Contiki OS and IoT

Sam Nguyen-Xuan
Faculty of Information Technology No. 2
Posts and Telecoms of Institute and Technology
Ho Chi Minh City Campus, Vietnam

Source: EEEM048/COM3023- Internet of Things/Dr Payam Barnaghi, Dr Chuan H Foh

Contiki overview

- Contiki is an open source operating system for the Internet of Things.
  - runs on networked embedded systems and wireless sensor nodes.
- It is designed for microcontrollers with small amounts of memory.
- A typical Contiki configuration is 2 kilobytes of RAM and 40 kilobytes of ROM.
- Contiki provides IP communication, both for IPv4 and IPv6.
  - It has an IPv6 stack that, combined with power-efficient radio mechanisms such as ContikiMAC, allow battery-operated devices to participate in IPv6 networking.
  - Contiki supports 6lowPAN header compression and the CoAP application layer protocol.
    - We will study 6LowPAN and CoAP protocols later in this module.

Source: http://www.contiki-os.org

Contiki- Functional aspects

- It's kernel functions as an event-driven kernel; multithreading is supported by an application library. In this sense it is a hybrid OS (monolithic and microkernel).
- Contiki realises the separation of concern of the basic system support form the rest of the dynamically loadable and programmable services (called processes).
- The services communicate with each other through the kernel by posting events.
- The ContikiOS kernel does not provide any hardware abstraction; but it allows device drivers and application directly communicate with the hardware.
- Each Contiki service manages its own state in a private memory space and the kernel keeps a pointer to the process state.

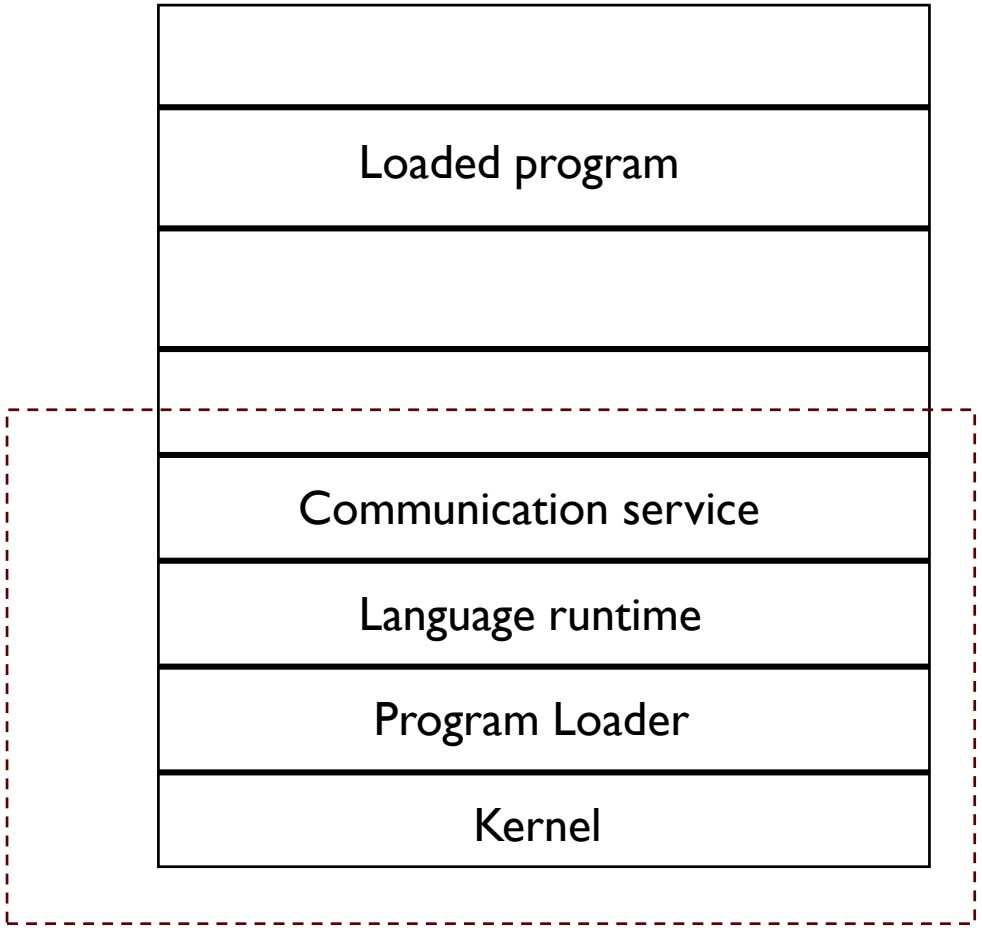
The Contiki system

- A Contiki system includes the kernel, libraries and applications or services, of which applications and services are implemented as modules.
- All communication between processes are done through the kernel.
- The ContikiOS kernel does not provide any hardware abstraction. As a result of this, device drivers and applications communicate directly with the hardware.
- All processes share the same address and space. This enables Contiki to run in memory constrained devices.

The Contiki system

- A process runs when an event related to the process occurs, such as a timer event or an external one.
- There are two types of execution modes in Contiki. These are cooperative and preemptive modes.
- A process is defined by an event handler function and an optional poll handler function.
- The Contiki process is a combination of two different parts. These are process control block (The block is defined via the process macro) and the process threat.
  - The process control block is composed of information about each process such as the state of the process, the pointer to the next process, name of the process, a pointer to a process thread, the state of the proto thread and internal flags.

The Contiki OS



Contiki's kernel architecture

- Contiki is based on a modular kernel architecture. Therefore, the kernel is minimal.
- In Contiki, all program execution is triggered either through the polling mechanism or by events which are sent by the kernel.
  - the kernel supports two types of events, synchronous and asynchronous.
  - the polling mechanism behaves as high priority events that are scheduled between asynchronous events.
- Contiki schedules all events using a single level hierarchy, and events cannot be preempted by other events. The only way to preempt an event is using interrupts.

Contiki's kernel architecture

- When a program is loaded, the loader uses the relocation information provided by the binary format to allocate memory. In case that there is not enough memory, the loading is aborted.
- In case of successful loading, the function for initialization is called for starting or replacing other processes.

Protothreads

- Protothreads can be seen as lightweight (stakless) threads.
- They can be also seen as interruptible tasks in event-based programming.
- A protothread provides a conditional blocking "wait" statement which takes a conditional statement and blocks the protothread until the statement is evaluated true.
- By the time the protothread reaches the wait time if the conditional statement is true, it continues executing without any interruption.
- A protothread is invoked whenever a process receives a message from another process or a timer event.

Protothreads- example

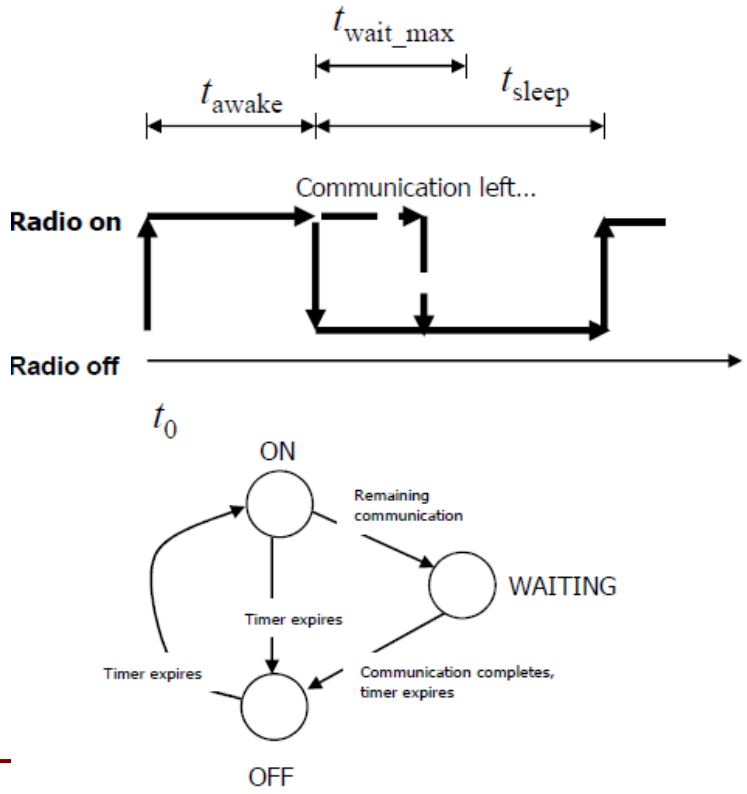
- For example consider a MAC protocol that turns off the radio subsystem on a periodic basis; but you want to make sure that the radio subsystem completes the communication before it goes to sleep state.

1. *At $t=t_0$ set the radio ON*
2. *The radio remains on for a period of t_{awake} seconds*
3. *Once t_{awake} is over, the radio has to be switched off, but any on-going communication needs to be completed.*
4. *If there is an on-going communication, the MAC protocol will wait for a period, t_{wait_max} before switching off the radio.*
5. *If the communication is completed or the maximum wait time is over, then the radio will go off and will remain in the off state for a period of t_{sleep}.*
6. *The process is repeated.*

Source: Adam Dunkels, Oliver Schmidt, Thiemo Voigt, and Muneeb Ali. 2006. Protothreads: simplifying event-driven programming of memory-constrained embedded systems. In Proceedings of the 4th international conference on Embedded networked sensor systems (SenSys '06). ACM.

Radio sleep cycle code with events

Event driven code can be messy and complex

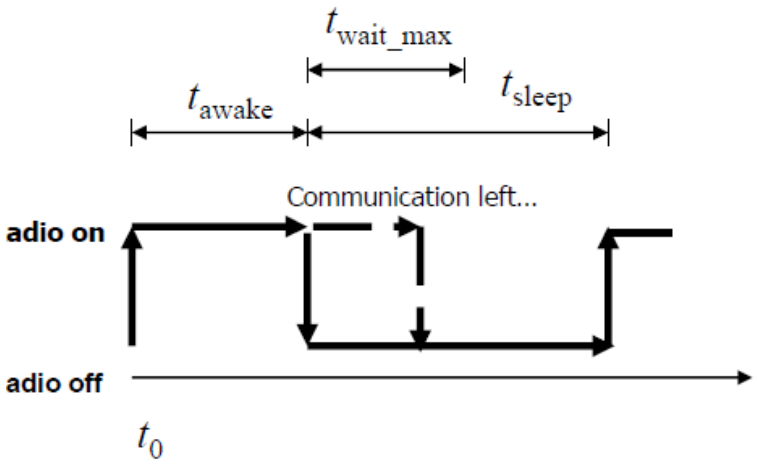



```
enum {ON, WAITING, OFF} state;

void eventhandler() {
  if(state == ON) {
    if(expired(timer)) {
      timer = t_sleep;
      if(!comm_complete()) {
        state = WAITING;
        wait_timer = t_wait_max;
      } else {
        radio_off();
        state = OFF;
      }
    }
  } else if(state == WAITING) {
    if(comm_complete() ||
       expired(wait_timer)) {
      state = OFF;
      radio_off();
    }
  } else if(state == OFF) {
    if(expired(timer)) {
      radio_on();
      state = ON;
      timer = t_awake;
    }
  }
}
```

Source: Adam Dunkels, Oliver Schmidt, Thiemo Voigt, and Muneeb Ali. 2006. Protothreads: simplifying event-driven programming of memory-constrained embedded systems. In Proceedings of the 4th international conference on Embedded networked sensor systems (SenSys '06). ACM.

Radio sleep cycle with Protothreads




```
int protothread(struct pt *pt) {
  PT_BEGIN(pt);
  while(1) {
    radio_on();
    timer = t_awake;
    PT_WAIT_UNTIL(pt, expired(timer));
    timer = t_sleep;
    if(!comm_complete()) {
      wait_timer = t_wait_max;
      PT_WAIT_UNTIL(pt, comm_complete()
                || expired(wait_timer));
    }
    radio_off();
    PT_WAIT_UNTIL(pt, expired(timer));
  }
  PT_END(pt);
}
```

Source: Adam Dunkels, Oliver Schmidt, Thiemo Voigt, and Muneeb Ali. 2006. Protothreads: simplifying event-driven programming of memory-constrained embedded systems. In Proceedings of the 4th international conference on Embedded networked sensor systems (SenSys '06). ACM.

Sensor Network Programming

- Sensor Network programming can be: node centric or it can be application centric.
- Node-centric approaches focus on development of a software for nodes (on a per-node level).
- Application-centric approaches focus on developing software for a part or all of the network as one entity.
- The application centric programming will require collaboration among different nodes in the network for collection, dissemination, analysis and/or processing of the generated and collected data.
- While in node centric programming the main focus is on developing a software on a per-node level.

The Contiki code

```
#include "contiki.h"

PROCESS(sample_process, "My sample process");

AUTOSTART_PROCESSES(&sample_process);

PROCESS_THREAD(sample_process, ev, data) {
      PROCESS_BEGIN();
          while(1) {
                        PROCESS_WAIT_EVENT();
          }
      PROCESS_END();

}
```

Header files

Defines the name of the process

Defines the process will be started every time module is loaded

contains the process code

Event parameter; process can respond to events

Threads must have an end statement

process can receive data during an event

The Contiki code

```
#include "contiki.h"
PROCESS(sample_process, "My sample process");

AUTOSTART_PROCESSES(&sample_process, &LED_process);

PROCESS_THREAD(sample_process, ev, data) {
        static struct etimer t;
        static int c = 0;
         PROCESS_BEGIN();
                  etimer_set(&t, CLOCK_CONF_SECOND);
                 while(1) {
                                  PROCESS_WAIT_EVENT();
                                  if(ev == PROCESS_EVENT_TIMER)  {
                                                   printf("Timer event #%i\n", c);
                                                   c++;
                                                   etimer_reset(&t);
                                  }
                 }
         PROCESS_END();
}

PROCESS_THREAD(LED_process, ev, data) {
        static uint8_t leds_state = 0;
        PROCESS_BEGIN();
                  leds_off(0xFF);
                  leds_on(leds_state);
        PROCESS_END();
}
```

Process thread names

Process thread 1

Process thread 2

Typedefs

typedef uint8_t	**u8_t** The 8-bit unsigned data type.
typedef uint16_t	**u16_t** The 16-bit unsigned data type.
typedef uint32_t	**u32_t** The 32-bit unsigned data type.
typedef int32_t	**s32_t** The 32-bit signed data type.
typedef unsigned short	**uip_stats_t** The statistics data type.

Running Contiki on a Hardware

- Write your code
- Compile Contiki and the application
  - make TARGET=XM1000 sample_process
  - Ma

```
CONTIKI = ../..
all: simple_process
include $(CONTIKI)/Makefile.include
```

- If you plan to compile your code on the chosen platfrom more than once;
  - make TARGE=XM1000 savetarget
- Upload your code
  - make simple_process.upload
- Login to the device
  - make login

Nodes and Applications in Wireless Sensor Networks

- Sensor Networks consist of nodes with different capabilities.
  - Large number of heterogeneous sensor nodes
  - Spread over a physical location
  - It includes physical sensing, data processing and networking
- In ad-hoc networks, sensors can join and leave due to mobility, failure etc.
- Data can be processed in-network, or it can be directly communicated to the endpoints.

Types of nodes

- Sensor nodes
  - Low power
  - Consist of sensing device, memory, processor and radio
  - Resource-constrained
- Sink nodes
  - Another sensor node or a different wireless node
  - Normally more powerful/better resources
- Gateway
  - A more powerful node
  - Connection to core network

Types of applications

- Event detection
  - Reporting occurrences of events
  - Reporting abnormalities and changes
  - Could require collaboration of other nearby or remote nodes
  - Event definition and classification is an issue
- Periodic measurements
  - Sensors periodically measure and report the observation and measurement data
  - Reporting period is application dependent
- Approximation and pattern detection
  - Sending messages along the boundaries of patterns in both space/time
- Tracking
  - When the source of an event is mobile
  - Sending event updates with location information

Requirements and challenges

- Fault tolerance
  - The nodes can get damaged, run out of power, the wireless communication between two nodes can be interrupted, etc.
  - To tolerate node failures, redundant deployments can be necessary.
- Lifetime
  - The nodes could have a limited energy supply;
  - Sometimes replacing the energy sources is not practical (e.g. underwater deployment, large/remote field deployments).
  - Energy efficient operation can be a necessity

Requirements and challenges – Cont'd

- Scalability
  - A WSN can consists of a large number of nodes
  - The employed architectures and protocols should scale to these numbers.
- Wide range of densities
  - Density of the network can vary
  - Different applications can have different node densities
  - Density does not need to be homogeneous in the entire network and network should adapt to such variations.

Requirements and challenges – Cont'd

- Programmability
  - Nodes should be flexible and their tasks could change
  - The programmes should be also changeable during operation.
- Maintainability
  - WSN and environment of a WSN can change;
  - The system should be adaptable to the changes.
  - The operational parameters can change to choose different trade-offs (e.g. to provide lower quality when energy efficiency is more important)

Required mechanisms

- Multi-hop wireless communications
  - Communication over long distances can require intermediary nodes as relay (instead of using high transmission power for long range communications).
- Energy-efficient operation
  - To support long lifetime
  - Energy efficient communication/dissemination of information
  - Energy efficient determination of a requested information
- Auto-configuration
  - Self-xxx functionalities

Required mechanisms

- Collaboration and in-network processing
  - In some applications a single sensor node is not able to handle the given task or provide the requested information.
  - Instead of sending the information form various source to an external network/node, the information can be processed in the network itself.
    - e.g. data aggregation, summarisation and then propagating the processed data with reduced size (hence improving energy efficiency by reducing the amount of data to be transmitted).
- Data-centric
  - Conventional networks often focus on sending data between two specific nodes each equipped with an address.
  - Here what is important is data and the observations and measurements not the node that provides it.

Communication and Network Protocol Support

Communication Protocols

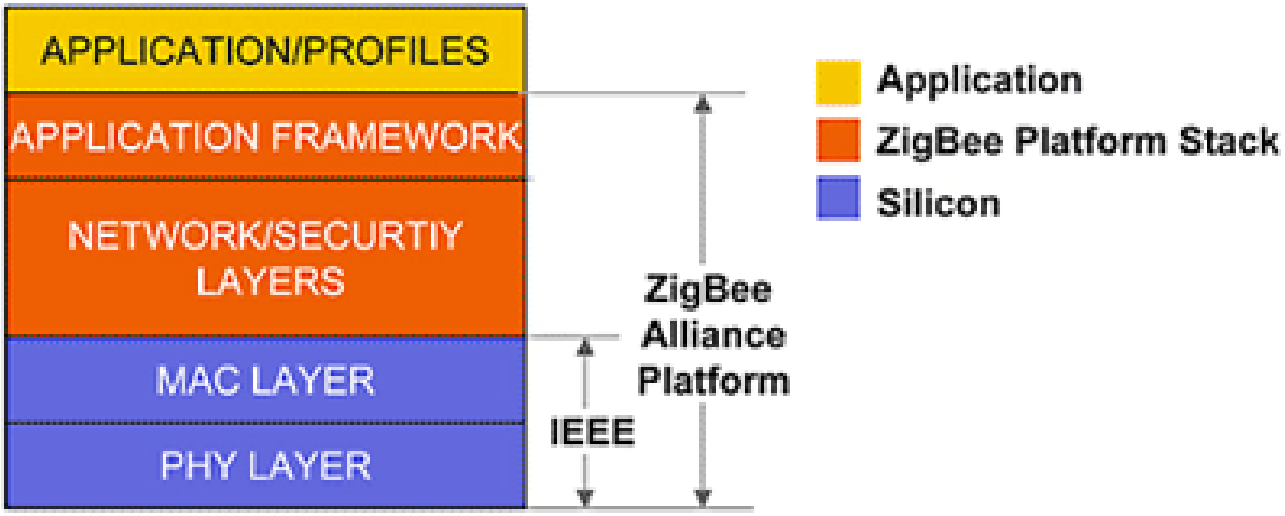
- Wired
  - USB, Ethernet
- Wireless
  - Wifi, Bluetooth, ZigBee, IEEE 802.15.x
- Single-hop or multi-hop
  - Sink nodes, cluster heads…
- Point-to-Point or Point-to-Multi Point
- (Energy) efficient routing

ZigBee

- It is supposed to be a low cost, low power mesh network protocol.
- ZigBee operation range is in the industrial, scientific and medical radio bands;
- ZigBee's physical layer and media access control defined in defined based on the IEEE 802.15.4 standard.
- ZigBee nodes can go from sleep to active mode in 30 ms or less, the latency can be low and in result the devices can be responsive, in particular compared to Bluetooth devices that wake-up time can be longer (typically around three seconds).

[source: Gary Legg, ZigBee: Wireless Technology for Low-Power Sensor Networks, http://www.eetimes.com/document.asp?doc_id=1275760]

ZigBee



BAND	COVERAGE	DATA RATE	# OF CHANNEL(S)
2.4 GHz ISM	Worldwide	250 kbps	16
868 MHz	Europe	20 kbps	1
915 MHz ISM	Americas	40 kbps	10

[source: Gary Legg, ZigBee: Wireless Technology for Low-Power Sensor Networks, http://www.eetimes.com/document.asp?doc_id=1275760]

Network protocols

- The network (or OSI Layer 3 abstraction) provides an abstraction of the physical world.
- Communication protocols
  - Most of the IP-based communications are based on the IPV.4 (and often via gateway middleware solutions)
  - IP overhead makes it inefficient for embedded devices with low bit rate and constrained power.
  - However, IPv6.0 is increasingly being introduced for embedded devices
    - 6LowPAN

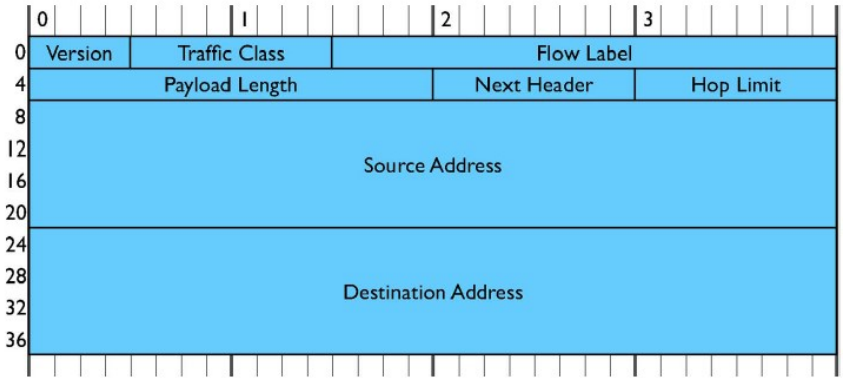
IPv6 over Low power Wireless Personal Area Networks (6LowPAN)

- 6LoWPAN typically includes devices that work together to connect the physical environment to real-world applications, e.g., wireless sensors.
- Small packet size
  - the maximum physical layer packet is 127 bytes
  - 81 octets (81 * 8 bits) for data packets.
- Header compression
- Fragmentation and reassembly
  - 6LoWPAN defines a header encoding to support fragmentation when IPv6 datagrams do not fit within a single frame and compresses IPv6 headers to reduce header overhead.
- Support for both 16-bit short or IEEE 64-bit extended media access control addresses.
- Low bandwidth
  - Data rates of 250 kbps, 40 kbps, and 20 kbps for each of the currently defined physical layers (2.4 GHz, 915 MHz, and 868 MHz, respectively).

Source: Jonathan W. Hui and David E. Culler, IPv6 in Low-Power Wireless Networks, Proceedings of the IEEE (Volume:98 , Issue: 11).

6LowPAN

- IPv6 requires the link to carry a payload of up to 1280 Bytes.
- Low-power radio links often do not support such a large payload - IEEE 802.15.4 frame only supports 127 Bytes of payload and around 80 B in the worst case (with extended addressing and full security information).
- the IPv6 base header, as at 40 Bytes.

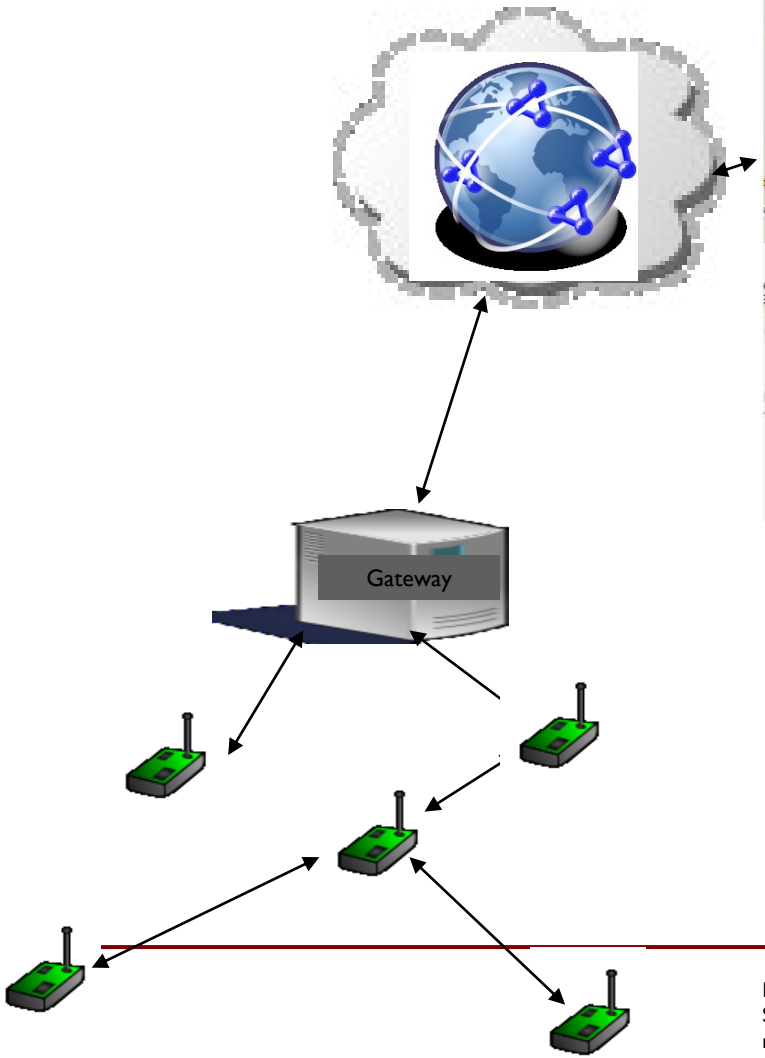


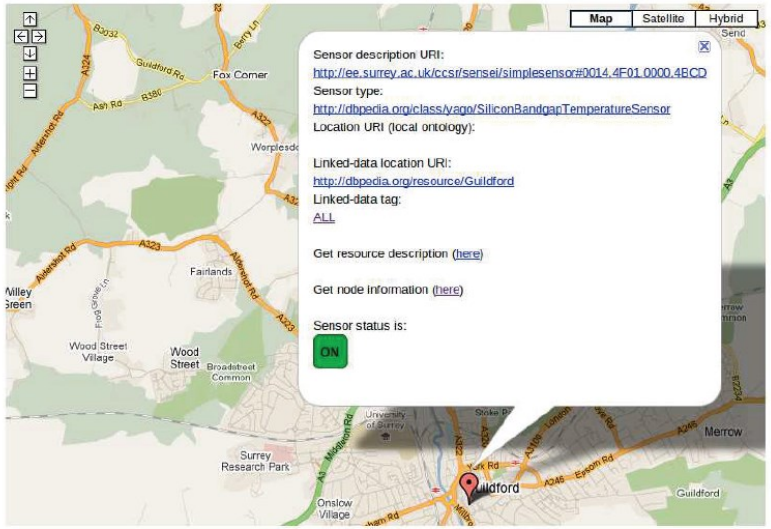
Source: Jonathan W. Hui and David E. Culler, IPv6 in Low-Power Wireless Networks, Proceedings of the IEEE (Volume:98 , Issue: 11).

Using gateway and middleware

- It is unlikely that everything will be IP enabled and/or will run IP protocol stack
- Gateway and middleware solutions can interfaces between low-level sensor island protocols and IP-based networks.
- The gateway can also provide other components such as QoS support, caching, mechanisms to address heterogeneity and interoperability issues.

Gateway and IP networks





Discovered Nodes

0014.4F01.0000.53F6
0014.4F01.0000.4BCD

Device and Context Information:
Sensor ID: 0014.4F01.0000.4BCD
Sensor Type: SunSpot
Status: On
Battery Information: 2
Link Strength: -6
Gateway: g1
Location: Guildford
Similar Sensors: 0014.4F01.0000.4000, 0014.4F01.0000
Sensors in same area: 0014.4F01.0000.4000

Capabilities
Temperature Sensor: 27.5
Light Sensor
52
Accelrometer
-0.0
-0.0
0.91

Frieder Ganz, Payam Barnaghi, Francois Carrez and Klaus Moessner, "Context-aware Management for Sensor Networks", in the Fifth International Conference on COMmunication System softWAre and middlewaRE (COMSWARE11), July 2011.

Service interfaces to WSN

- Supporting high-level request/response interactions
- Asynchronous event notifications
- Identifying and accessing data
  - By location, by observed entity,
    - By semantically meaningful representations – "Room 35BA01"
- Accessibility of in-network processing functions
- Accessing node/network status information (e.g., battery level)
  - Security, management functionality, …
- There are emerging solutions and standards in this domain supported by Semantic Web technologies and Linked-data (we study some of these next week).

Service interfaces and Web connectivity

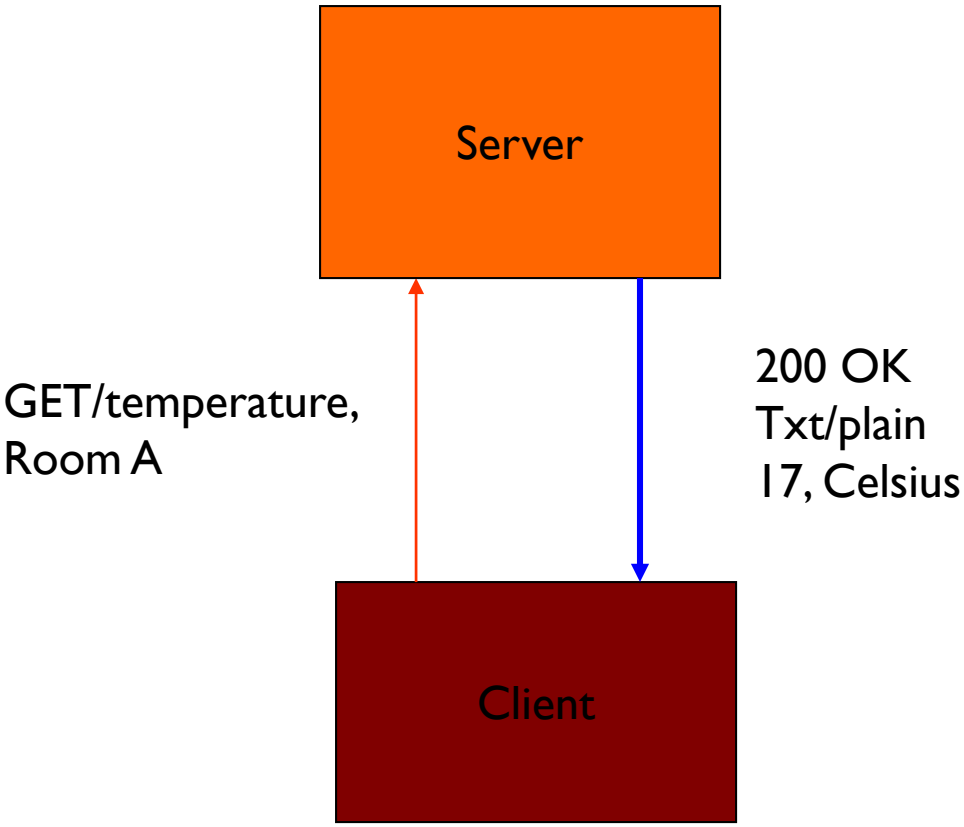
- WSN nodes are typically resource constrained
  - Memory and process limitations
  - Communication load
- Often none-IP or use 6LowPAN
- Using gateway and middleware is a clear solution
- Or can the nodes directly connect to the Web and or support service interfaces?

Constrained Application Protocol (CoAP)

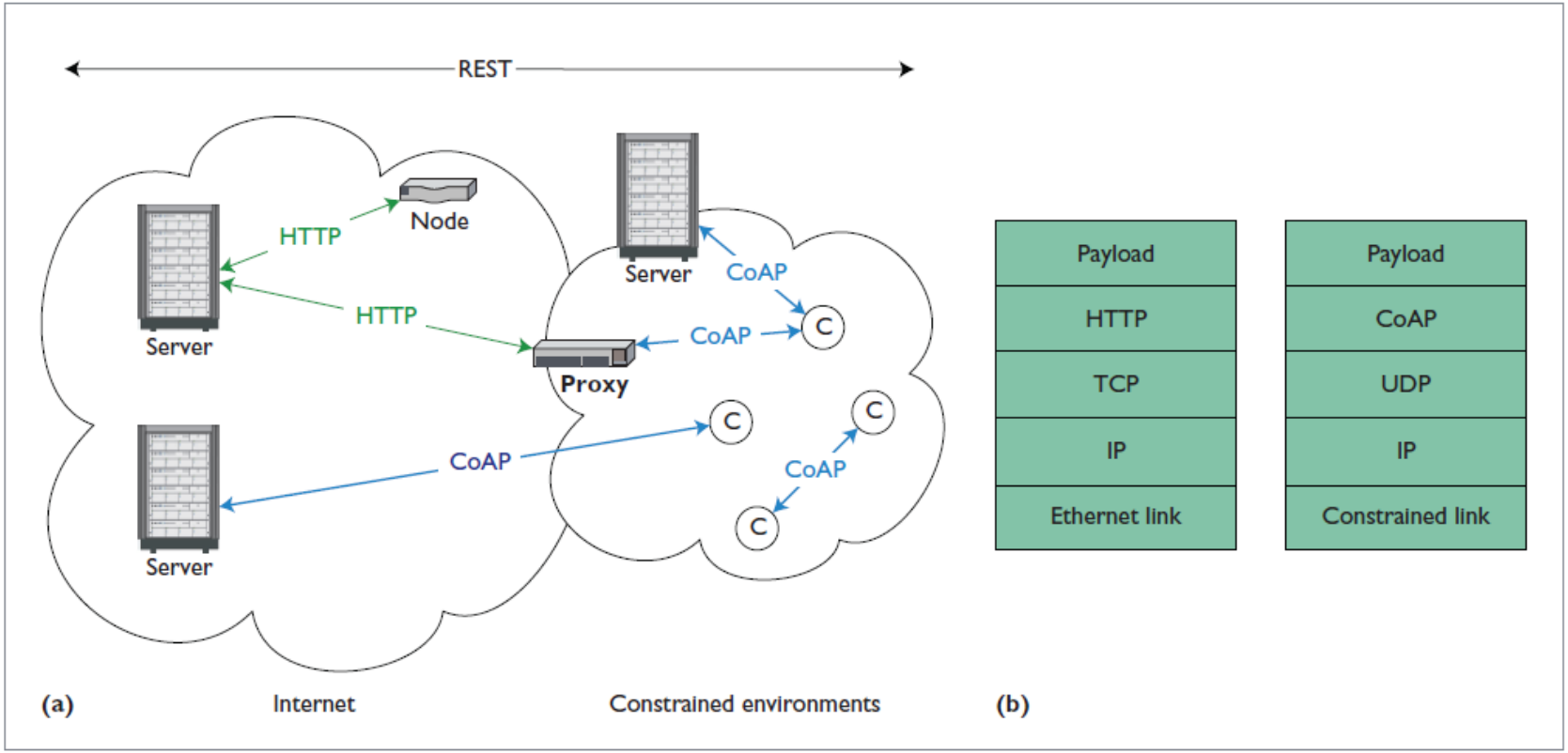
- CoAP is a transfer protocol for constrained nodes and networks.
- CoAP uses the Representational State Transfer (REST) architecture.
  - REST make information available as resources that are identified by URIs.
  - Applications communication by exchanging representation of these resources using a transfer protocol such as HTTP.
  - Clients access servicer controlled resources using synchronous request/response mechanisms.
    - Such as GET, PUT, POST and DELETE.
  - CoAp uses UDP instead of TCP and has a simple "message layer" for re-transmitting lost packets.
  - It also uses compression techniques.

C. Bormann, A. P. Castellani, Z. Shelby, "CoAP: An Application Protocol for Billions of Tiny Internet Nodes," IEEE Internet Computing, vol. 16, no. 2, pp. 62-67, Feb. 2012, doi:10.1109/MIC.2012.29

Constrained Application Protocol (CoAP)



CoAP protocol stack and interactions



Implementing the Web architecture with HTTP and the Constrained Application Protocol (CoAP). (a) HTTP and CoAP work together across constrained and traditional Internet environments; (b) the CoAP protocol stack is similar to, but less complex than, the HTTP protocol stack.

C. Bormann, A. P. Castellani, Z. Shelby, "CoAP: An Application Protocol for Billions of Tiny Internet Nodes," IEEE Internet Computing, vol. 16, no. 2, pp. 62-67, Feb. 2012, doi:10.1109/MIC.2012.29