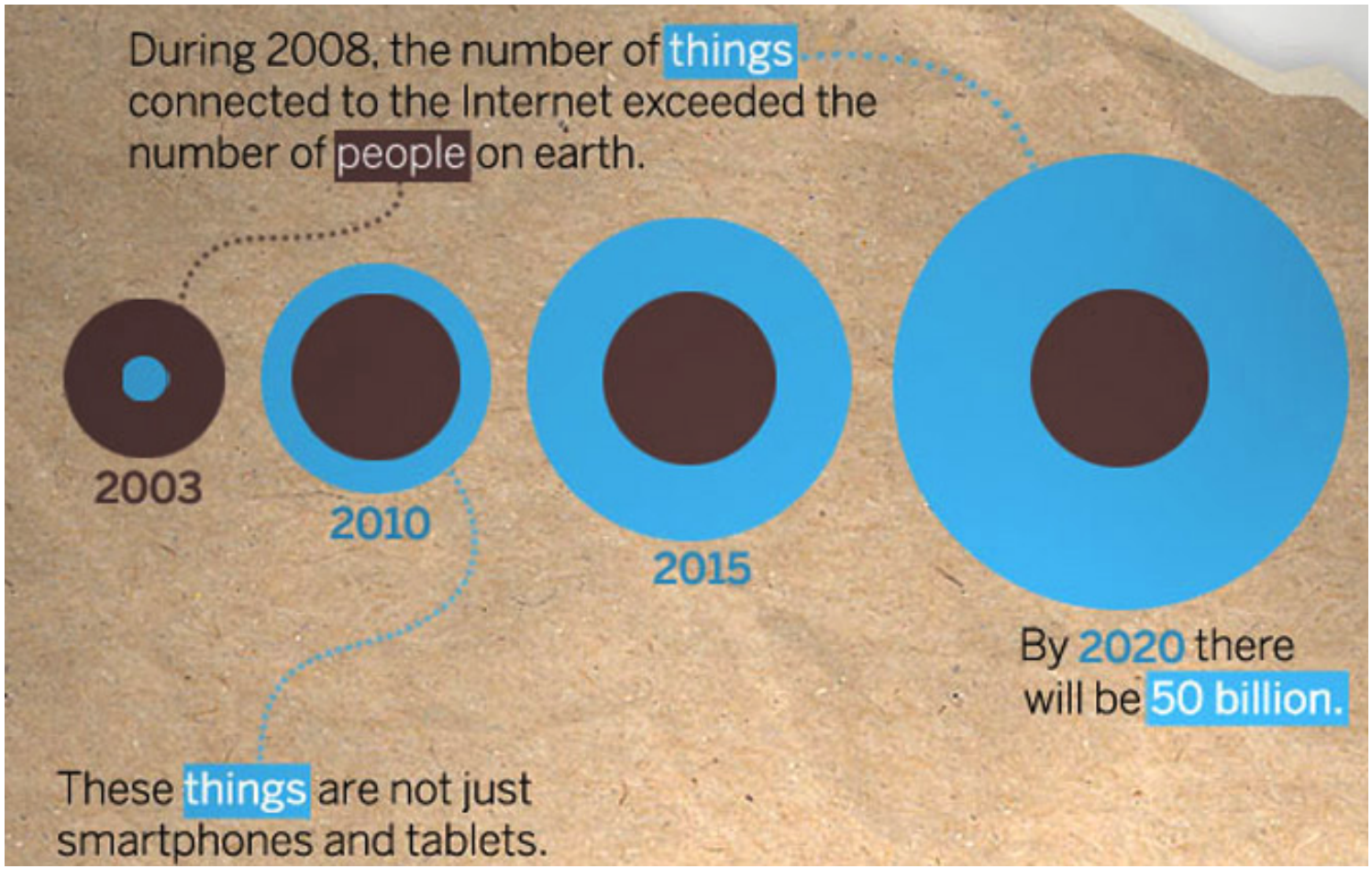
Thomas Scheffler
scheffler@beuth-hochschule.de

BEUTH HOCHSCHULE
FÜR TECHNIK
BERLIN
University of Applied Sciences

Heise IPv6 Kongress, Frankfurt 23. Mai 2014

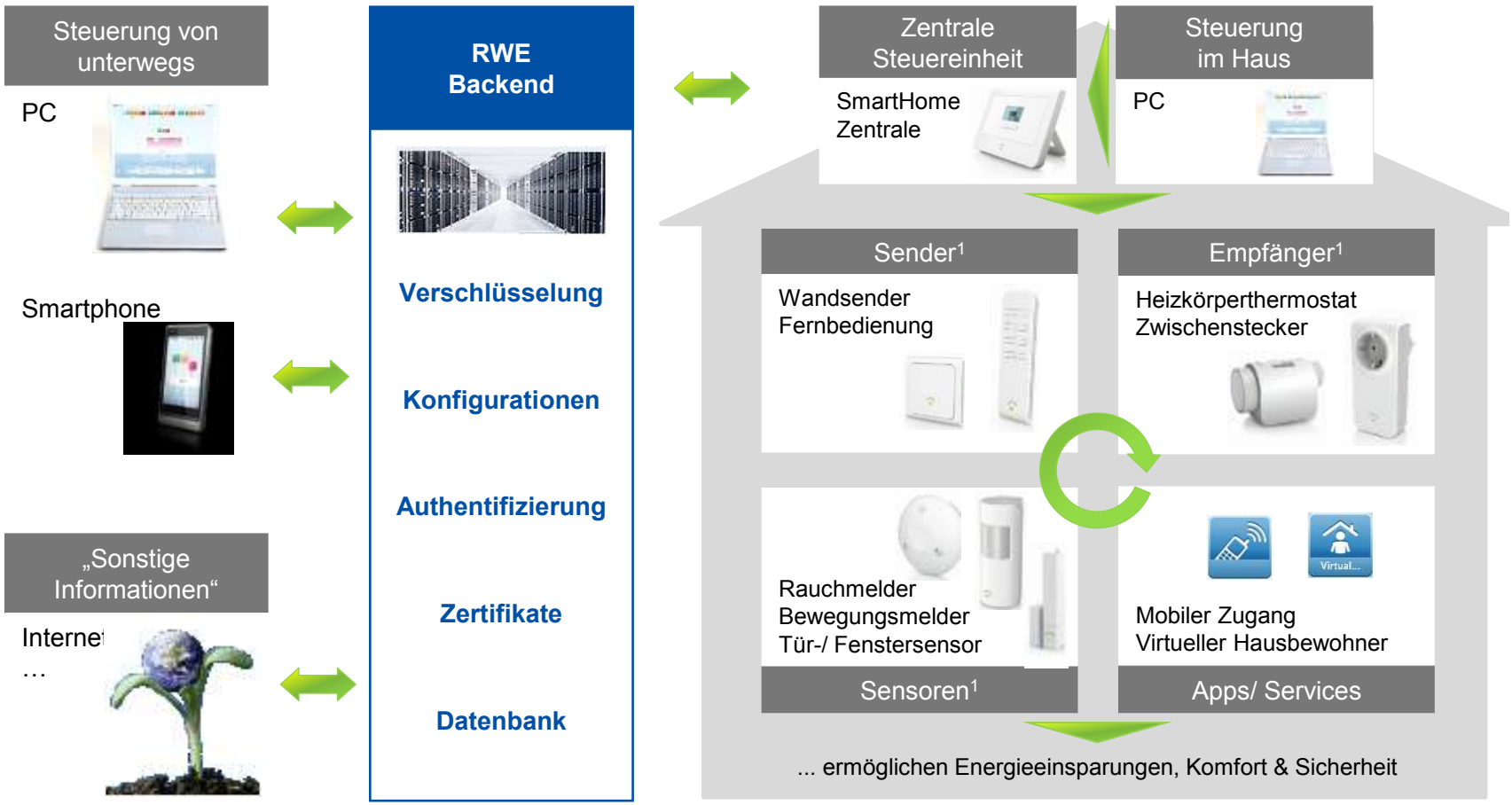
Inhaltsverzeichnis

Vernetzte Geräte im *Internet der Dinge*



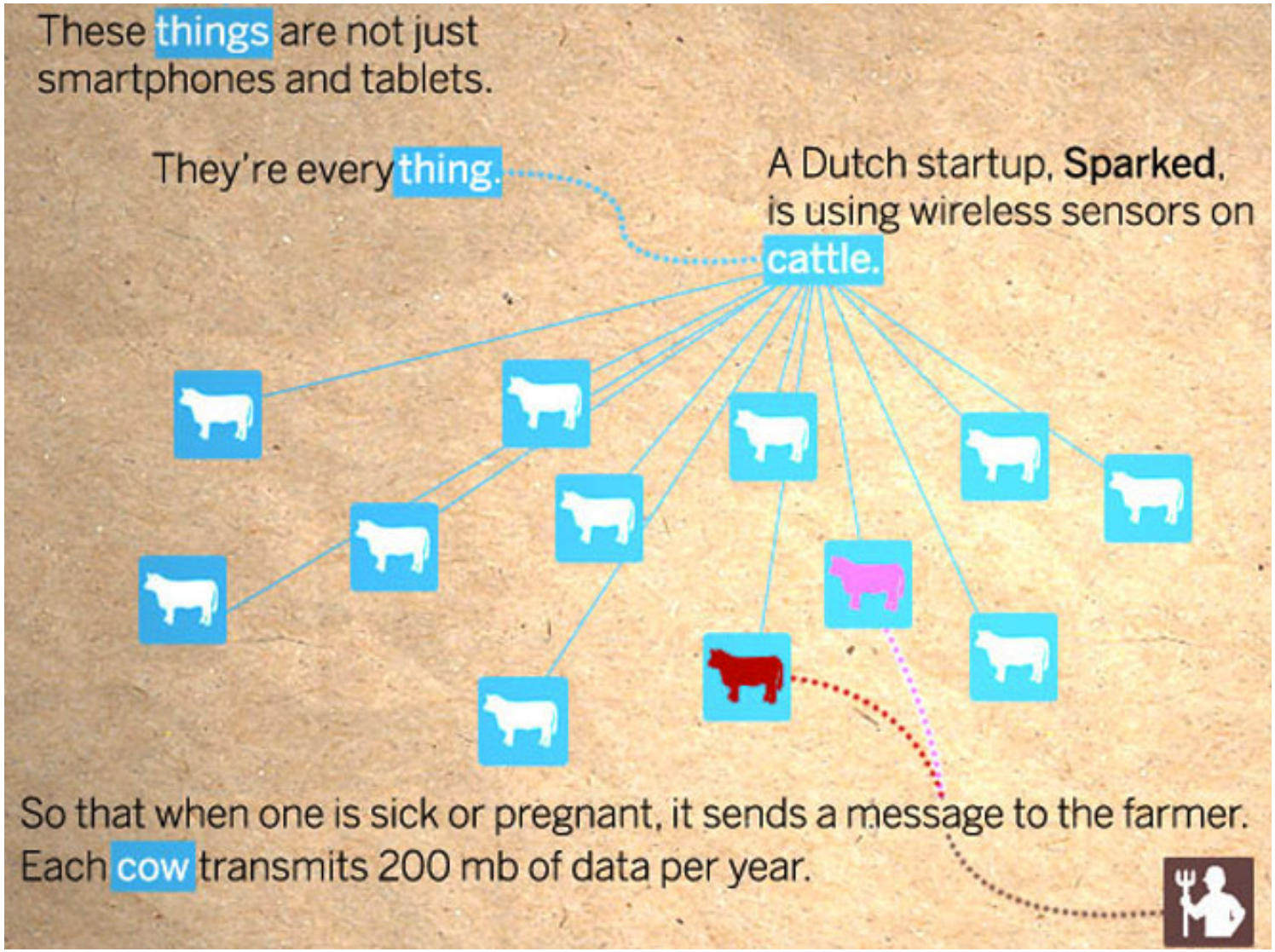
Quelle: http://share.cisco.com/internet-of-things.html

Home Automation aus Sicht eines Service Providers



Quelle: http://www.smarthome-deutschland.de/files/upload/1352627292_Verweyen.pdf

Was ist das *Internet der Dinge*?



Quelle: http://share.cisco.com/internet-of-things.html

Internet Protocol - Version 6 (IPv6)

Das Kernprotokoll wurde 1998 von Steve Deering und Rob Hinden in RFC 2460 beschrieben:

- Adressen mit einer Länge von 128-bit (16 Byte) bilden eine nahezu unerschöpfliche Ressource

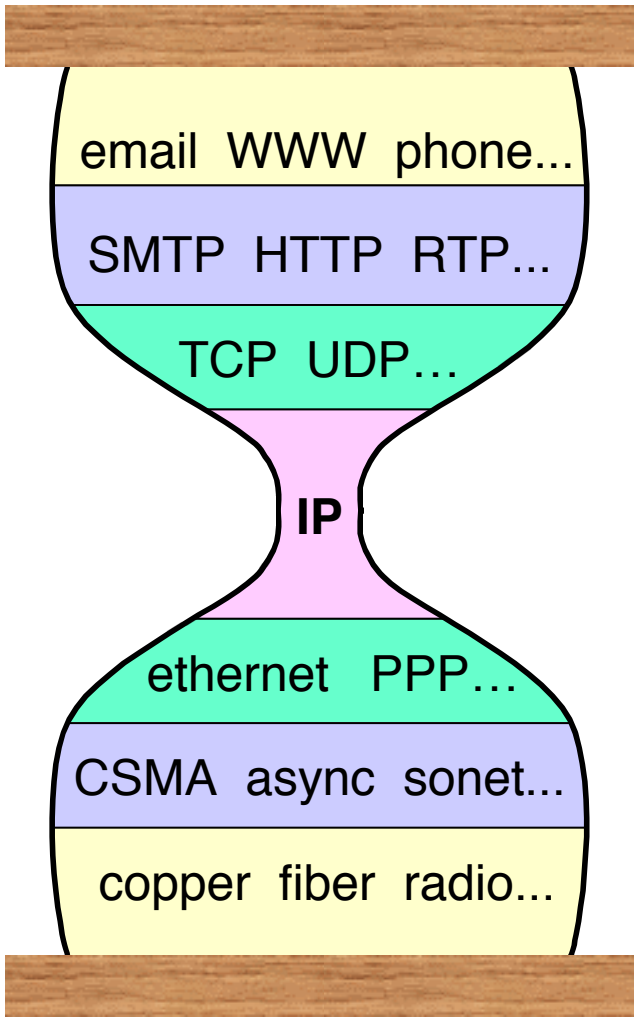
$\Rightarrow$ Auslagerung von Funktionalität in Erweiterungsheader

- IPv6-Endsysteme werden im Netz automatisch konfiguriert
- Strukturierte Adressierung (providerbasiert, hierarchisch) erlaubt effizientes Routing auch in der Site

IPv6 als *Enabling Technology* für das zukünftige Internet

Quelle: http://share.cisco.com/internet-of-things.html

Das *Internet Protocol* verbindet Netze und Anwendungen



Quelle: http://www.iab.org/wp-content/IAB-uploads/2011/03/hourglass-london-ietf.pdf

Demo

Anbindung eines 6LoWPAN Netzes über WLAN-Router

Smart Objects als Hosts im 'Internet of Things'

*"... a **smart object** is an item equipped with a form of sensor or actuator, a tiny microprocessor, a communication device, and a power source."*

[Vasseur & Dunkels, 2010]

Herausforderung: Einsatz von Smart Objects

- Unterstützung einer große Anzahl von Objekten und Ad-Hoc Routing
- Geringer Ressourcenbedarf
- Unterstützung sicherer Application Layer
- Nutzung etablierter Netzinfrastrukturen ohne Einsatz proprietärer Gateways
- Verwendung offener, getesteter IP-basierter Protokolle

$\Rightarrow$ Verwendung etablierter Anwendungen und IPv6 auf einer Mikrocontroller-Plattform

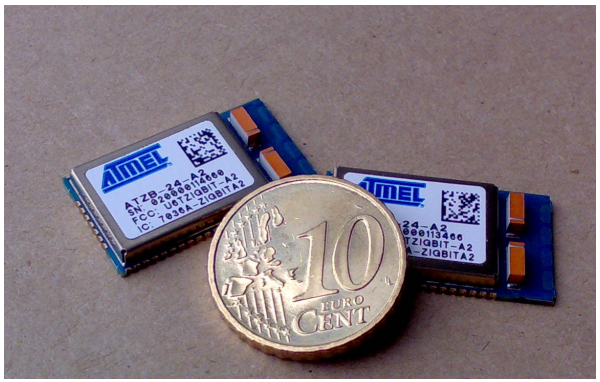
Bausteine für das *Internet der Dinge*

Contiki OS

- Offenes Betriebssystem für Embedded Devices
- Entwickelt vom *Swedish Institute of Computer Science* - aktive internationale Entwicklergemeinde
- Portierungen für verschiedene Hardwareplattformen verfügbar (AVR, ARM, MSP430,...)
- Unterstützung für Multithreading und Protothreads
- IPv6-Netzwerkstack implementiert
  - $\sim$ 40 kByte Flash/ROM
  - $\sim$ 2 kByte RAM
  - Entwicklungsumgebung als VM verfügbar: **Instant Contiki**
  - `http://www.contiki-os.org/`

Hardwareplattform

- **Zigbit-Modul**
  - 8-Bit RISC (Atmega 1281)
  - 4 MHz Taktfrequenz
  - 128-KB Flash
  - 8-KB SRAM
  - AT86RF230 Radio Transceiver mit keramischer Chip-Antenne



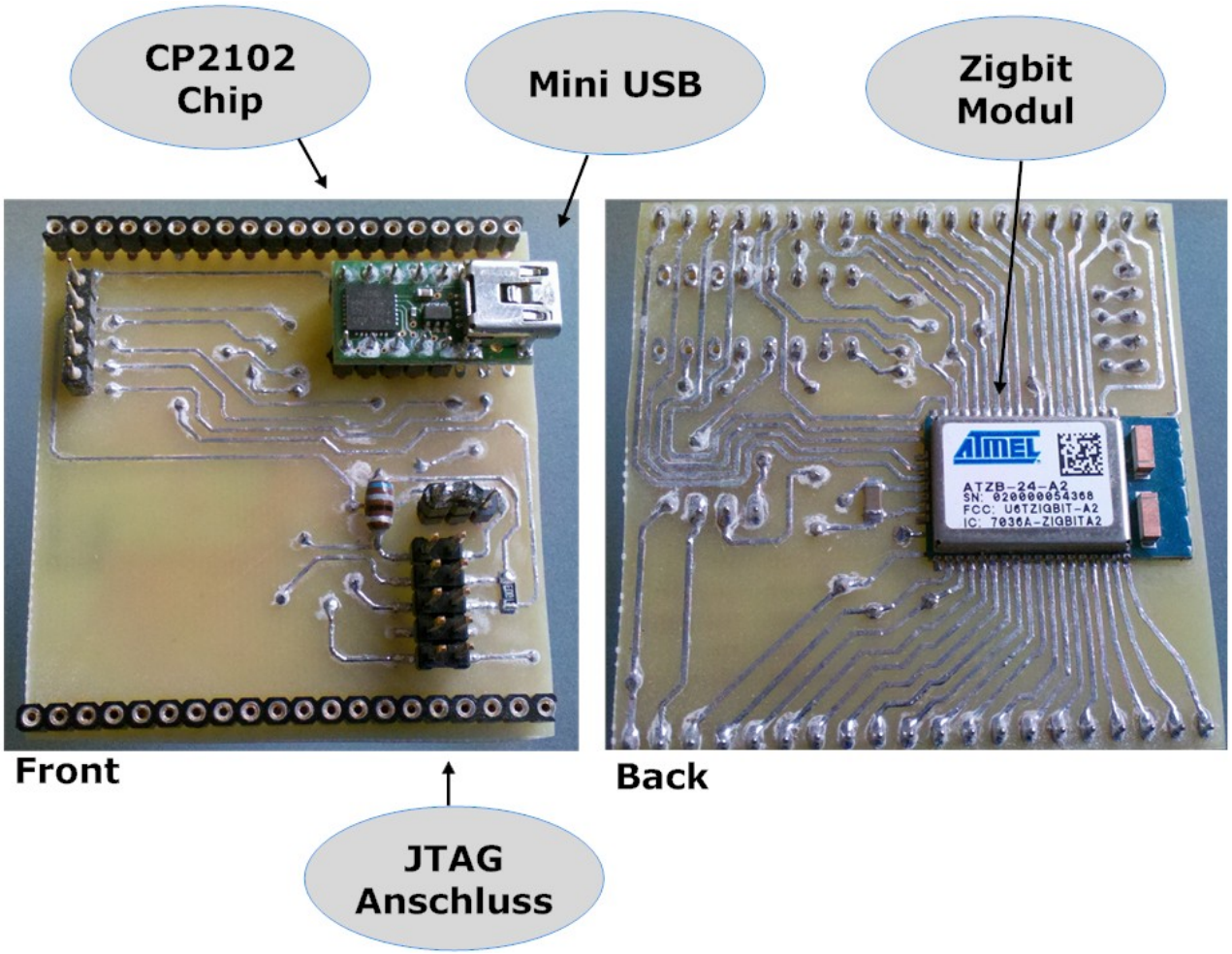
- **Atmel Raven USB-Stick**
  - 8-Bit RISC (Atmega AT90USB1287)
  - 8 MHz Taktfrequenz
  - 128-KB Flash
  - 8-KB SRAM
  - AT86RF230 Radio Transceiver mit PCB Antenne

Ziele und Anforderungen

Entwicklung eines Smart Objects zur Home Automatisierung mit folgenden Eigenschaften:

- IPv6
  - globale Adressierung und Erreichbarkeit
  - Autokonfiguration
- 8-Bit Mikrocontroller
  - kleine Abmessungen (Einbau in vorhandene Geräte)
  - geringer Energiebedarf ($< 1W$)
  - preisgünstig (10-20 Euro)
- Einsatz von standardisierten Werkzeugen und Anwendungen
  - Restful-API für einfache Anwendungsentwicklung mit Web-Technologien

Zigbit Platine

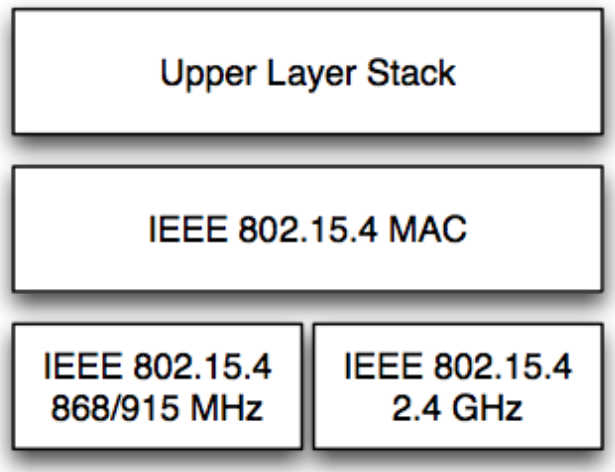


Smart Object : 6LoWPAN Steckdose

IEEE 802.15.4 - Funklayer

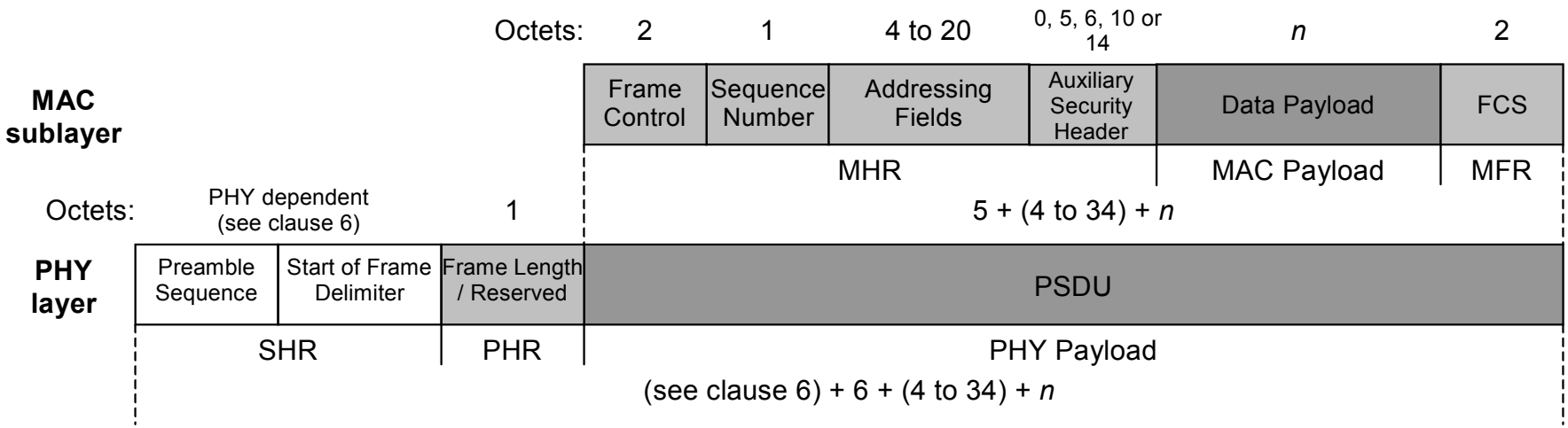
Wichtiger Funkstandard für Home Networks, Industrie- und Gebäudeautomatisierung

- Unterstützt drei Übertragungsverfahren
  - 20 kbit/s bei 868 MHz
  - 40 kbit/s bei 915 MHz
  - 250 kbit/s bei 2.4 GHz (DSSS)
- Beaconless mode
  - Einfacher CSMA Algorithmus
- Beacon mode mit Superframe
  - Hybrider TDMA-CSMA Algorithmus
- Bis zu 64k Netzknoten mit 16-bit Adressen
- Erweiterungen:
  - IEEE 802.15.4a, 802.15.4e, 802.15.5



IEEE 802.15.4 - Funklayer

- Die Größe der IEEE 802.15.4 Frames beträgt 127 Byte.
- Je nach Adressierungsart und aktiver Sicherheitsfeatures können davon 93-118 Bytes als Payload genutzt werden.



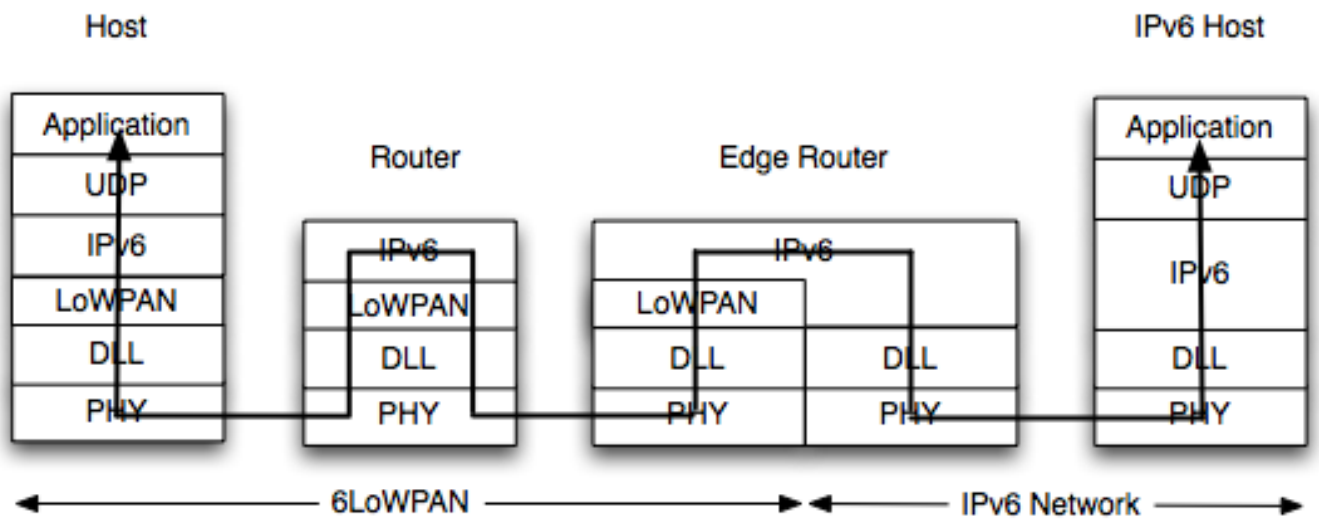
Quelle: IEEE Std 802.15.4 2006

IPv6 over Low-Power Wireless Area Networks - 6LoWPAN

- 802.15.4 maximum Payload beträgt 118 Bytes

- IPv6 definiert eine Path MTU von mindestens 1280 Bytes
- Der einfache IPv6-Header hat eine Länge von 40 Bytes

⇒ Um IPv6 Pakete in IEEE 802.15.4 Frames zu übertragen wird eine Anpassungsschicht benötigt: **6LoWPAN**

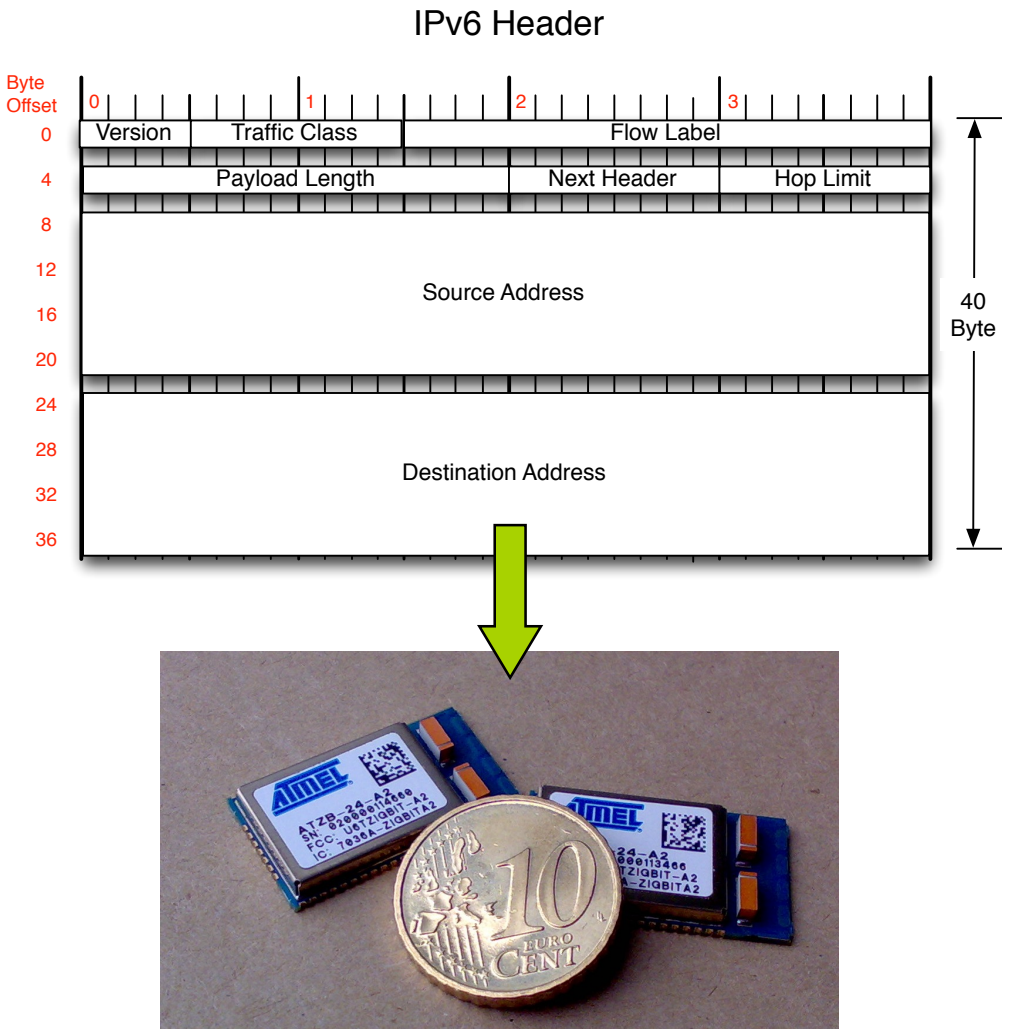


Quelle: http://6lowpan.net

IPv6 over Low-Power Wireless Area Networks - 6LoWPAN

IPv6 Adaptation Layer:

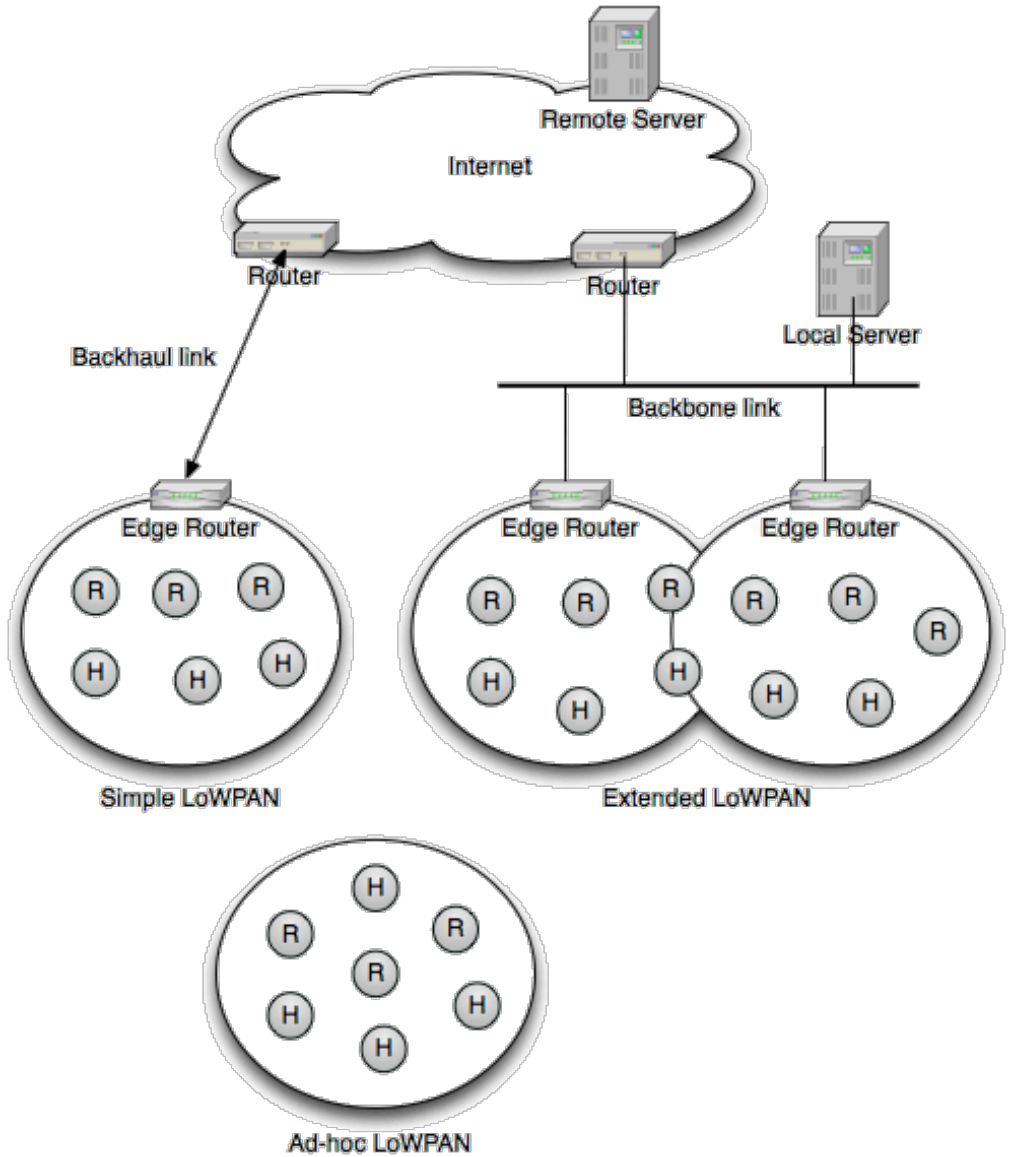
- Standardisiert in RFC 4919, 4944, u.a.
- Stateless Header Compression (IP, UDP)
- Standard Socket API
- Minimaler Code- und Speicherbedarf
- Unterstützung verschiedener Topology Optionen



LoWPAN Netztopologien

LoWPANs sind Stub-Netzwerke:

- Ad-hoc LoWPAN
  - Keine Routen außerhalb des LoWPAN
- Simple LoWPAN
  - Single Edge Router
- Extended LoWPAN
  - Multiple Edge Router mit gemeinsamen Backbone



Quelle: http://6lowpan.net

Routing in Low-Power Lossy Networks (RPL)

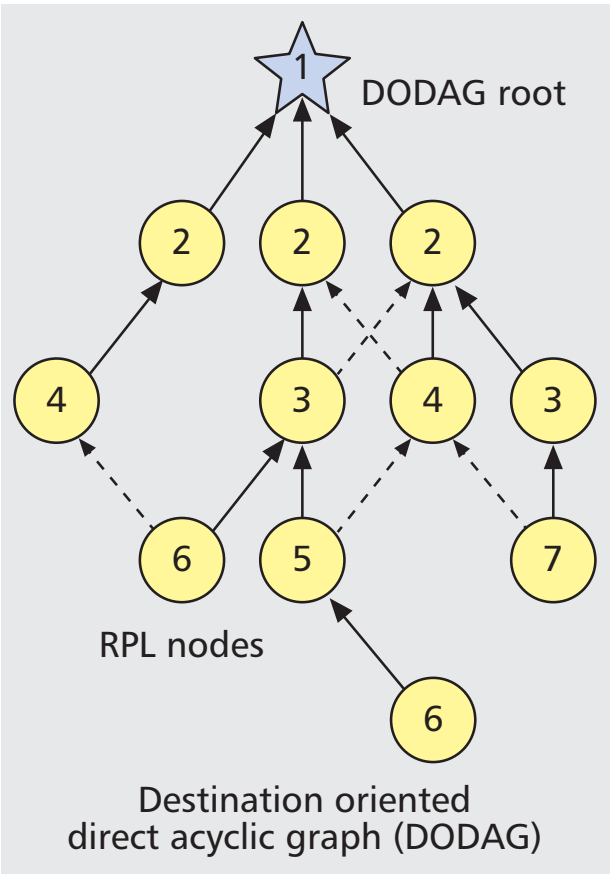
(RFC 6550)

Herausforderungen:

- Begrenzte Ressourcen (RAM, CPU, Batterie)
- Unstabile, verlustbehaftete Links mit geringer Datenrate

RPL Routing:

- Dynamisches Routingprotokoll
- Aufbau eines zielorientierten, gerichteten, azyklischen Graphs (DODAG)
- Minimale Information auf den Knoten
- Forwarding in den meisten Fällen über DODAG Root (Border Router)



Quelle: IEEE Communications Magazine, 04/2011

REST Services im 'Internet of Things'

REST (Representational State Transfer)

- REST ist ein *Architektur Stil* für netzwerkfähige Anwendungen.
- Dabei werden für alle Operationen der Anwendung (create/read/update/delete) ausschließlich HTTP Methoden (GET, POST, PUT, DELETE) verwendet.
- REST ist eine leichtgewichtige Alternative zu Mechanismen wie RPC (Remote Procedure Calls) and Web Services (SOAP, WSDL, etc.).

REST ist kein Standard!

Verschiedene Frameworks unterstützen und implementieren RESTful Services. Da es sich um ein sehr einfaches Konzept handelt, sind komplett eigene Implementierungen möglich.

REST Prinzip – I

Ressourcen:

- werden durch URLs identifiziert
- repräsentieren Zustände und Funktionalität
- lassen sich aus allen Teilen des Systems unmittelbar ansprechen

Ressourcen sind das Schlüsselelement eines RESTful Designs. Die Gesamtheit der Ressourcen in Kombination mit den Zugriffsmethoden definiert die REST API.

Es existieren keine “Methoden” oder “Services” wie in RPC oder SOAP. Es ist in REST nicht notwendig eine bestimmte Methode zu kennen und aufzurufen (z.B. `getProductPrice()`).

REST Prinzip – I

Beispiel: Ein Service macht über folgende URL den Zugriff auf Berichte verschiedener Nutzer möglich:

```
http://example.com/reports/
```

über einen HTTP GET Aufruf lassen sich die Berichte der einzelnen Nutzer aufrufen,

```
http://example.com/reports/alice
```

oder alle vorhandenen Berichte abfragen:

```
http://example.com/reports/all
```

REST Prinzip – II

Client-Server Architektur: Das System funktioniert nach dem Muster der Client-Server Kommunikation. Client-Komponenten können Dienste für weitere Komponenten verfügbar machen.

Ressourcen-Web: Eine einzelne Ressource sollte nur relevante Daten bereitstellen, kann aber Links auf weitere Ressourcen enthalten (so wie bei Webseiten).

Connection state: Die Interaktion ist zustandslos, allerdings können Server und Ressourcen zustandsbehaftet sein. Jeder neue Request beinhaltet alle Daten für seine Ausführung und darf nicht von früheren Interaktionen abhängig sein.

Caching: Ressourcen sollten *cacheable* sein (Angabe eines Ablaufdatums/-zeit). Dafür kann der HTTP *cache-control* Header benutzt werden.

Proxy servers: können Teil der Architektur sein, um Performance und Skalierbarkeit zu erhöhen.

REST (Representational State Transfer)

Request

```
GET /name/meier/alter/43 HTTP/1.1
Host: echo.jsontest.com
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_6_8)
```

Anfragen werden als HTTP GET Request an die spzifizierte Ressource (URL) gesendet.

Response

```
HTTP/1.1 200 OK
Content-Type: application/json; charset=ISO-8859-1
Date: Sun, 01 Dec 2013 19:24:42 GMT

{
   "name": "meier",
   "alter": 43
}
```

Antworten enthalten nur maschinenlesbare Daten, kein HTML. Dabei kommen XML, JSON und CSV Formate zum Einsatz.

REST Eigenschaften

REST wird als Alternative zu Web Services und RPC eingesetzt. Vergleichbar mit Web Services, haben REST Services folgende Eigenschaften:

- Platform-independent (Server läuft unter Unix, Client unter Windows, ...),
- Language-independent (C# kann mit Java sprechen, etc.),
- basiert auf Internet-Standards (HTTP), und
- kann über Firewalls hinweg eingesetzt werden.

REST hat keine eingebauten Sicherheitsfeatures, Verschlüsselung, Session Management, etc.:

- Nutzer-Authentifizierung durch Benutzername/Passwort Token möglich.
- SSL (Secure Socket) kann übertragene Daten verschlüsseln.

JSON (JavaScript Object Notation)

JavaScript Object Notation (JSON)

- Entwickelt durch Douglas Crockford (RFC 4627, Juli 2006)
- Kompaktes, textbasiertes Format zum Datenaustausch zwischen Anwendungen. Für Mensch und Maschine einfach lesbar.
- JSON ist abgeleitet aus Sprachelementen von JavaScript.
- JSON kann vier primitive Datentypen (Strings, Numbers, Booleans, und Null) und zwei strukturierte Typen (Object und Array) darstellen.
- Wird oft in Verbindung mit JavaScript on Demand (JOD), Ajax oder WebSockets zur Übertragung von Daten zwischen Client und Server genutzt.

JSON: Vergleich mit XML

- Einfache Syntax: daher oft lesbarer und insbesondere leichter schreibbar
- reduzierter Overhead im Vergleich zu XML
- JSON-Daten sind im Gegensatz zu XML-Daten typisiert
- XML beschreibt Werte und Eigenschaften potenziell sowohl als Attribute als auch Kindknoten
- XML ist eine Auszeichnungssprache, JSON beschreibt Datenaustauschformate

JSON: Einfache Datentypen

Zahl Folge der Ziffern 0-9, Angabe von Vorzeichen und Exponent möglich

String eingeschlossen in doppelte Anführungszeichen ("). Kann Unicode-Zeichen und Escape-Sequenzen enthalten.

Nullwert dargestellt durch das Schlüsselwort *null*.

boolescher Wert dargestellt durch die Schlüsselwörter *true*, *false*

```
1  {
2      "buch"{
3          "autor": "Douglas Adams",
4          "titel": "The Hitchhiker's Guide to the Galaxy",
5          "publication-date"{
6              "day": 12,
7              "month": 10,
8              "year": 1979
9          }
10     }
11 }
```

JSON: Komplexe Datentypen

Array beginnt mit **[** und endet mit **]**. Enthält eine durch Komma getrennte, **geordnete Liste** von Werten, gleichen oder verschiedenen Typs.

Objekt beginnt mit **{** und endet mit **}**. Enthält eine durch Komma getrennte, **ungeordnete Liste** von Eigenschaften.

Leere Objekte und Arrays sind zulässig.

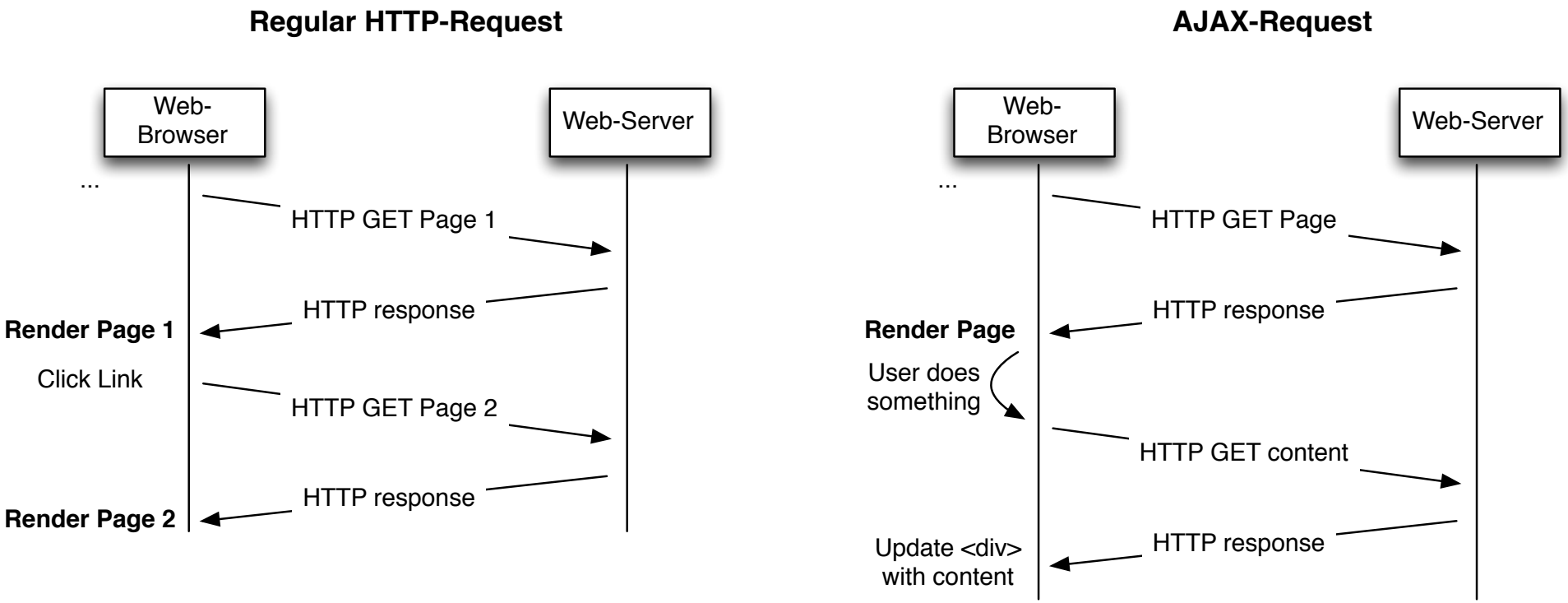
AJAX (Asynchronous JavaScript und XML)

AJAX (Asynchronous JavaScript und XML)

- Buzzword aus dem Web 2.0
- Besteht im wesentlichen aus einem asynchronen HTTP-Request: der Browser führt keinen Page-Refresh durch.
- Der Server sieht einen normalen HTTP-Request.
- Es muss auch kein XML zum Einsatz kommen.
- Implementiert durch das JavaScript `XMLHttpRequest` Objekt:
  - Update eine Webseite ohne die Seite komplett neu zu laden.
  - Daten vom Server anfragen und empfangen nachdem die Seite geladen wurde.
  - Data im Hintergrund an einen Server senden.

Bibliotheken verbergen die minimalen Unterschiede wie AJAX Requests in den verschiedenen Browsern behandelt werden: z.B. Ajax.Request object, Direct Web Remoting (DWR).

AJAX Request



AJAX im Einsatz

AJAX macht interaktive Web-Applikationen möglich:

Desktop-artig	• Drag & Drop • *In-place editing*
Rückmeldung	• Nur Teilbereiche müssen *refreshed* werden • Sofortige Rückmeldung über durchgeführte Operationen • Work-Flow des Nutzers wird nicht unterbrochen, während Daten verarbeitet werden
Benutzbarkeit	• Bessere Rückmeldungen • Verbergen von Wartezeiten • Intuitive Rückwärts/Vorwärts-Semantik • Verbessertes Nutzungsverhalten

Beispiel einer REST Implementierung

REST Implementierungsbeispiel

Implementierung eines einfachen *Internet of Things* Mashup's

jQuery (http://jquery.com/)

- Eine kleine und feature-reiche JavaScript library
- Funktionen zur DOM-Navigation und -Manipulation
- Event handling, Animation und AJAX

Highcharts (http://www.highcharts.com/)

- Interaktive Chart library (JavaScript)
- Freier Einsatz in *non-profit* Projekten
- Vielzahl von Chart-Typen, ansprechendes Design

Contiki RESTful HTTP Server

- https://wiki.ipv6lab.beuth-hochschule.de/contiki/rest-workshop

Contiki

Contiki

The Open Source OS for the Internet of Things

$\Rightarrow$ Contiki ist ein Open Source Betriebssystem für das Internet of Things. Contiki verbindet kleine, günstige, ressourcenarme Geräte mit dem Internet.

Was bietet Contiki?

- Multitasking mit Protothreads (globaler Stack)
- TCP/IP mit IPv6 Support über μIP
- Arbeitet auf 8-Bit Mikrocontrollern
- Benötigt nur wenig kByte RAM
- Contiki ist in Standard-C entwickelt und läuft auf Mikrocontrollern sowie PC-Systemen (Linux, Windows)

Contiki - Minimal-Net

- Für das schnelle Prototyping und erweiterte Debug-Möglichkeiten ist ein erster Einsatz auf einem PC-System empfehlenswert.
- Dafür steht ein vorkonfigurierte VM (Instant Contiki) zur Verfügung: (`http://www.contiki-os.org/start.html`)

  Wichtig:
  - Contiki aktiviert bei IPv6 standardmäßig das Routing Protokoll RPL.
  - Der Contiki-Prozess unter Linux erhält IP-Pakete über ein TAP-Interface (`tap0`).
  - Dieses Interface muss konfiguriert werden.
  - Über ein System von Makefiles wird die Konfiguration von Contiki sowie das Übersetzungsziel gesteuert.

Vorbereitende Maßnahmen I

Installation von Instant Contiki:

- Die Installation von *Instant Contiki* benötigt VMWare Player, oder Virtualbox
- Anmeldung an *Instant Contiki* mit dem Passwort: `user`
- Die Netzwerkverbindung zum Host-PC erfolgt über NAT. Das Ethernet-Interface des Linux Guests wird mit folgendem Kommando konfiguriert:

```
sudo ifconfig eth0 add fdfa::9/64
```

- Die folgende Beschreibung bezieht sich immer auf das Quellcode-Verzeichnis: `/home/user/contiki-2.7/`

Protothreads und Contiki Prozesse

Protothreads

- kooperatives Multitasking
- Kontextwechsel nur an speziell vorgesehenen Programmstellen möglich
- implementiert durch `switch/case`-Routinen (deshalb besser nicht in den eigenen Programmen verwenden!)
- kein eigener Stapelspeicher (thread-local Stack)
- lokale Variablen müssen statisch oder global definiert werden, wenn sie über einen Kontextwechsel hinweg erhalten bleiben sollen!

Contiki Prozesse I

- basierend auf Protothreads
- wie Protothreads basierend auf `switch` Anweisungen, aktivierbar durch Präprozessormakros
- werden dem Betriebssystem mitgeteilt über:
  `PROCESS([prozessname], "[Prozessbeschreibung]");`
- können automatisiert gestartet werden:
  `AUTOSTART_PROCESSES(&[prozessname]);`

Contiki Prozesse II

- Prozessbeginn durch Makro `PROCESS_BEGIN()` (bindet Prozess in globale `switch` Routine ein)
- Ende des Prozesses muss mit `PROCESS_END()` markiert werden (beendet `switch`)
- Der Prozess läuft solange weiter bis ein *blocking macro* erreicht wird oder der Prozess durch `PROCESS_EXIT()` vorzeitig beendet wird.

Contiki Prozesse III - *blocking macros*

- `PROCESS_WAIT_EVENT()`: unterbrich und warte bis eine Event-Nachricht an den Prozess gepostet wird.
- `PROCESS_WAIT_EVENT_UNTIL(cond)`: unterbrich und warte bis eine Event-Nachricht an den Prozess gepostet wird und eine bestimmte Bedingung eingetroffen ist.
- `PROCESS_WAIT_UNTIL(cond)`: warte bis die Bedingung eingetroffen ist. Wenn die Bedingung beim Aufruf schon wahr ist, wird der Prozess nicht unterbrochen.
- `PROCESS_WAIT_WHILE(cond)`: warte solange die Bedingung wahr ist. Wenn die Bedingung beim Aufruf nicht erfüllt ist, wird der Prozess nicht unterbrochen.

Protothreads - Beispiel

```
1  #include "contiki.h"
2  #include <stdio.h>
3
4  PROCESS(pt_example, "Protothread Example Process");
5  AUTOSTART_PROCESSES(&pt_example);
6  PROCESS_THREAD(pt_example, ev, data)
7   {
8     /*Timer statisch deklariert, damit er nach Kontextwechsel noch
9     verfügbar ist*/
10    static struct etimer et;
11
12    PROCESS_BEGIN();
13    etimer_set(&et, (CLOCK_SECOND*20));
14
15    while(1) {
16      /*Warte 20s, solange können andere Protothreads den Prozessor nutzen*/
17      PROCESS_WAIT_EVENT_UNTIL(etimer_expired(&et));
18      etimer_reset(&et);
19      printf("Timer abgelaufen, starte Timer neu...\n");
20    }
21    PROCESS_END();
22  }
```

μIP

μIP I

- μ-Internet Protocol Stack, speziell für ressourcenarme Geräte
- RFC-Standard kompatibel
- Unterstützung von IP, IPv6, ICMP, UDP und TCP
- Schnittstelle für TCP über ProtoSockets (BSD-Socket ähnlich, realisiert über Protothreads)
- Schnittstelle für UDP über die μIP raw API
- Außerdem:

  $\Rightarrow$ einzelner statischer Paketbuffer

  $\Rightarrow$ Prozesse müssen ihre Daten abholen bevor neue Pakete empfangen werden können.

Anwendungsentwicklung unter Contiki OS

Anwendungsentwicklung Contiki OS

- Entwicklungsumgebung Instant Contiki
- VMware Image basierend auf Ubuntu
- enthält alle benötigten Programme, Compiler, Toolchain und Simulatoren.
  $\rightarrow$ Alternativ kann auch eine Cygwin Umgebung auf Windows eingerichtet werden, jedoch das Vorgehen teilweise abweichend.
- Kompilierung mithilfe von `Makefiles`

Anwendungsentwicklung Contiki OS - make I

- Tool der Softwareentwicklung
- für Projekte mit vielen Quellcodedateien
- führt alle benötigten Arbeitsschritte zum Erstellen der ausführbaren Datei durch (übersetzen, linken, kopieren, umbenennen usw...)
- die Anweisungen für `make` werden immer in einem `Makefile` festgelegt.

Anwendungsentwicklung Contiki OS - make II

- Contiki hat mehrere Makefiles, mit unterschiedlichen Aufgaben
- `Makefile.include`, befindet sich im obersten Ordner des Contiki Quellcodes, enthält alle Anweisungen um Contiki OS zu kompilieren.
- `Makefile.[TARGET]`, wobei für TARGET die verwendete Hardware eingetragen ist, z.B. `avr-zigbit` (Makefile.avr-zigbit), befindet sich im Ordner `platform/avr-zigbit`. Enthält alle hardware-spezifischen Anweisungen um Contiki OS für eine bestimmte Zielplattform zu kompilieren.

Anwendungsentwicklung Contiki OS - Contiki Ordnerstruktur I

Der Contiki Quellcode hat eine feste Ordnerstruktur:

- **Contiki Wurzelordner** → oberster Ordner, hier befindet sich z.B. das globale `Makefile.include`
  - **apps** → Hier befinden sich alle Contiki Applikationen, der Quellcode einzelnen Apps kann über das Makefile mit dem Befehl `APPS=[name des App-Ordners]` hinzugefügt werden
  - **core** → In diesem Ordner befindet sich das eigentliche Betriebssystem, u.a auch der Ordner `net`, wo der komplette Quellcode für die Netzwerkkommunikation zu finden ist. Hier ist auch der Ordner `sys`, welcher den Quellcode für das Prozessmanagement und die Timer enthält.
  - **cpu** → Hier findet sich der Quellcode für die unterstützenden Prozessoren, also z.B. den Ordner `avr` für die ATmega Prozessoren oder der Ordner `arm` für ARM Prozessoren. In den jeweiligen Unterordnern sind auch die Treiber für passende Funkchips abgelegt.

Anwendungsentwicklung Contiki OS - Contiki Ordnerstruktur II

- **Contiki Wurzelordner**
  - **doc** → Hier ist einiges an Dokumentation sowie Beispielcode für und über Contiki zu finden.
  - **examples** → Dieser Ordner enthält fertige Contiki Projekte, er könnte auch genauso gut `projects` heissen. Er kann auch zum Erstellen von neuen Projekten genutzt werden.
  - **platform** → enthält alle hardwarespezifischen Einstellungen für die verwendete Plattform.
  - **tools** → Bestimmte Werkzeuge die bei der Arbeit mit Contiki hilfreich sind, z.B. der Simulator `cooja` oder der Quellcode für den SLIP Border Router auf der PC Seite.

Anwendungsentwicklung Contiki OS - der Platform Ordner

Contiki Platform-Ordner - `/platform/[device]`, z.B. `/platform/avr-zigbit/`.
$\Rightarrow$ enthält alle hardwarespezifischen Einstellungen für die verwendete Plattform.

- Datei `contiki.conf`, enthält alle Veränderungen am Contiki Quellcode, welche für die spezielle Hardwareplatform benötigt werden:
  - Einstellungen für μIP, manuelles Ab-/Zuschalten von TCP/UDP und ICMP, Maximas für Neighbors, Verbindungen, ListenPorts uvm...
  - Einstellungen für 6LoWPAN, Algorithmus der Headerkompression
  - Einstellungen für 802.15.4, Funktreiber, MAC-Layer
  - Einstellungen für RPL Routing
  - ...

RESTful Webserver unter Contiki OS

Contiki: minimal-net

Für die Ausführung des Code-Beispiels in *Instant Contiki* müssen einige Änderungen am Quellcode vorgenommen werden:

```
/contiki-2.7/platform/minimal-net/contiki-conf.h
```

- Aktivieren des ULA-Präfixes in Zeile 79:
  `#define HARD_CODED_ADDRESS "fdfd::"`
- Dektivieren des RPL-Routings in Zeile 93:
  `#define UIP_CONF_IPV6_RPL 0`
- Deaktivieren der Router-Funktion in Zeile 94:
  `#define RPL_BORDER_ROUTER 0`
- Vergrößern des Paketpuffers durch Einfügen der Zeile 95:
  `#define UIP_CONF_BUFFER_SIZE 1300`

Contiki: rest-example

Die Contiki REST-Server-Anwendung befindet sich im Verzeichnis

```
/contiki-2.7/examples/rest-example/
```

An dem Makefile müssen folgende Veränderungen vorgenommen werden:

```
/contiki-2.7/examples/rest-example/Makefile
```

- Übersetzungsziel des Makefiles ändern:
  `all: rest-server-example`
- Kein COAP sondern HTTP verwenden:
  `WITH_COAP = 0`
- Compileflags für IPv6 hinzufügen:
  `CFLAGS += -DUIP_CONF_IPV6 = 1`

Contiki: rest-example – Build

Anschließend kann die Anwendung mit dem Befehl

```
make TARGET=minimal-net
```

gebaut werden.

Sollte es notwenig sein nachträglich weitere Änderungen am Makefile durchzuführen, kann das Projekt vor der erneuten Übersetzung zurückgesetzt werden:

```
make TARGET=minimal-net clean
```

Ausführung des REST-Servers und Netzwerk-Anpassung

Die Anwendung wird mit dem folgenden Befehl gestartet:

```
sudo ./rest-server-example.minimal-net
```

Contiki richtet dabei ein neues Netzwerkinterface `tap0` ein. Dieses muss noch konfiguriert werden (Achtung: bei jedem Neustart der Contiki-Anwendung erforderlich):

```
sudo ifconfig tap0 add fdfd::ff:fe00:10/64
```

Danach kann über den Webbrowser auf den Contiki-Service zugegriffen werden:
`http://[fdfd::206:98ff:fe00:232]:8080/helloworld`

REST Example: rest-server-example.c

```
1  RESOURCE(rand, METHOD_GET, "rand");
2
3  /* For each resource defined, there corresponds an handler method.
4   * Name of the handler method should be [resource name]_handler
5   * */
6  void
7  rand_handler(REQUEST* request, RESPONSE* response)
8  {
9    /*Construct a simple JSON Object containing random numbers (0-10)*/
10   sprintf(temp,"{\n\"test\" : \"%d\"\n}", rand() % 11);
11
12   rest_set_header_content_type(response, APPLICATION_JSON);
13   rest_set_response_payload(response, (uint8_t*)temp, strlen(temp));
14 }
15 ...
16
17 PROCESS_THREAD(rest_server_example, ev, data)
18 {
19 ...
20   rest_activate_resource(&resource_rand);
21 }
```

RESTful Service auf Contiki

Request

```
GET /rand HTTP/1.1
Host: [FDFD::206:98FF:FE00:232]
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_9_2)
```

Abfrage einer Zufallszahl durch HTTP GET Request an die URL-spezifizierte Ressource.

Response

```
HTTP/1.1 200 OK
Access-Control-Allow-Origin: *
Server: Contiki
Content-Type: application/json

{ "test":  9 }
```

Der Service antwortet mit JSON-kodierter Pseudo-Zufallszahl.

RESTful Mashup

- Erstellung von Rest-Ressourcen direkt als C-Funktionen (siehe Beispiel):

```
/examples/rest-example/rest-server-example.c
```

- Aufruf der Ressource `rand` des REST Webservers aus der JavaScript-Anwendung:

```
var test = 0;
var jsonURL = "http://[FDFD::206:98FF:FE00:232]:8080/rand";
jQuery.getJSON(jsonURL,function(data){
                        test = Number(data.test);
});
```

- Ausgabe der Abfragewerte des Contiki REST Webservers:

```
1 ...
2    alert("Zufallszahl: " + test);
3 ...
```

RESTful Mashup

Es gibt ein Problem!

- Scripte werden nicht direkt vom Mikrocontroller geladen, sie sind lokal gespeichert → Browser verhindert die Anzeige der JSON-Daten.
- Moderne Webbrowser unterstützen Cross-Origin Resource Sharing (CORS): http://www.w3.org/TR/2014/REC-cors-20140116/
- Dies erfordert einen zusätzlichen HTTP-Header: `Access-Control-Allow-Origin`

```
HTTP/1.1 200 OK
Access-Control-Allow-Origin: *
```

Anpassung des Rest-Webservers

Es wird eine neue Funktion zur Datei `rest.c` hinzugefügt:

/apps/rest-common/rest.c

```
1  int
2  rest_set_header_access_control_allow_origin(RESPONSE* response,
3                          char* access_string)
4  {
5  #ifdef WITH_COAP
6    //return coap_set_header_content_type(response, content_type);
7  #else
8    return http_set_res_header(response,
9                          HTTP_HEADER_NAME_ACCESS_CONTROL_ALLOW_ORIGIN,
10                         access_string, 1);
11 #endif /*WITH_COAP*/
12 }
```

RESTful Mashup

Zur Datei `http-common.c`:

```
/apps/rest-http/http-common.c
```

wird die folgende Variable hinzugefügt:

```
1 const char* HTTP_HEADER_NAME_ACCESS_CONTROL_ALLOW_ORIGIN =
2             "Access-Control-Allow-Origin";
```

Die relevanten Header-Dateien müssen angepasst werden, dabei kann der HTTP Port geändert werden (default: 8080):

```
/apps/rest-http/http-common.h
```

- `#define HTTP_PORT 80`

RESTful Mashup

Zur Funktion `rand_handler()` in der Datei:

```
/examples/rest-example/rest-server-example.c
```

wird folgender Funktionsaufruf hinzugefügt:

```
1 void
2 rand_handler(REQUEST* request, RESPONSE* response)
3 {
4 ...
5     rest_set_header_access_control_allow_origin(response, "*");
6 ...
7 }
```

Fazit

Pro:

- *Smart Objects* mit IPv6 stellen das Ende-zu-Ende Prinzip des Internets in Sensor-Netzen her
- Entwicklungsaufwand ist durch Nachnutzung bestehender Bibliotheken überschaubar (Entwicklung in C)
- Leistungsfähigkeit für viele Anwendungsfelder ausreichend
- Unterstützung bewährter APIs erlaubt einfache Dienstentwicklung

Con:

- Debugging auf dem Mikrocontroller ist schwierig (Trial-und-Error, Ausweichen auf Simulationsumgebung)
- Aufbau realer Netze derzeit noch konfigurationsintensiv

... zukünftige, mögliche Erweiterungen

- Implementierung von *over-the-air-updates* aus dunkels06runtime.pdf (Softwareupdate über Funk)
- Sicherheit von Smart Objects in öffentlichen Netzen ist ein interessantes Thema (Hacking, Denial-of-Service, etc.)
- Unterstützung von realen Deployment-Szenarien (Adressvergabe, Konfiguration, Prefix-Delegation, etc.)
- Entwicklung einer effizienten Spannungsversorgung: *Energieharvesting, Duty-Cycling*

... zukünftige, mögliche Erweiterungen

Wir suchen Forschungspartner aus der Industrie für die Weiterentwicklung und den praktische Einsatz und Test von IPv6-fähigen SmartObjects!

- BMBF, ZIM
- Studentenprojekte
- Praktikas
- Abschlussarbeiten

Kontakt

Email: scheffler@beuth-hochschule.de

WWW: http://prof.beuth-hochschule.de/scheffler/

Folien & Code: https://wiki.ipv6lab.beuth-hochschule.de/contiki/rest-workshop

BEUTH HOCHSCHULE
FÜR TECHNIK
BERLIN
University of Applied Sciences