Lightweight, Low-Power IP

Adam Dunkels, PhD

Swedish Institute of Computer Science

adam@sics.se

The Message

- IP is lightweight
  - … but weight has performance implications
- IP is small
  - … but means API changes
- IP is power-efficient
  - … but low power subtly affect the system

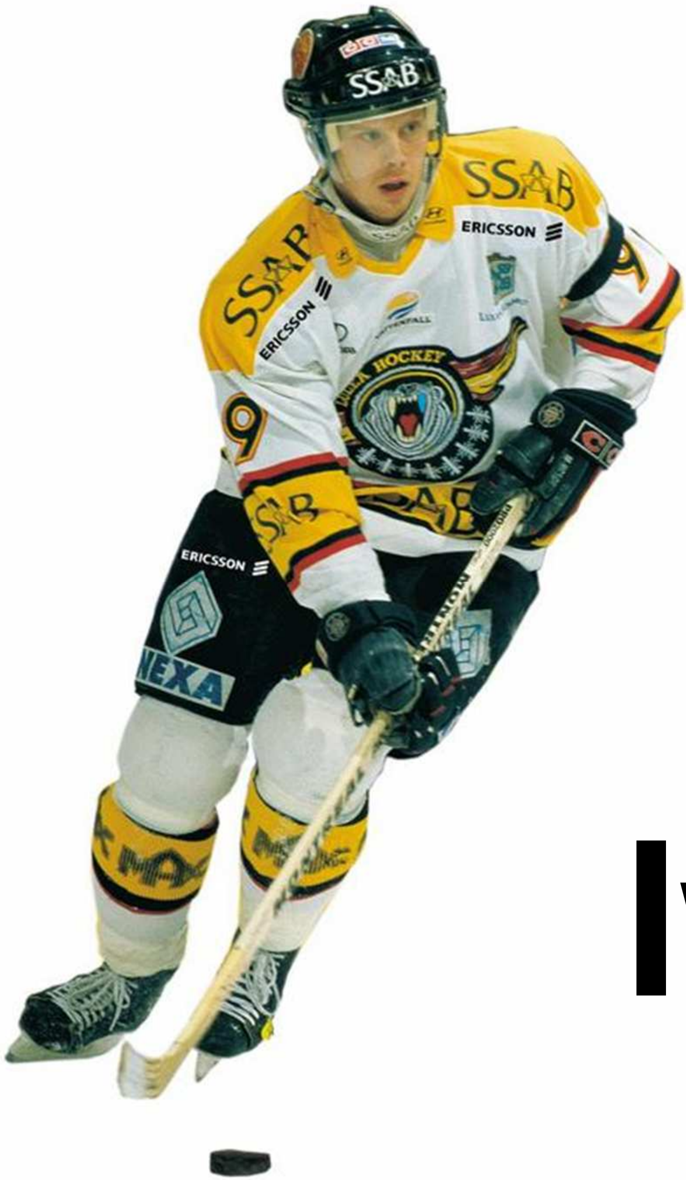
id -G -n

lwIP

uIP

Adam Dunkels
Swedish Institute of Computer Science
adam@sics.se
http://www.sics.se/~adam/

Contiki



lwIP

Slashdot NEWS FOR NERDS. STUFF THAT MATTERS.

Log In | Create Account | Help | Subscribe | Firehose

▼ Sections

Main
Apple
AskSlashdot
Books
Developers

TCP/IP Enabled Lego Brick

Posted by timothy on Tue Jan 29, 2002 05:12 PM
from the **snap-snap-ping** dept.

An anonymous reader submits:

"Yesterday, Olaf Christ announced that he has the world's
brick that can be used as a web server. Imagine the poss

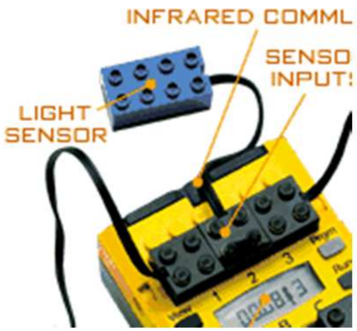
InfoSatellite.com / News

Lego hackers bring TCP/IP to Mindstorn

By Oystein W. Hoie
InfoSatellite.com
January 30, 2002

Reprogramming and modifying Lego's Mindstorms set to create a robotics has become the latest craze in the community. A while universal RCX language was translated to C++, and today, the fi offering full TCP/IP support was announced. Being able to connec the Internet, the possibilites are endless.

Toys have been part of the technological revolution society has seen in the last couple of months, and so has the classic company Lego. Robot sets such as Mindstorms come with a programmable computer that is able to function as the brain of a robot, interfacing with the outside world through a range of sensors.



Considering that a majority of buyers

vnunet.com

Home | News | Jobs | Blogs | Audio/Video | Reviews | White papers | Dow

7 days | Business | Business hardware | Business software | Communicati

Where am I? > Home > News > Hacking

Latest hacker toy? Web-enabled Lego

'Christ' creates world's first TCP/IP-enabled RCX

Written by James Middleton
vnunet.com, 30 Jan 2002

We've seen web servers that fit into matchboxes, but now a protocol hacker ha first web-enabled Lego brick.

By porting the extremely small uIP TCP/IP (transmission control protocol/inter the Lego Mindstorms platform, Olaf Christ has set the foundations for a Lego b

uIP

SWEDISH INSTITUTE OF COMPUTER SCIENCE SICS

IPv6 READY

WATTECO

SAP

CISCO

sensinode

ATMEL

(and many more)

Contiki uIPv6


N504EA


(C) Copyright 2003 Pumpkin, Inc.

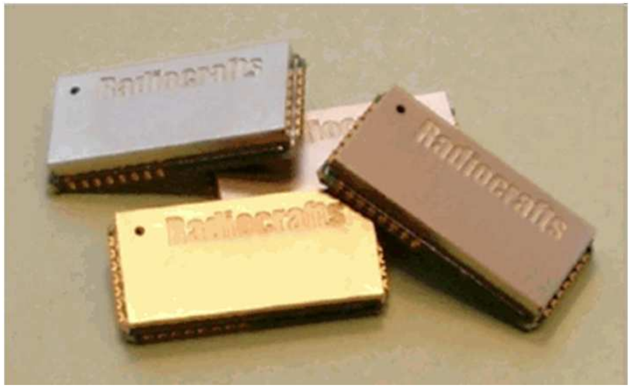

Micronet


YA07 JYW

Fundamental Challenges

Fundamental Challenges

- Cost
- Physical size
- Energy
  - Batteries
- Power
  - Energy scavenging
- Wireless/lossy communication
  - Bandwith
  - Lossy links
- Lack of visibility



Device-level Implications

- Small memory size
  - Kilobytes of RAM
- Simple processor
  - Megahertz
  - No MMU
- Example: MSP430f1449
  - 8 kb RAM, 40 kb ROM

Memory

The size of TCP/IP

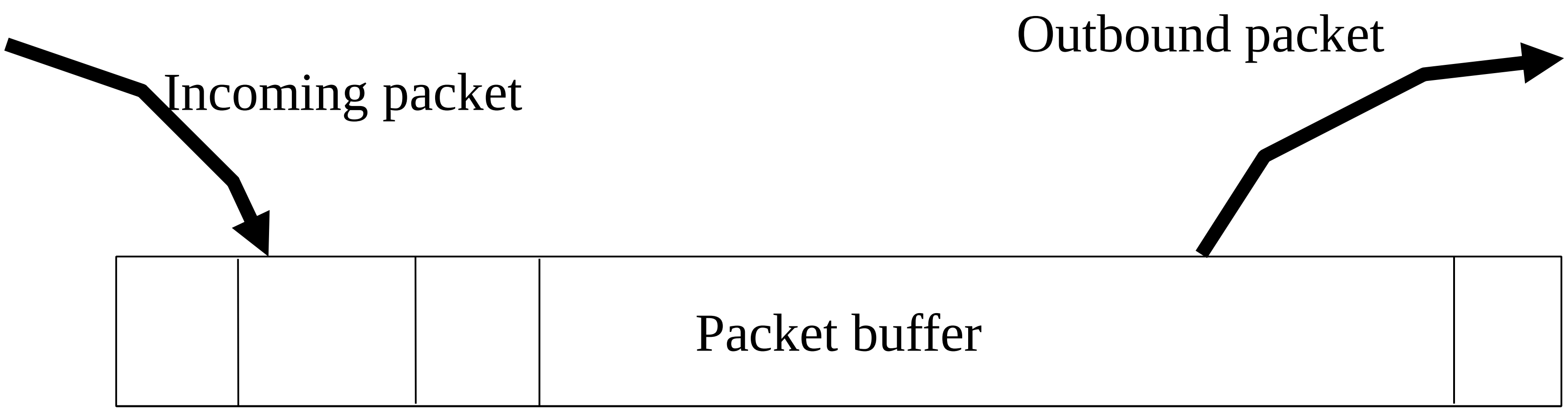
- ”IP is heavyweight”
  - Linux, BSD stacks 200+ kb code, 300+ kb RAM
- lwIP: TCP/UDP/ICMP/IPv4 in 30 kb ROM
- uIP: TCP/UDP/ICMP/IPv4 in 5 kb ROM
- uIPv6: TCP/UDP/ICMPv6/IPv6 in 13 kb ROM

uIP: Making IP Small

- Shared packet buffer
- Event-driven API

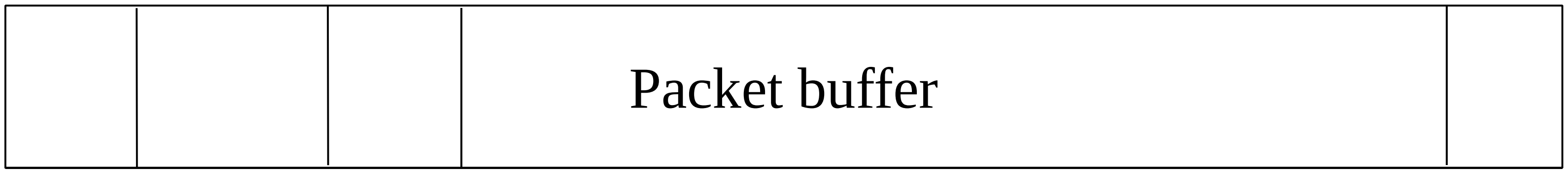
Shared packet buffer

- **All** packets – both outbound and inbound – use the same buffer
  - Size of buffer determines throughput



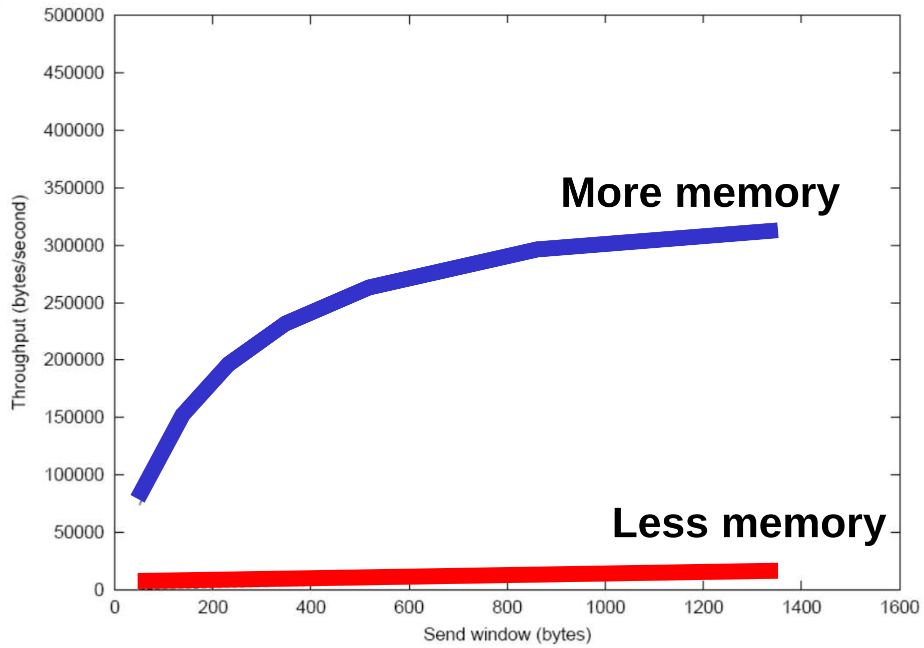
Shared packet buffer II

- Implicit locking: single-threaded access
  1) Grab packet from network – put into buffer
  2) Process packet
     - Put reply packet in the same buffer
  3) Send reply packet into network



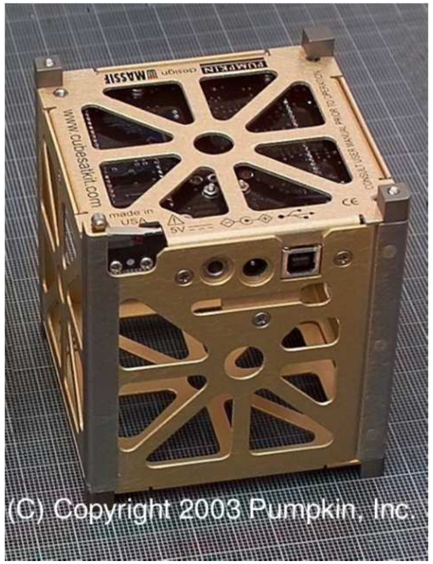
Throughput

- uIP trades throughp
  - Low RAM usage = lo
- Small systems = no
- Current work:
  - Improve TCP throughput by low-power pipelined forwarding



Example: uIP-based Pico Satellite

- CubeSat pico satellite
  - MSP430-based
- 128 bytes of RAM for uIP



Application Programming Interface I

- uIP does not have BSD sockets
  - BSD sockets are built on threads
  - Threads induce overhead (RAM)
- Instead – event-driven API
- Execution is always initiated by uIP
  - Applications are called by uIP, call must return
- Protosockets – BSD socket-like API based on protothreads

Application Programming Interface II

```
void example2_app(void) {
  struct example2_state *s =
   (struct example2_state *)uip_conn->appstate;

  if(uip_connected()) {
    s->state = WELCOME_SENT;
    uip_send("Welcome!\n", 9);
    return;
  }

  if(uip_acked() &&
    s->state == WELCOME_SENT) {
    s->state = WELCOME_ACKED;
  }

  if(uip_newdata()) {
    uip_send("ok\n", 3);
  }

  if(uip_rexmit()) {
    switch(s->state) {
    case WELCOME_SENT:
      uip_send("Welcome!\n", 9);
      break;
    case WELCOME_ACKED:
      uip_send("ok\n", 3);
      break;
    }
  }
}
```

Application Programming Interface III

- Event-driven API sometimes is problematic
  - Not all programs are well-suited to it
  - Programs are explicit state machines
- Protosockets: sockets-like API using protothreads
  - Extremely lightweight stackless threads
  - 2 bytes per-thread state, no stack
- Protothreads allow “blocking” functions, even when called from uIP

Application Programming Interface IV

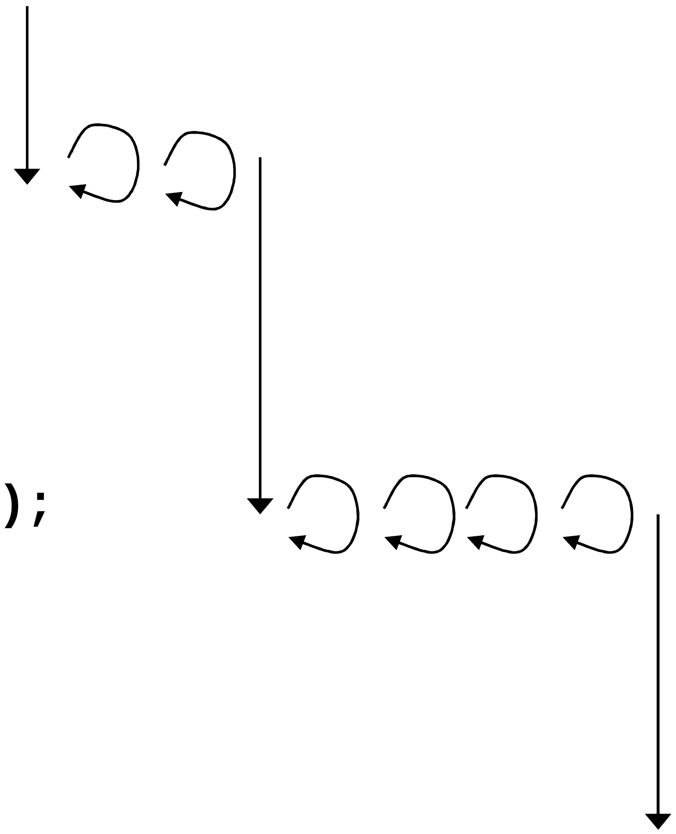
```
PT_THREAD(smtp_protothread(void))
{
  PSOCK_BEGIN(s);

  PSOCK_READTO(s, '\n');

  if(strncmp(inputbuffer, “220”, 3) != 0) {
    PSOCK_CLOSE(s);
    PSOCK_EXIT(s);
  }

  PSOCK_SEND(s, “HELO ”, 5);
  PSOCK_SEND(s, hostname, strlen(hostname));
  PSOCK_SEND(s, “\r\n”, 2);

  PSOCK_READTO(s, '\n');

  if(inputbuffer[0] != '2') {
    PSOCK_CLOSE(s);
    PSOCK_EXIT(s);
  }
```

Protothreads: Lightweight, thread-like programming

- A design point between events and threads
- Programming primitive: conditional blocking wait
  - PT_WAIT_UNTIL(*condition*)
- Single stack
  - Low memory usage, just like events
- Sequential flow of control
  - No explicit state machine, just like threads
  - Programming language helps us: **if** and **while**

An example protothread

```
int a_protothread(struct pt *pt) {
  PT_BEGIN(pt);

   /* … */

  PT_WAIT_UNTIL(pt, condition1);

   /* … */

  if(something) {

     /* … */

    PT_WAIT_UNTIL(pt, condition2);

     /* … */
  }

  PT_END(pt);
}
```



Six-line implementation

Protothreads implemented using the C switch statement

```
struct pt { unsigned short lc; };

#define PT_INIT(pt)           pt->lc = 0
#define PT_BEGIN(pt)          switch(pt->lc) { case 0:
#define PT_EXIT(pt)           pt->lc = 0; return 2
#define PT_WAIT_UNTIL(pt, c)  pt->lc = __LINE__; case __LINE__: \
                              if(!(c)) return 0
#define PT_END(pt)            } pt->lc = 0; return 1
```


Adam Dunkels

C-switch expansion

```
int a_protothread(struct pt *pt) {
  PT_BEGIN(pt);


  PT_WAIT_UNTIL(pt, condition1);


  if(something) {


    PT_WAIT_UNTIL(pt, condition2);


  }

  PT_END(pt);
}
```

```
int a_protothread(struct pt *pt) {
  switch(pt->lc) { case 0:


  pt->lc = 5; case 5:
  if(!condition1) return 0;

  if(something) {


    pt->lc = 10; case 10:
    if(!condition2) return 0;

  }

  } return 1;
}
```

Line numbers

Memory

- Yes, IP can be done in small amounts of memory
  - But may affect performance
- Event-driven interfaces, bottom-up design, static memory allocations reduce memory
  - But changes the API

Power and Energy

Power and Energy are Crucial

- Unattended operation, long lifetime
- Battery-powered nodes
  - Replacement, recharging not feasible/possible
- Energy generation
  - Low power

Radio Power Consumption

- Radio dominates power consumption
- Listening as expensive as transmitting

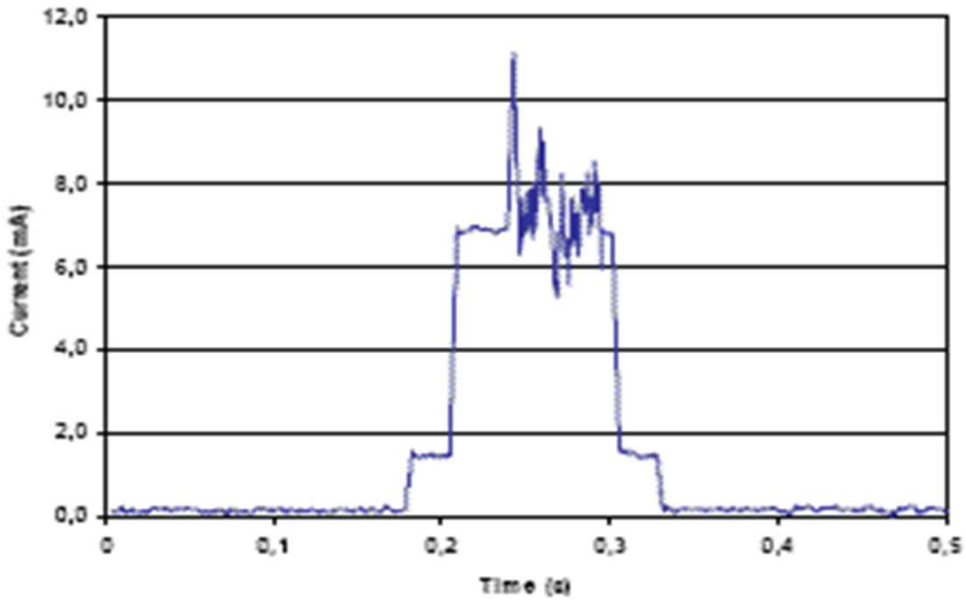


Figure 6. Power consumption: sending

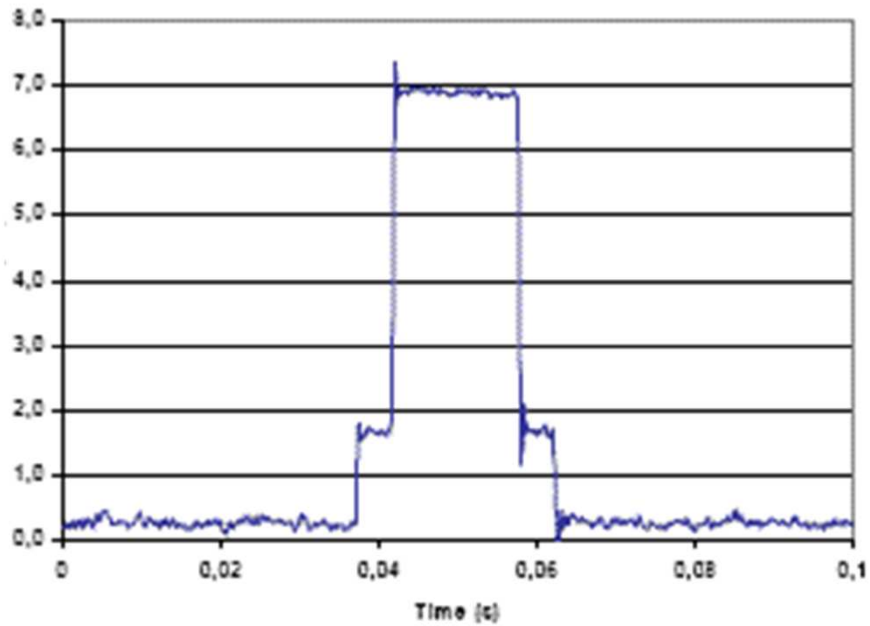
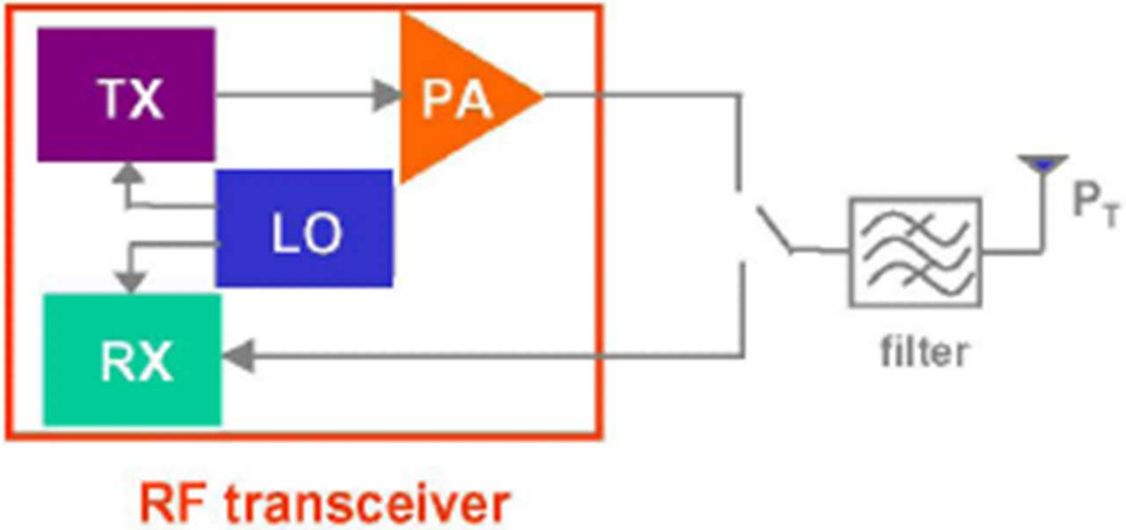


Figure 7. Power consumption: receiving

Why is Listening More Expensive than Transmitting?



	GSM	802.11b	Bluetooth
$P_{RX} + P_{LO}$ (mW)	240	60	30
$P_{TX} + P_{LO}$ (mW)	360	100	12
P_T (mW)	1000	100	1
P_{PA} (mW)	2500	250	2.5
η_P	32%	24%	2%

Source: Wang & Sodini, ICC 2006

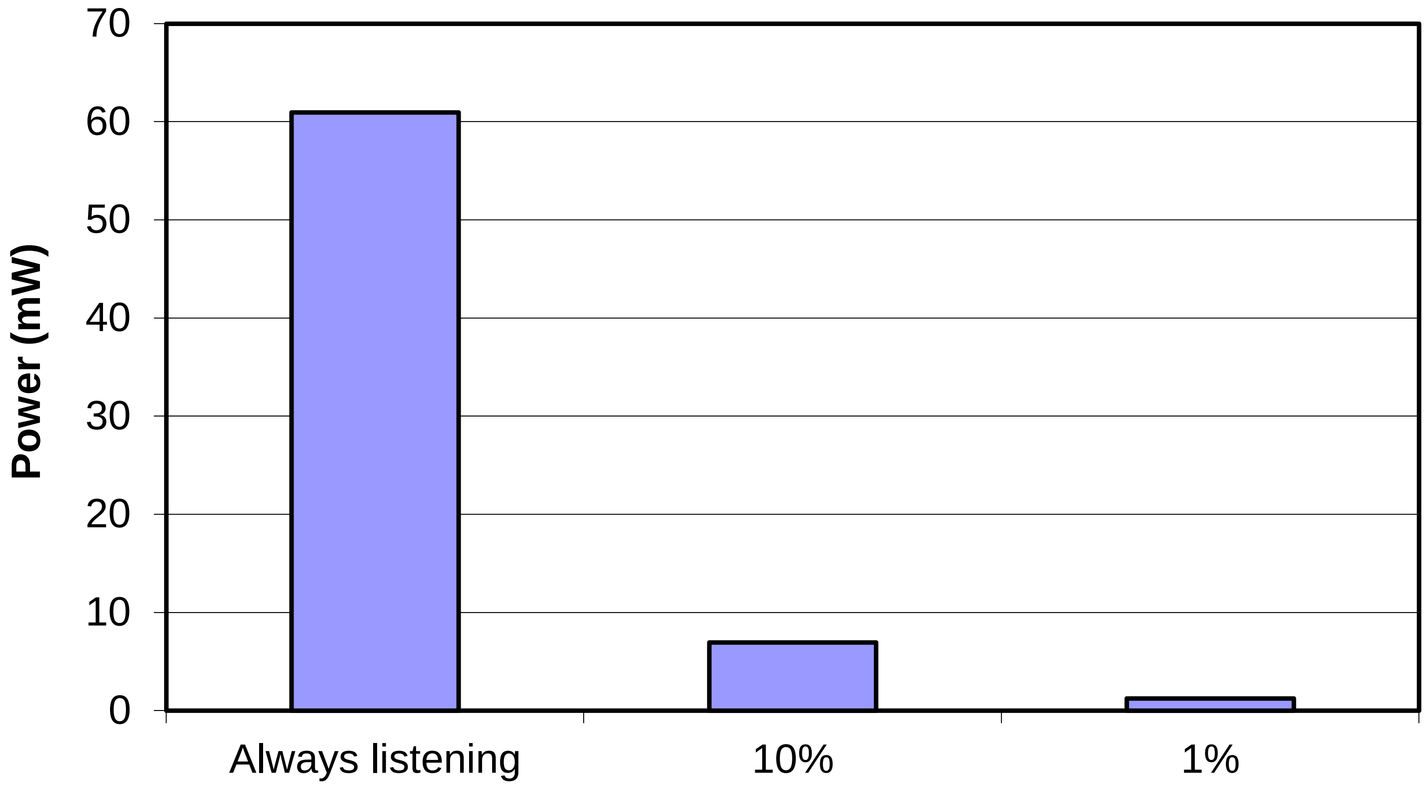
Low-power radio hardware

- IEEE 802.15.4 (250 kilobits/second)
  - Power ~60 mW
  - Sleep ~0.01 mW
  - Range 40 m
- Low-power WiFi (2-50 megabits/second)
  - Power ~300 mW
  - Sleep ~0.02 mW
  - Range 400 m

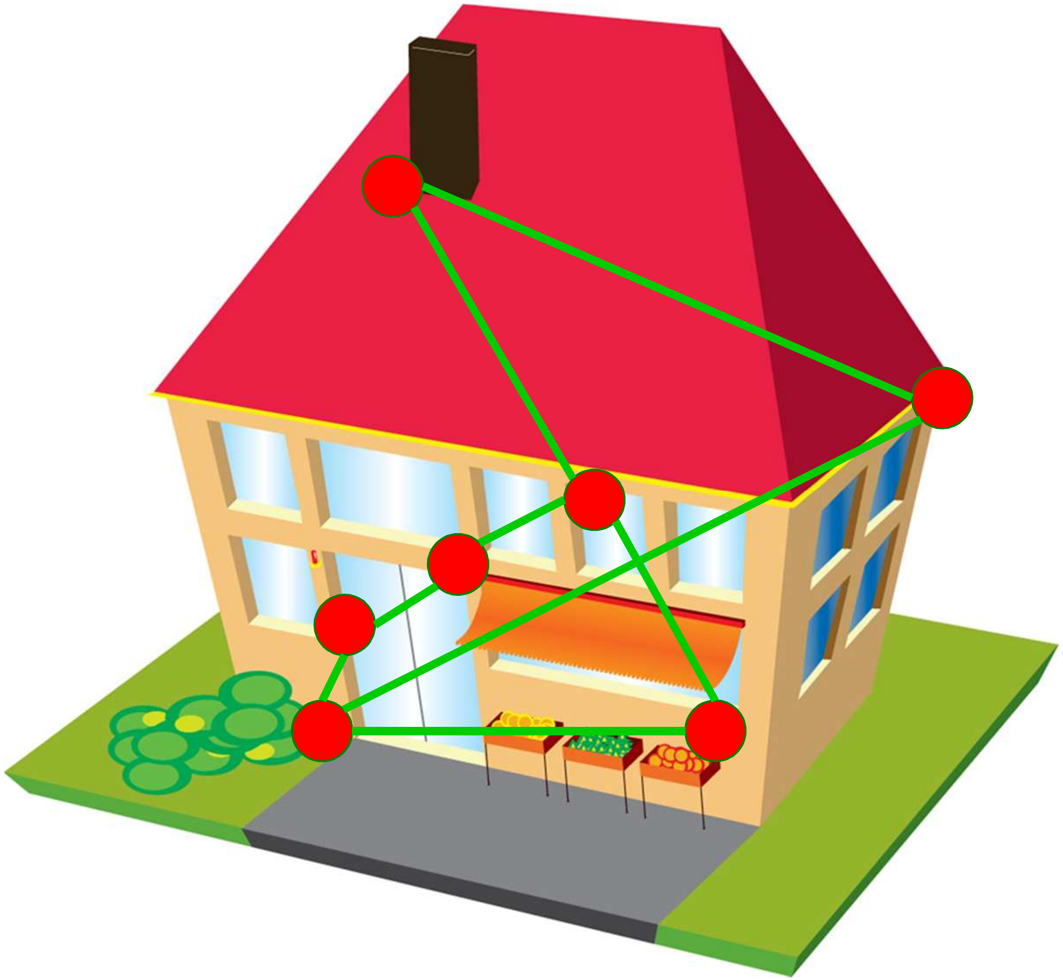
Listening is Expensive

- Being always on kills you quickly
  - Days of lifetime on batteries
- What about always sleeping?
  - Waking up only to send
  - Wake up on the hour, every hour
  - Introduces strange semantics
  - What about multi-hop?

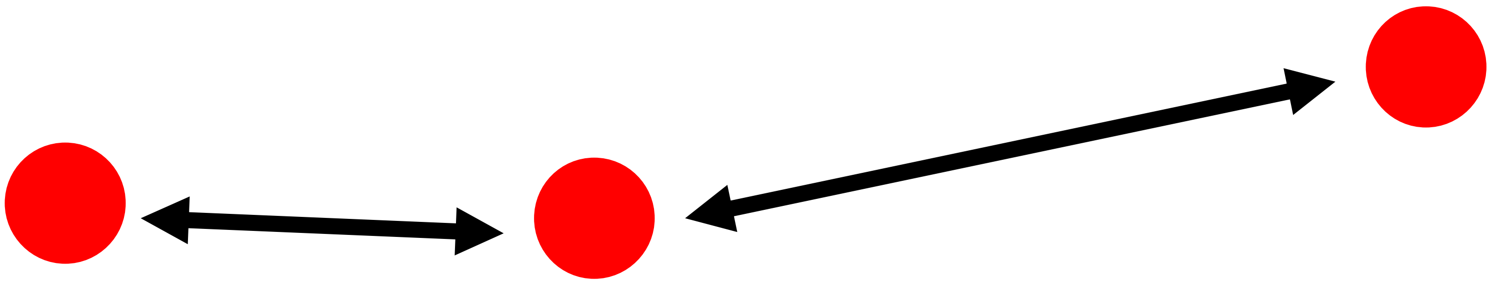
Listening is expensive



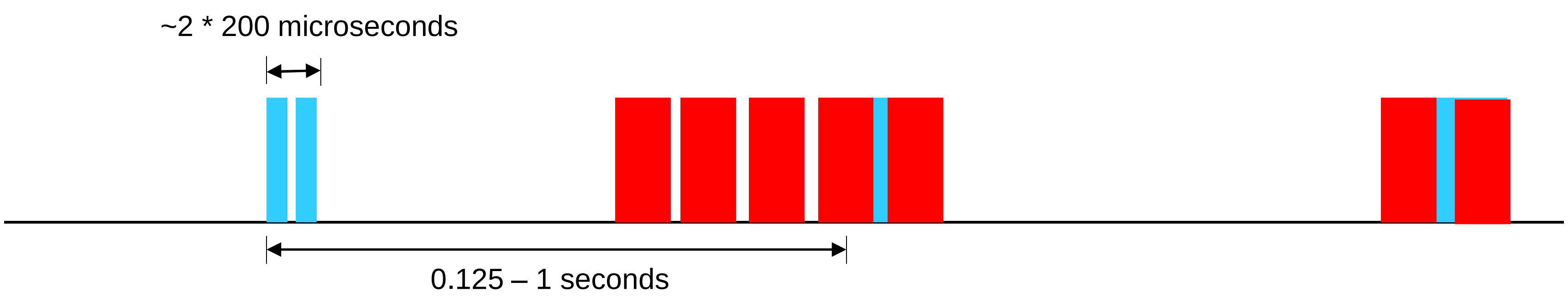
Multi-hop communication



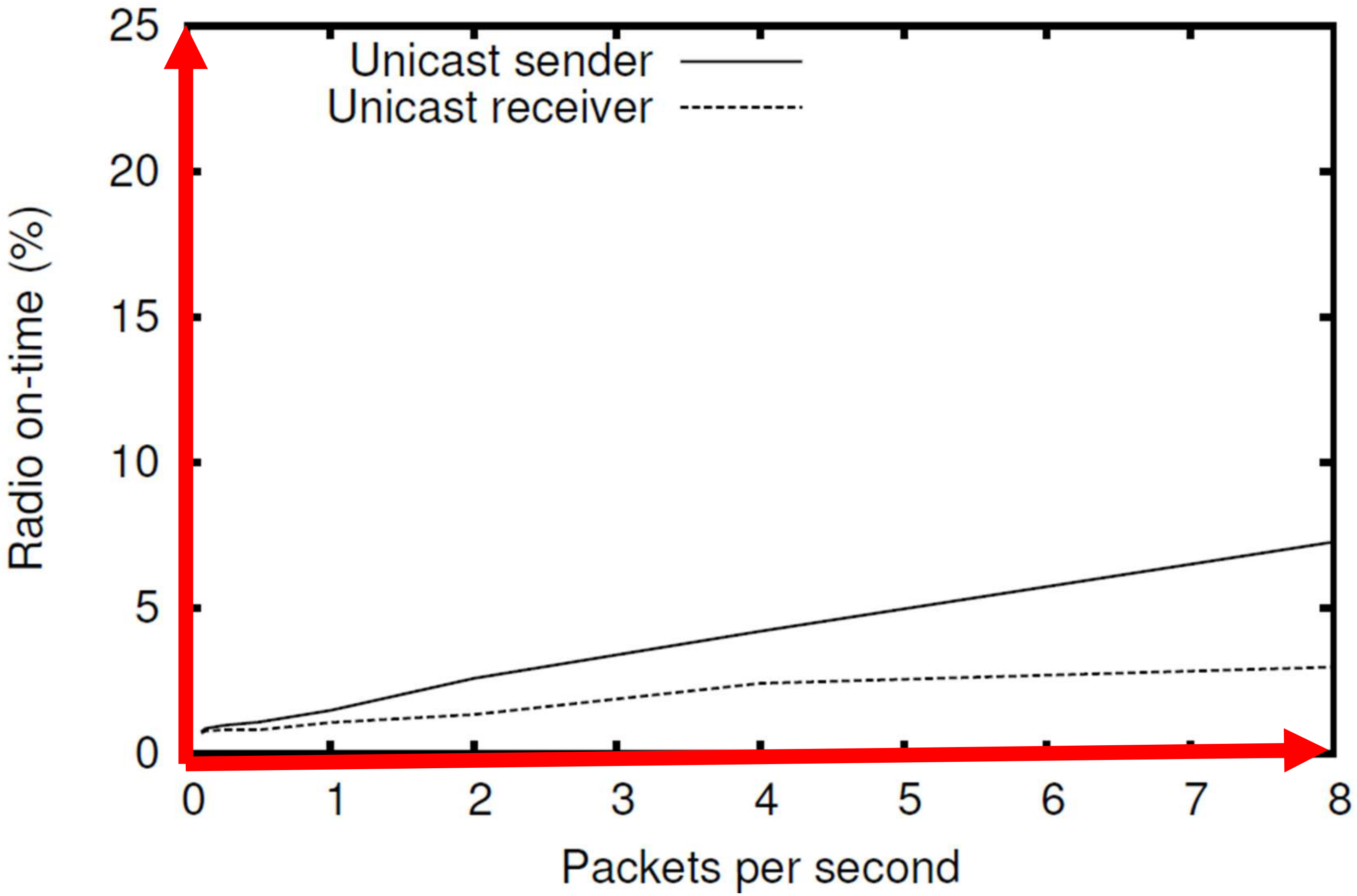
Multi-hop communication



Duty Cycling (ContikiMAC)

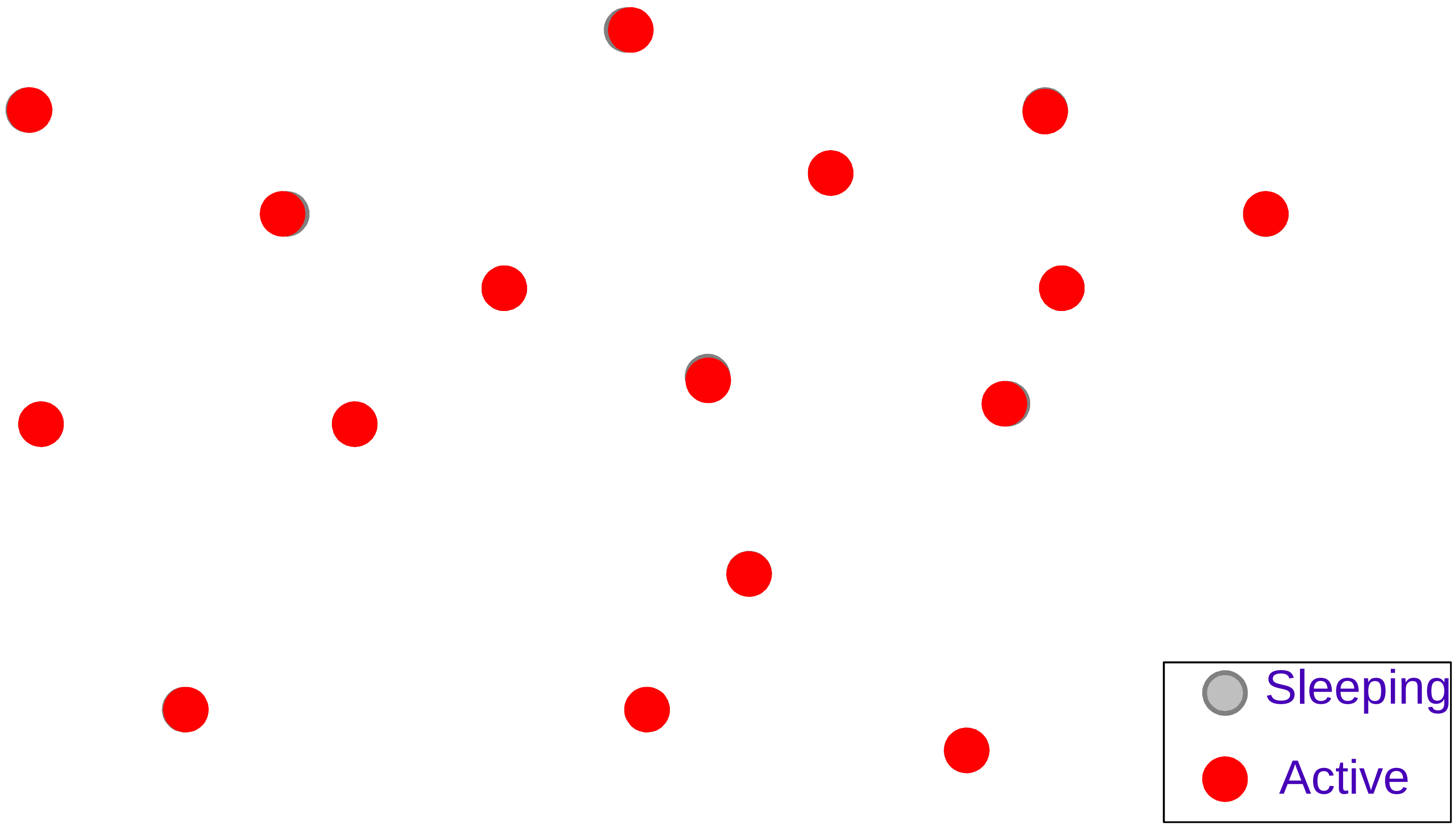


Efficiency

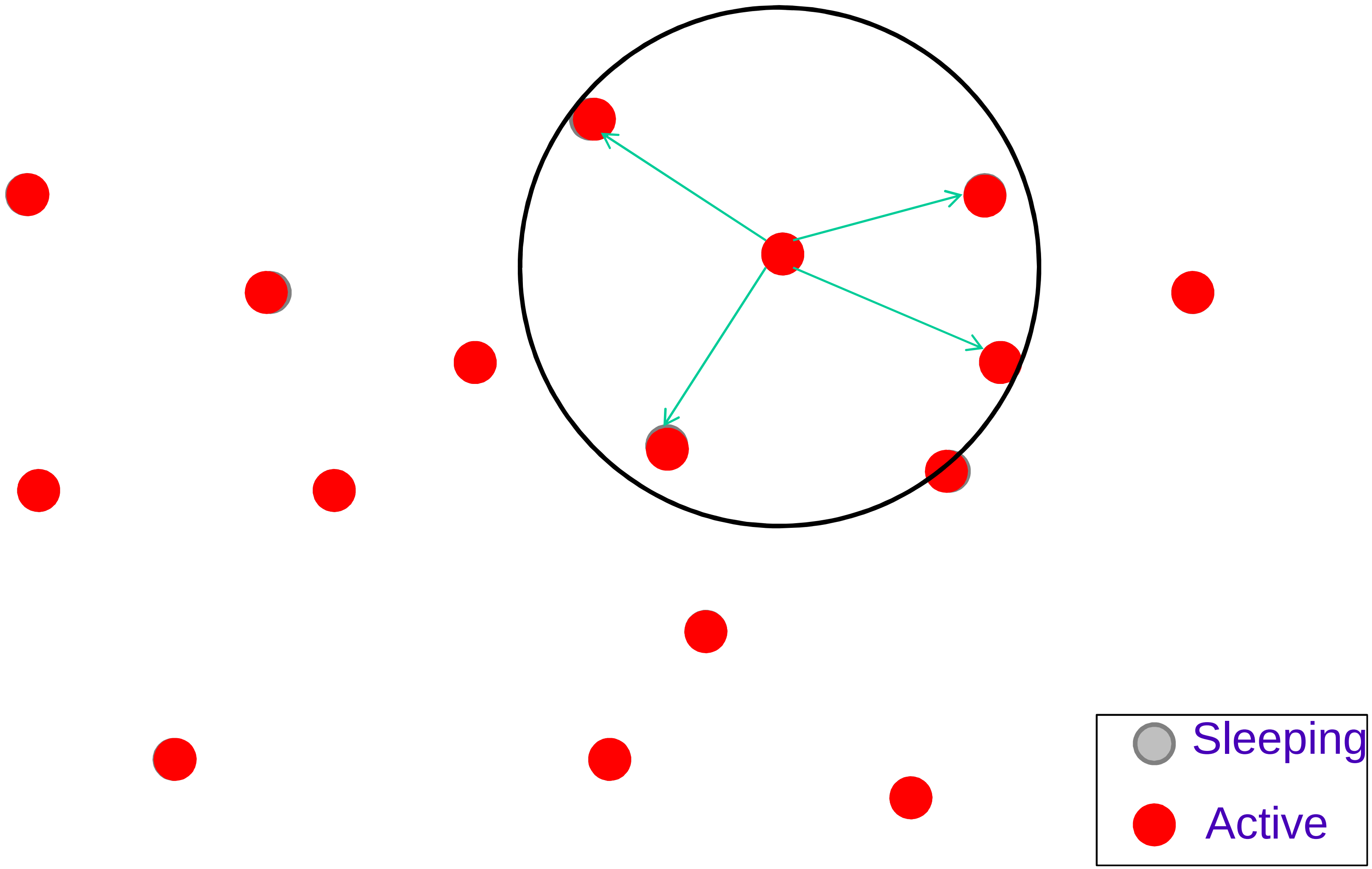


Unicast sender
Unicast receiver
Radio on-time (%)
25
20
15
10
5
0
0
1
2
3
4
5
6
7
8
Packets per second

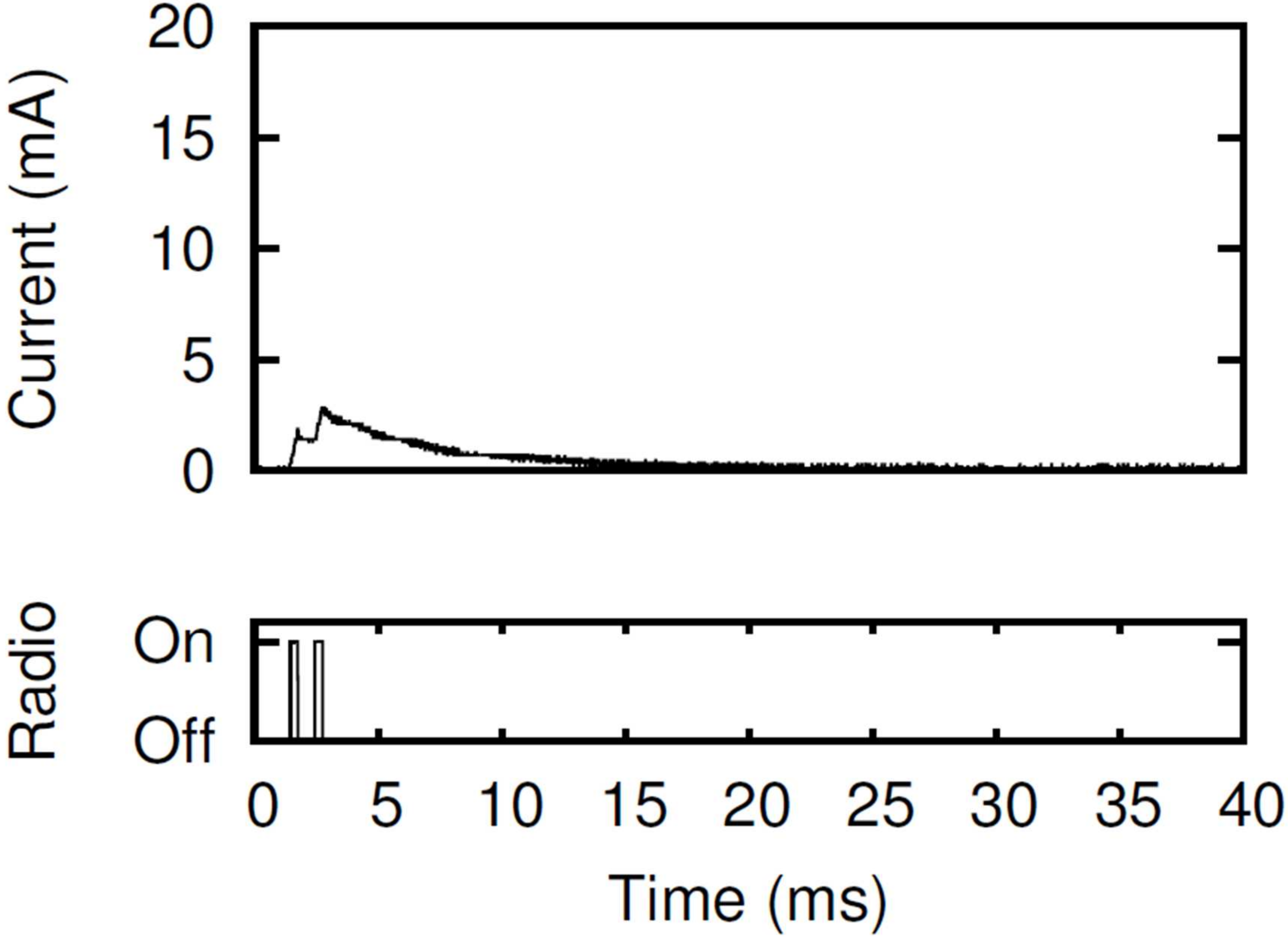
Duty Cycling



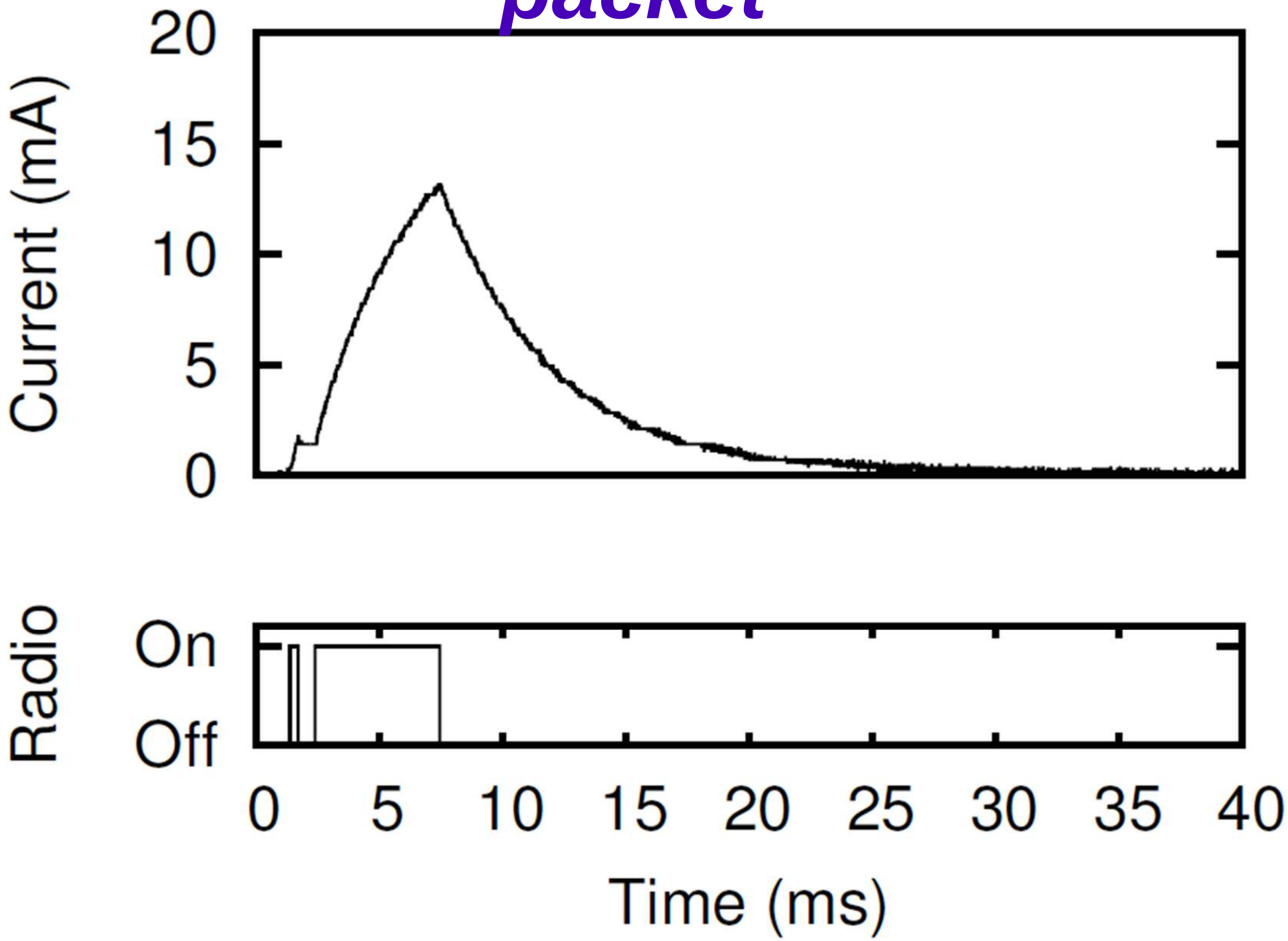
Broadcast with Duty Cycling



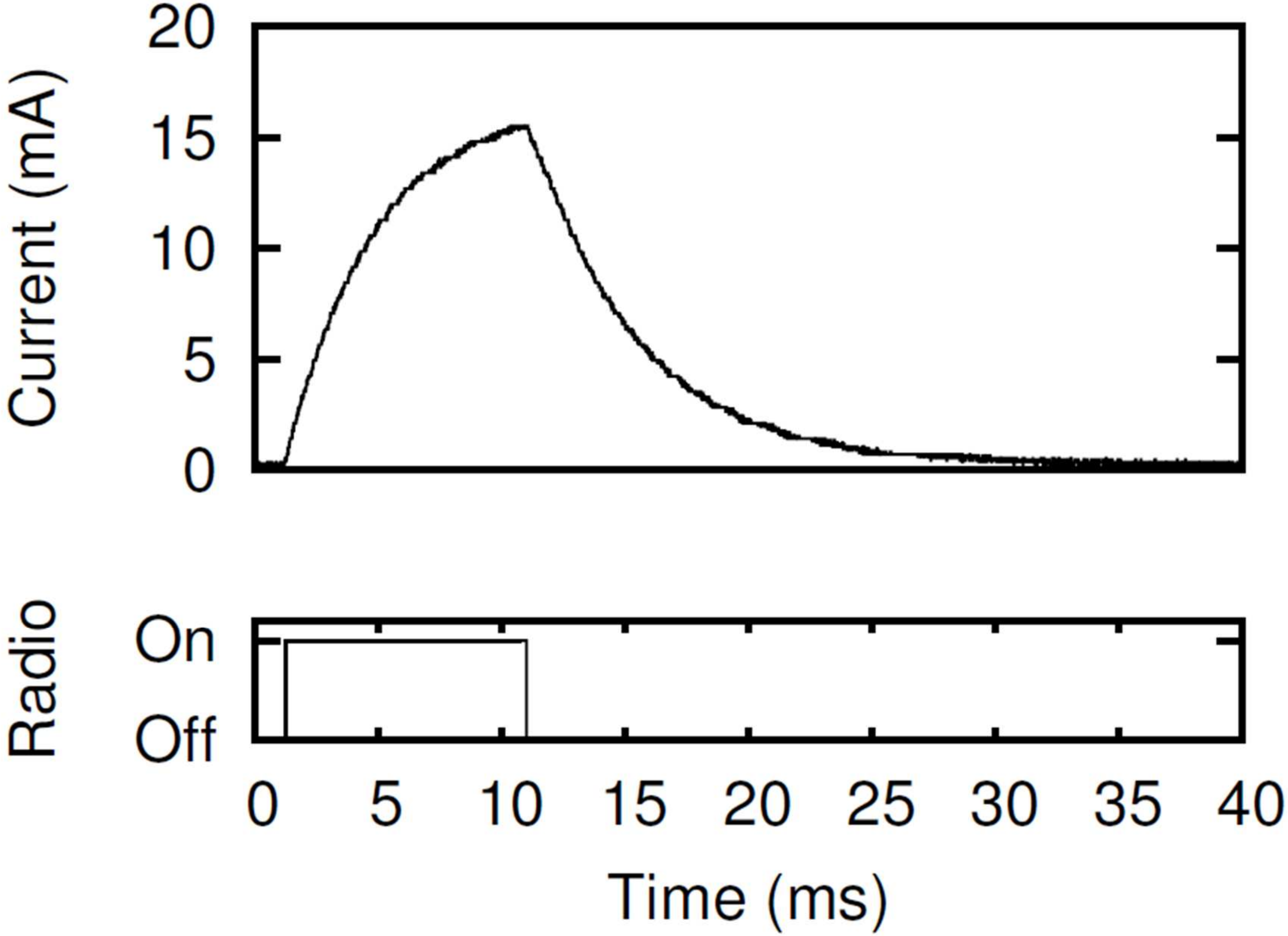
Wake-up



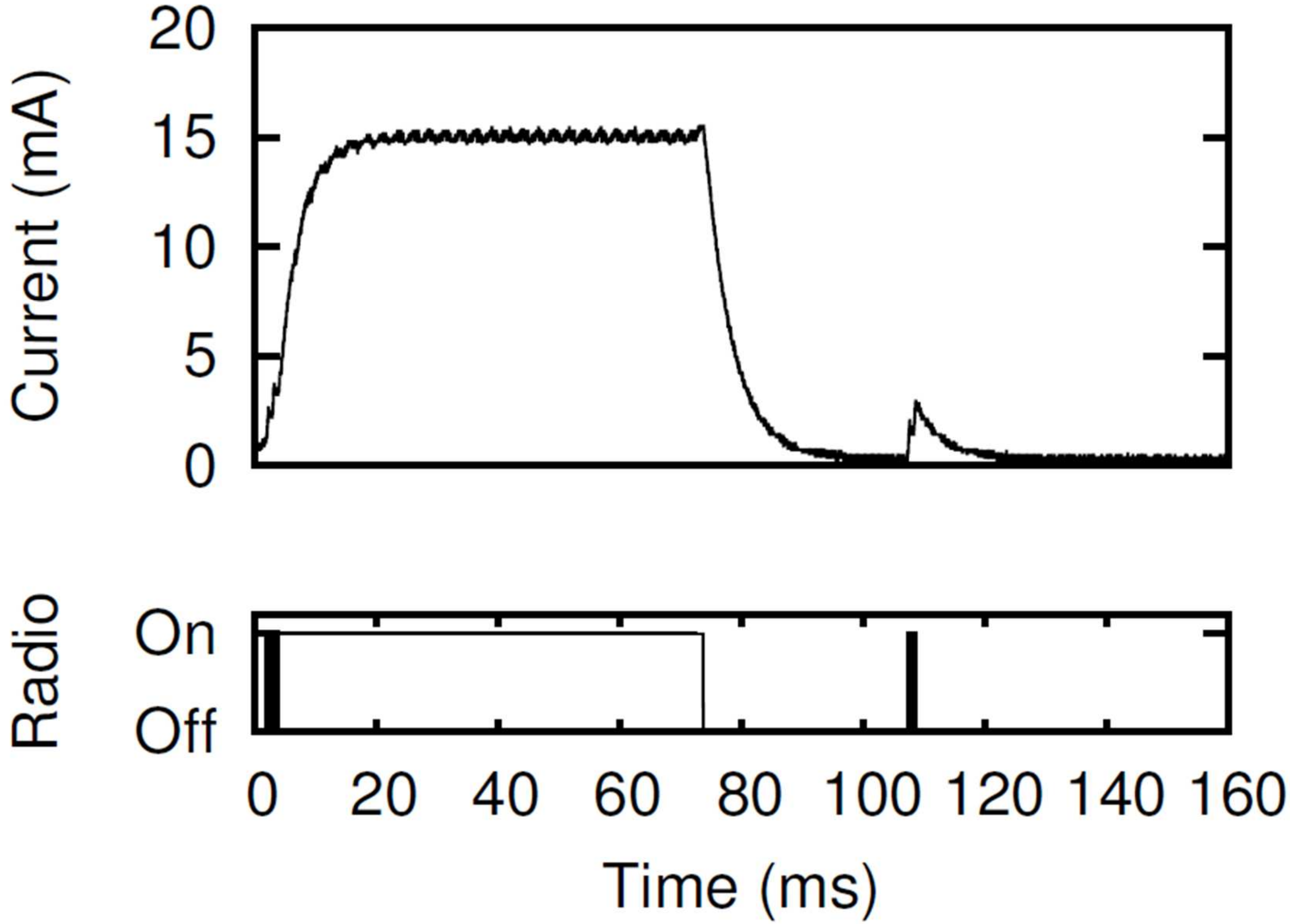
Wake-up, signal detected but no packet



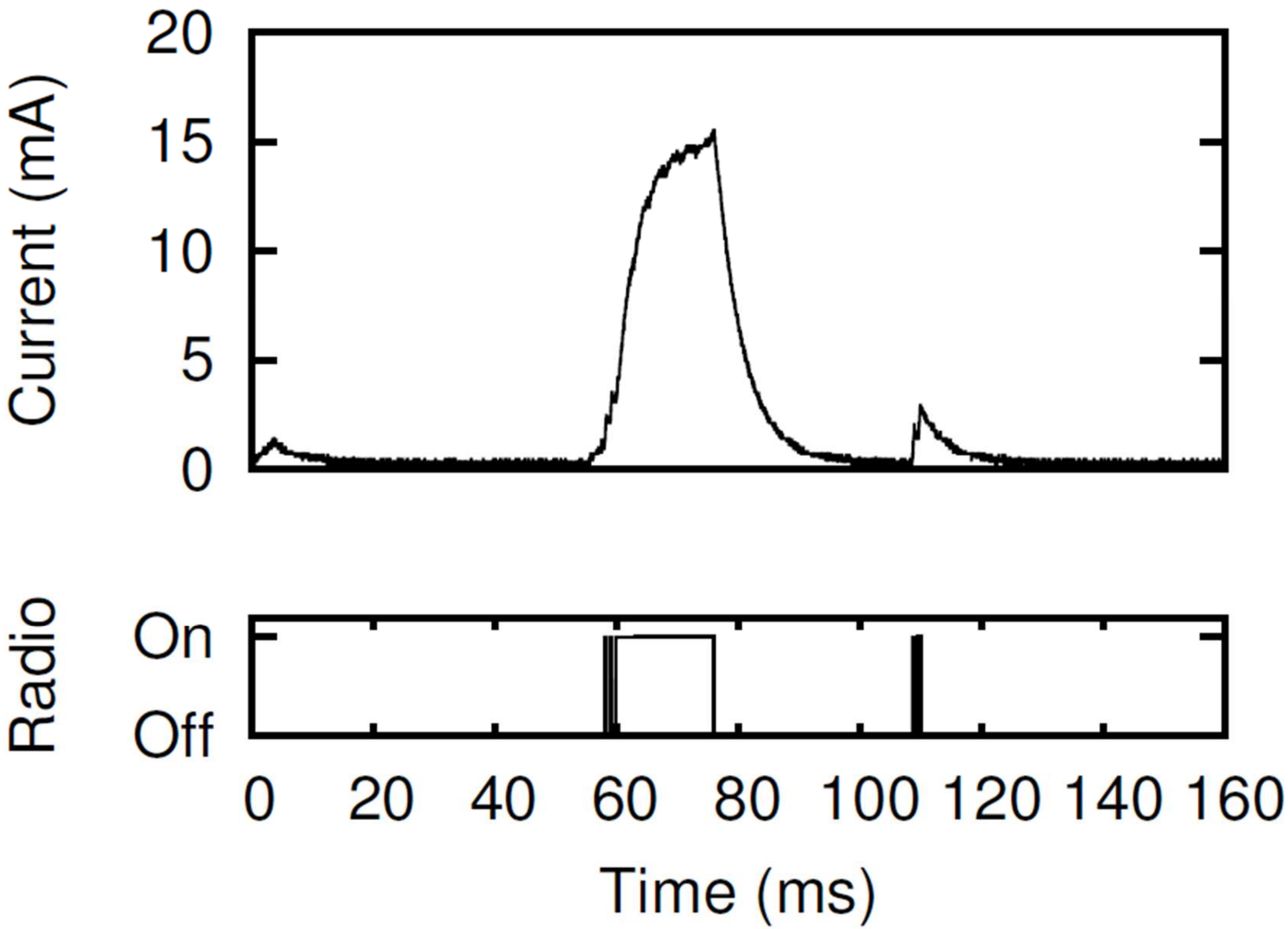
Reception



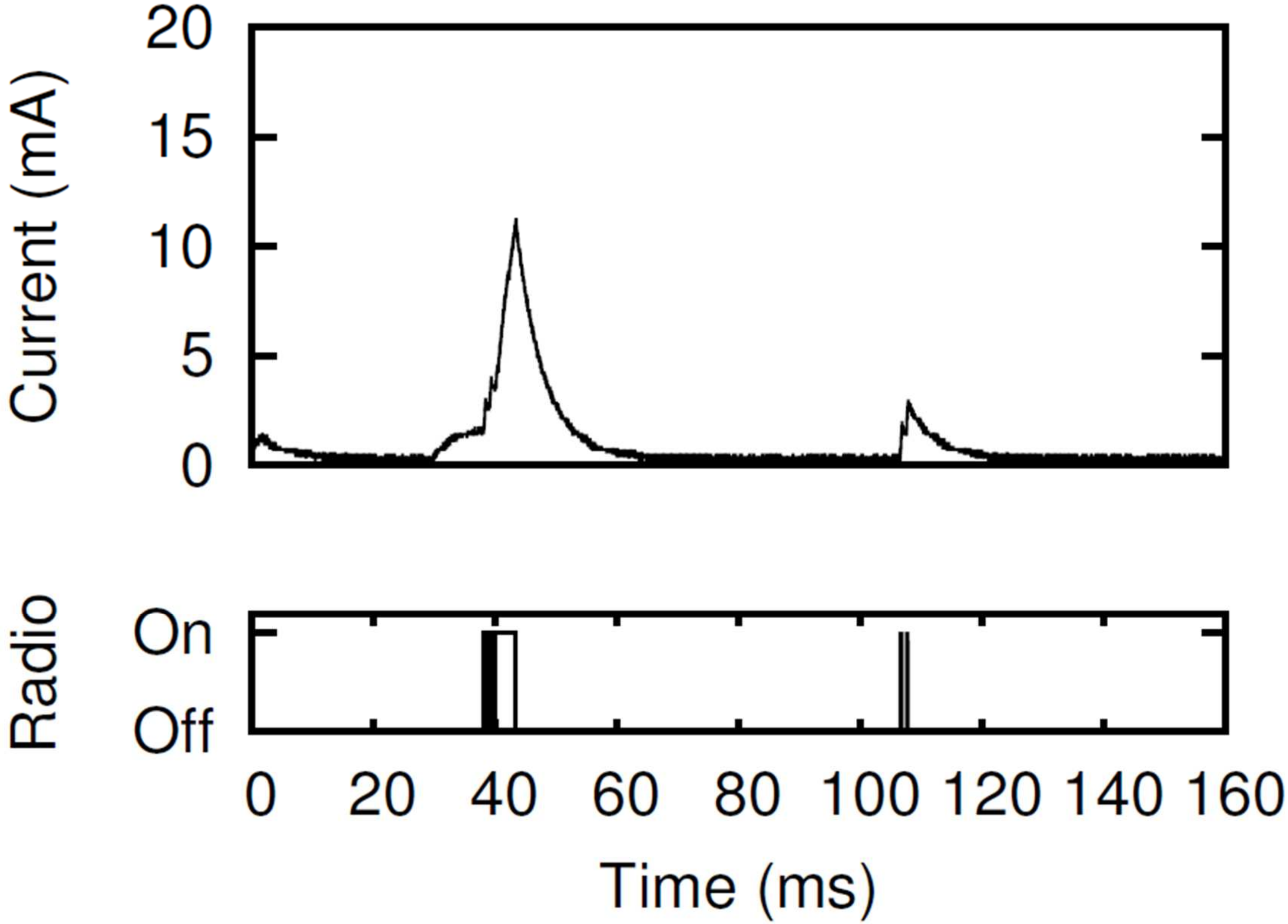
Unicast transmission, first time



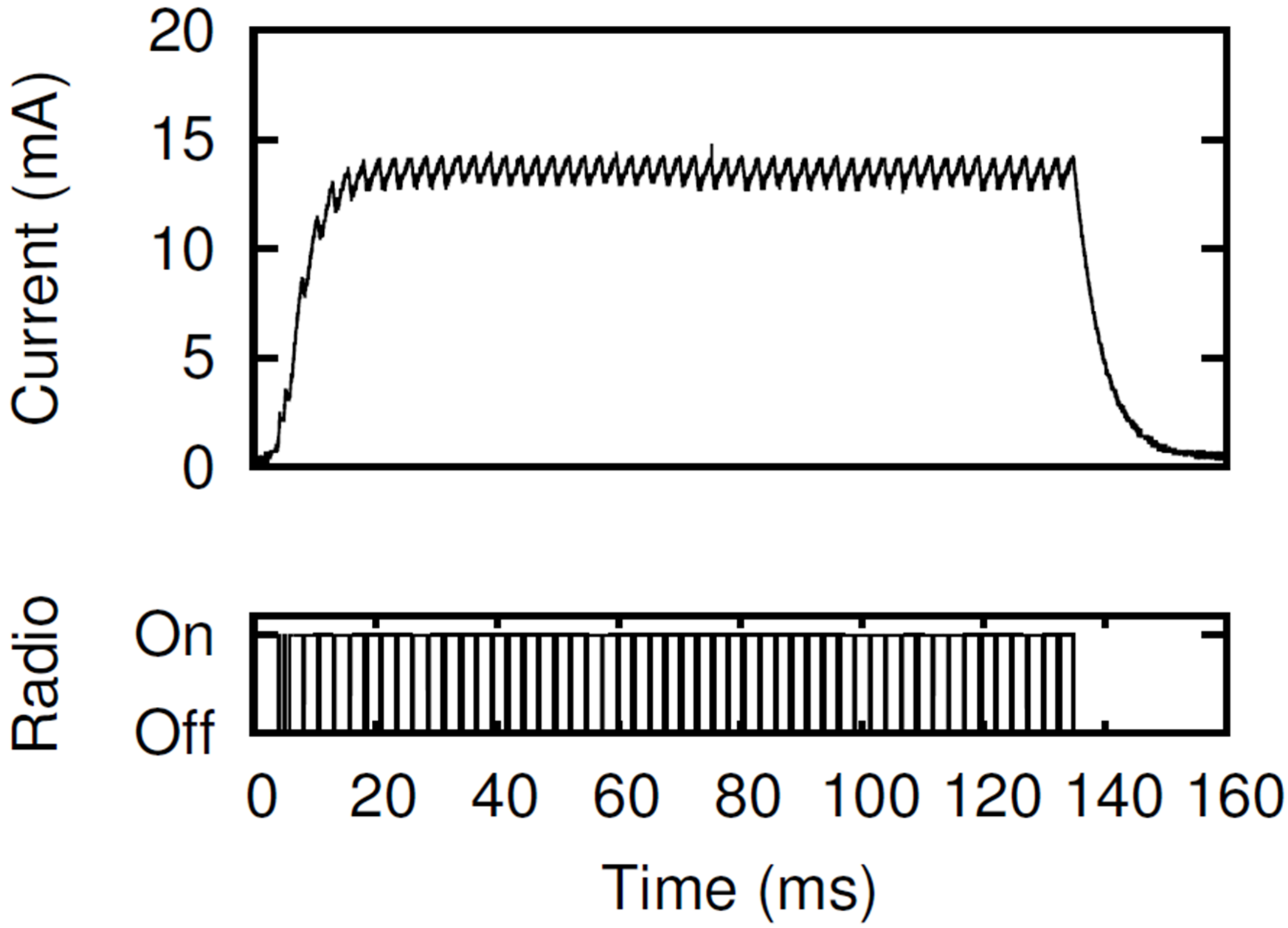
Unicast transmission, second time



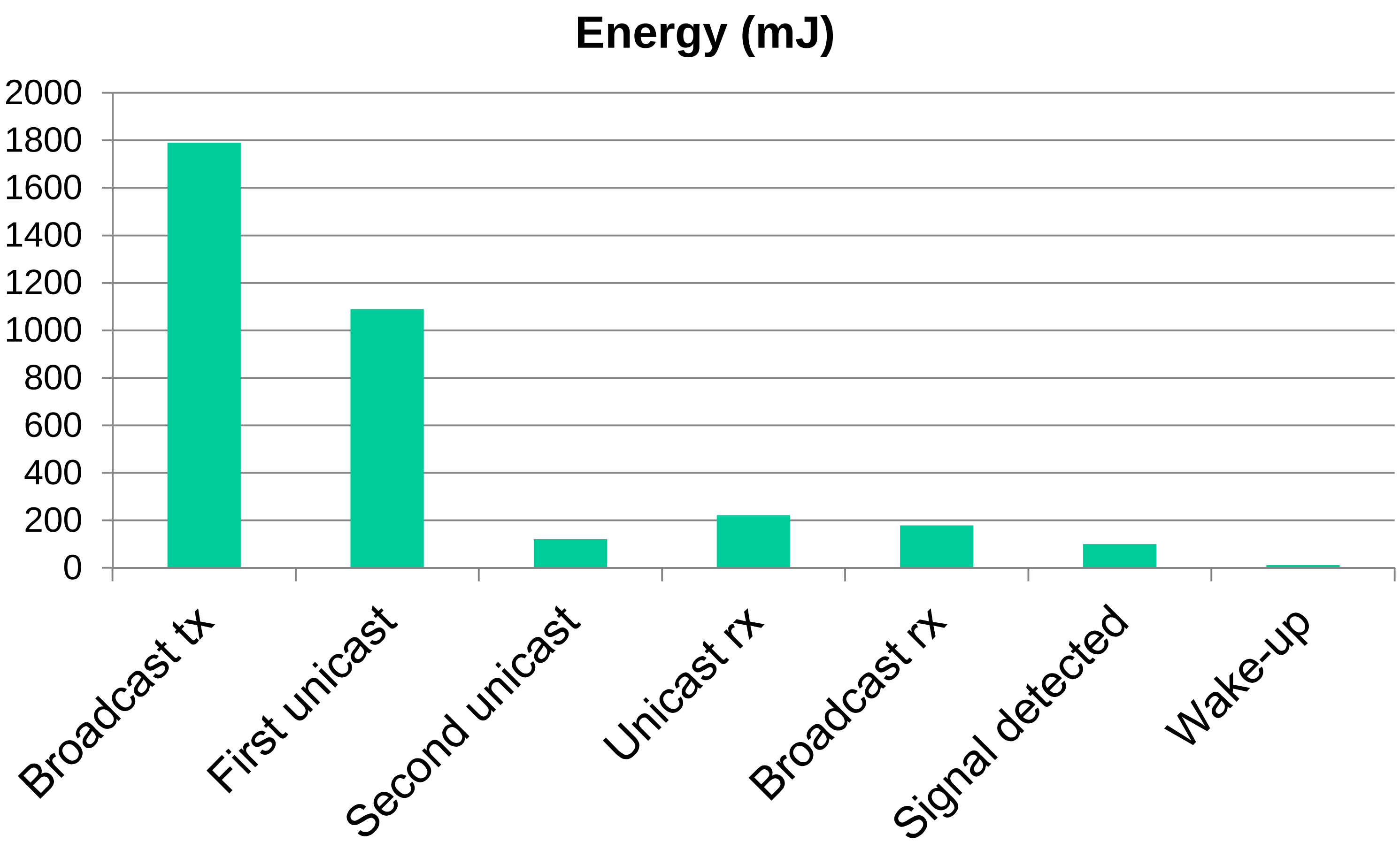
Unicast Tx to Awake Neighbor



Broadcast transmission



Energy



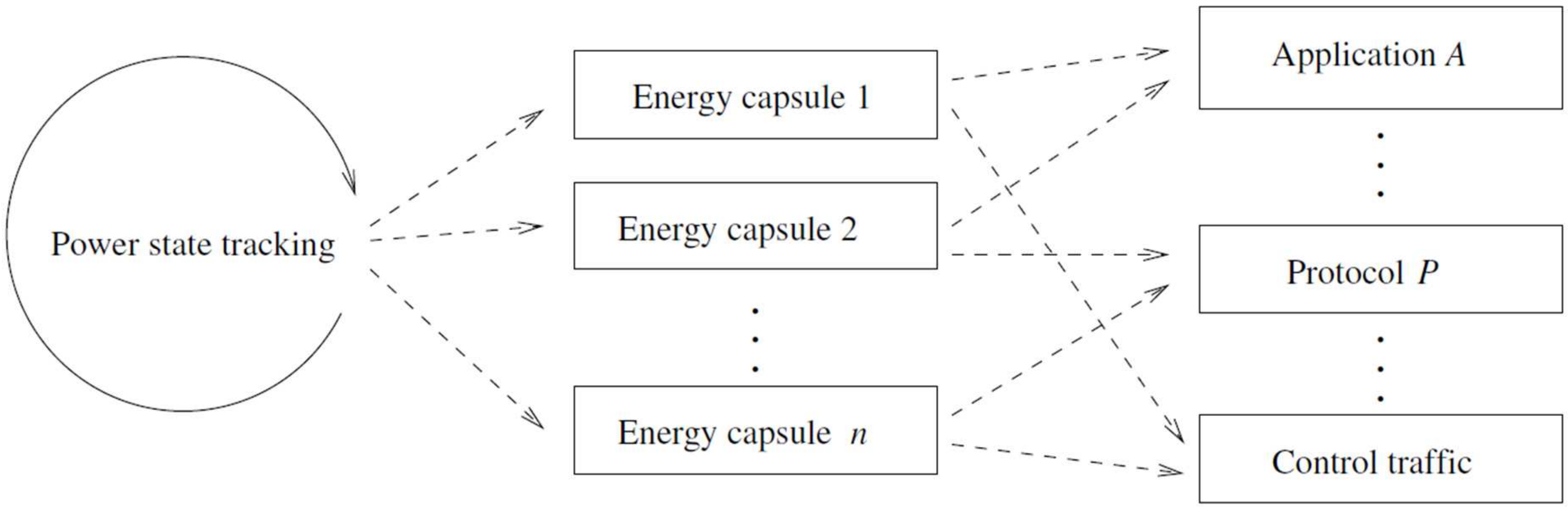
Energy (mJ)
2000
1800
1600
1400
1200
1000
800
600
400
200
0
Broadcast tx
First unicast
Second unicast
Unicast rx
Broadcast rx
Signal detected
Wake-up

Broadcast in Low-power Wireless

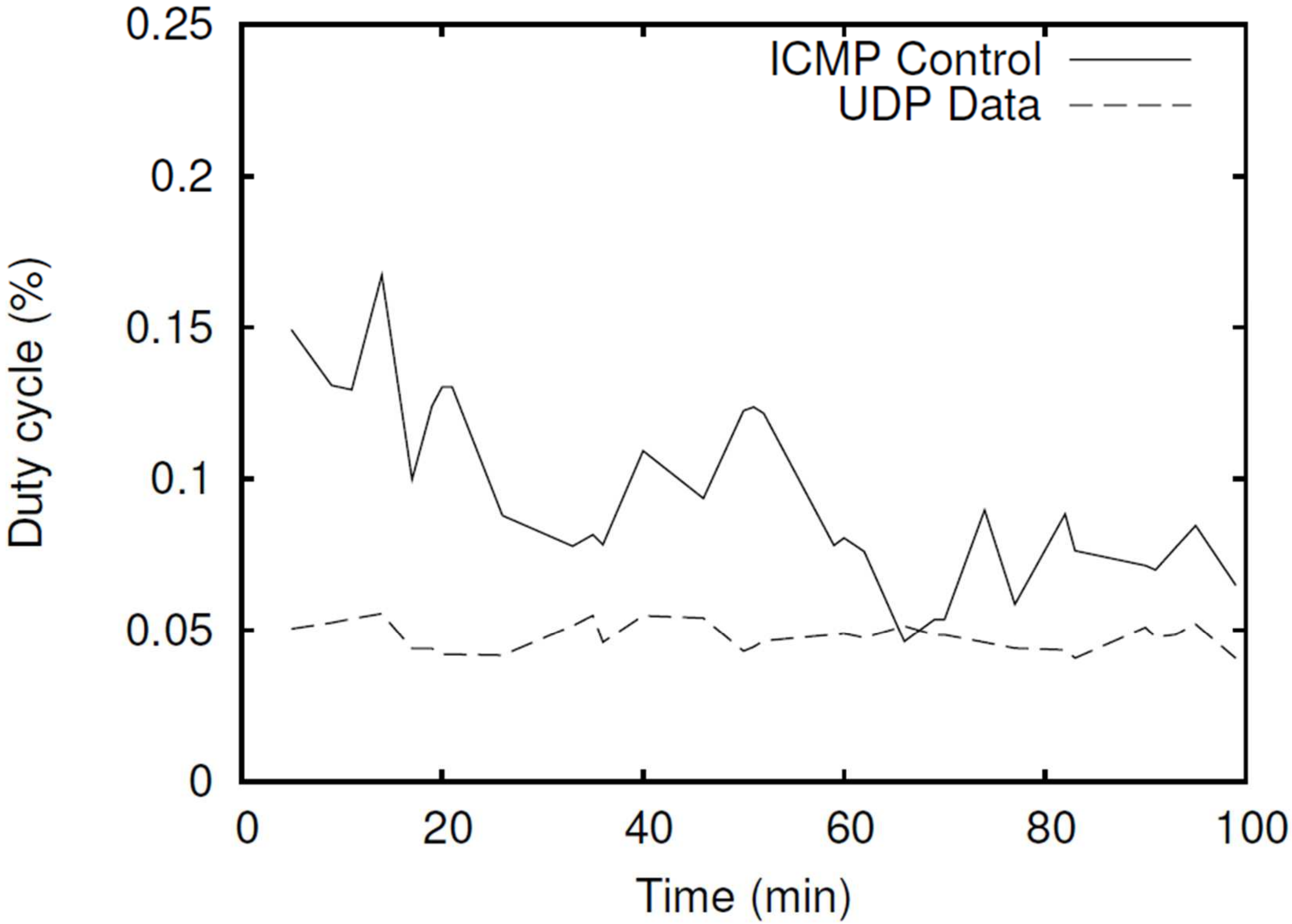
- Semantics slightly changed
  - Not necessarily atomic
  - Not necessarily synchronous
- Quantitative changes
  - Broadcast more expensive than unicast

Contiki Powertrace

- Network-level energy estimation
- Power state tracking, energy capsules
- Energy attribution to network-level activities



Power Consumption in RPL



What to do about Broadcast?

- Still an open question
  - Adaptive beaconing (CTP)
  - Beacon suppression (Trickle, RPL)
  - Beacon coordination (Dunkels et al, EWSN 2011)
  - Politecast (Lunden and Dunkels, ACM CCR April 2011)

Alternatives to Asynchronous Duty Cycling

- Sleepy nodes
  - Never wake up to receive
  - But fundamentally changes semantics of IP
- Time-synchronized wake-ups
  - Schedule wake-ups at known times, schedule broadcast slot
  - Makes broadcast and unicast equivalent, but spends idle energy on broadcast anyway

Power and Energy

- Power and communication intertwined
- Must turn radio off to conserve power
  - Reducing transmissions is not enough
- Duty cycling slightly changes things
  - Non-atomic, non-synchronous
  - Broadcast gets expensive
  - But it does not fundamentally change the semantics

Conclusions

- Lightweight, low-power IP
  - Memory
  - Power
- The memory vs performance trade-off
  - API changes
- The power vs communication trade-off
  - Changes to protocols may be needed

Thank you

http://www.sics.se/contiki/