



picoTCP

**The reference TCP/IP stack for the
Internet of Things**

FOSDEM 2014

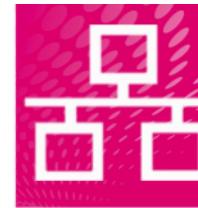


A leading **embedded** technology company, located in Leuven and Gent, providing **Consultancy, Training, Projects** and **Outsourced Services**

We ensure that our clients in **co-creation** with TASS can launch innovative and better **quality** products with a shorter **time to market**.

We believe "**Connected devices**" improve the quality of our lives in an innovative way.

That's why we made this...



picoTCP

A fully featured, highly portable **TCP/IP stack** designed for **small footprint embedded systems**.

The reference TCP/IP stack for the Internet of Things

Our focus areas:

Modularity

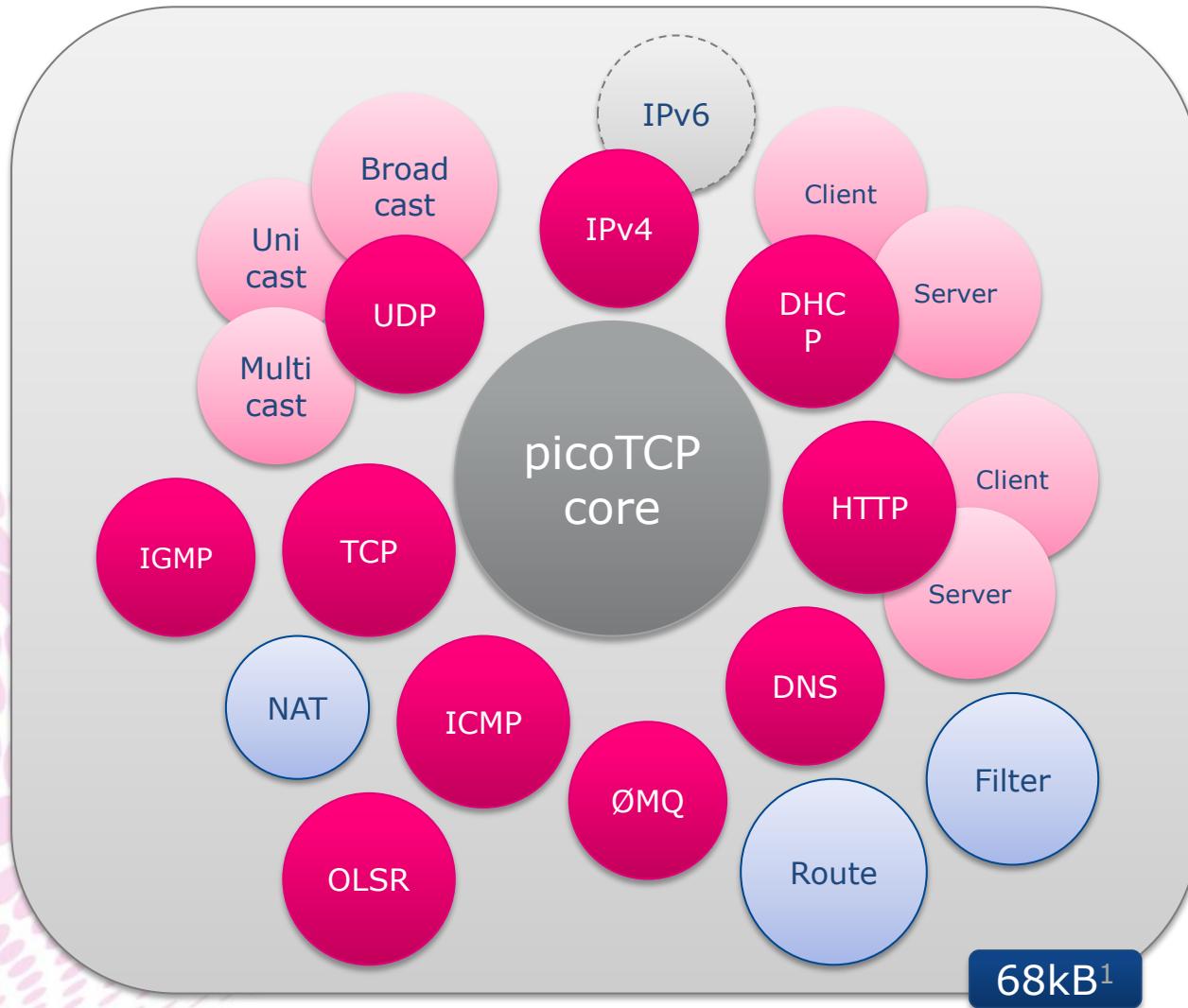
Portability

Performance

Quality

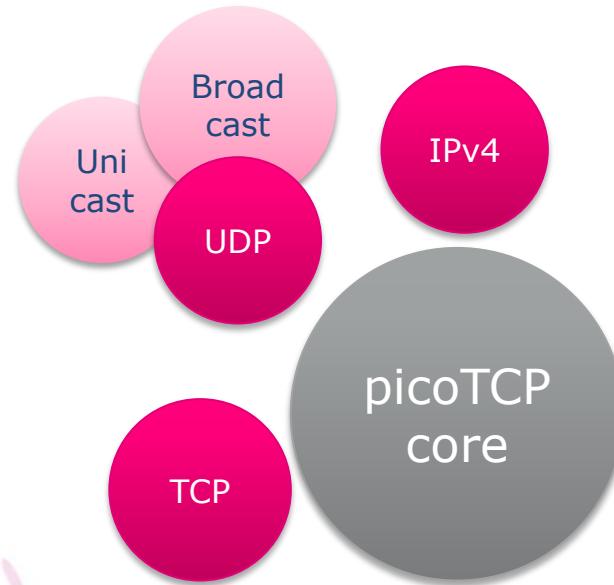
Free as in free speech

- Dual licensing policy in place
 - Freely distributed under the **GNU General Public License v2** for the benefit of the community
 - **Proprietary license available** at user's option, when the platform code can't fully comply with the terms of GPL
 - **Full copyright is owned by TASS:** different licensing agreements are possible for special cases

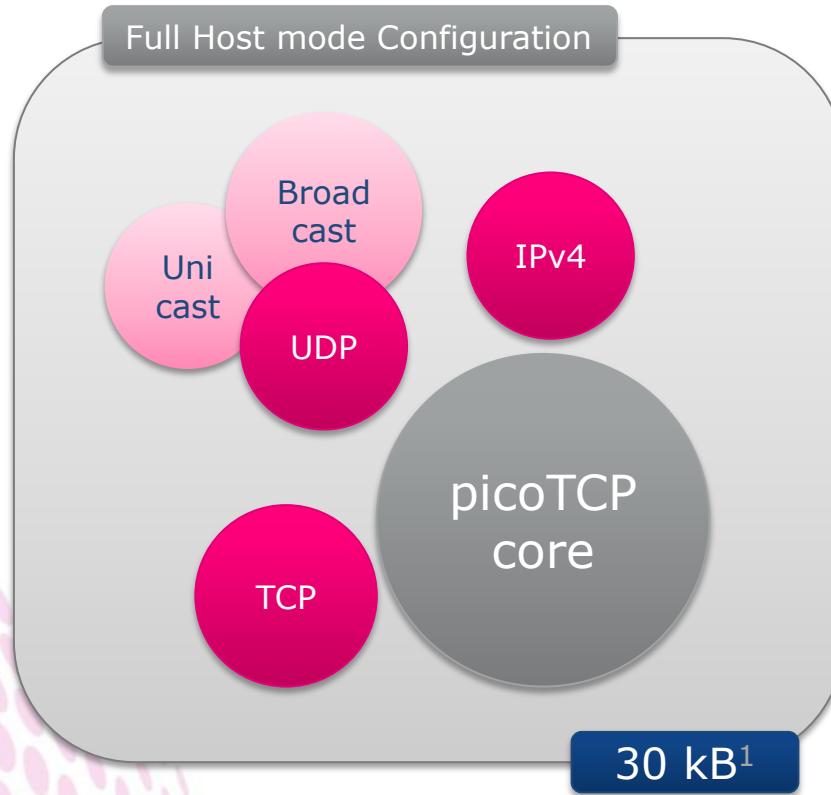


¹ Figures for ARM Cortex-M3, gcc compiler

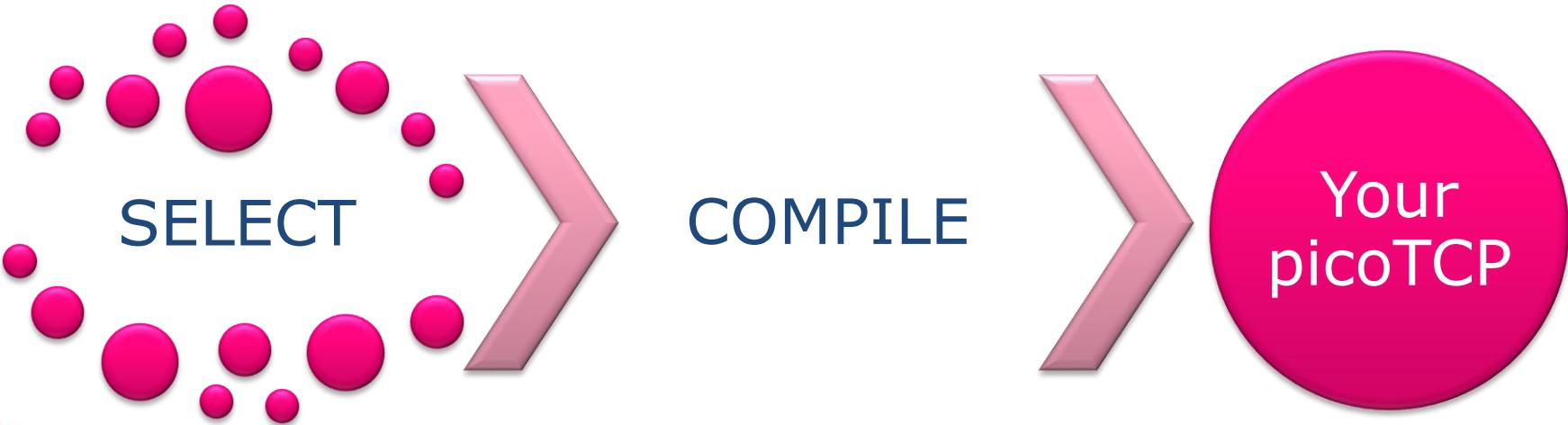
Modularity – Full Host Mode Config



Modularity– Full Host Mode Config



¹ Figures for ARM



```
$> make ARCH=stm32 CROSS_COMPILE=arm-none-eabi- \
    TCP=1 UDP=1 IPV4=1 IPFRAG=0 NAT=0 ICMP4=0 \
    MCAST=0 DEVLOOP=0 PING=0 DHCP_CLIENT=0 \
    DHCP_SERVER=0 DNS_CLIENT=0 IPFILTER=0 \
    CRC=0 HTTP_CLIENT=0 HTTP_SERVER=0 ZMQ=0 \
    OLSR=0 SLAACV4=0 STRIP=1 DEBUG=0
[CC] ...
[CC] ...
[AR] ./build/lib/libpicotcp.a
[RANLIB] ./build/lib/libpicotcp.a
[STRIP] ./build/lib/libpicotcp.a
[LIBSIZE] 30508      ./build/lib/libpicotcp.a
```

- CPU Architecture independent
- 8, 16, 32 & 64 bit
- Big or Little endian
- Bare Metal / Embedded OS / OS / RTOS
- 10+ different platforms already supported
- (RT)OS easily added, because of our OS Abstraction Layer

Supported architectures

- 32 bit
 - ARM variants:
 - LPC1768 (Cortex-M3)
 - LPC4357 (Cortex-M0 + Cortex-M4)
 - STM32 variants
- 16 bit
 - MSP430
 - PIC24Fxxxx
- 8 bit
 - AVR ATmega128

Supported Communication channels/chips

- BCM43362 (IEEE 802.11)
- MRF24WG (IEEE 802.11)
- LPC Ethernet ENET/EMAC (IEEE 802.3)
- Stellaris Ethernet (IEEE 802.3)
- Wiznet W5100 (IEEE 802.3)
- USB CDC-ECM (CDC1.2)
- Virtual drivers
 - TUN/TAP
 - VDE
 - libpcap

Supported RTOSes

- No OS / Bare metal
- FreeRTOS
- mbed-RTOS (CMSIS-RTOS compliant)
- Linux / POSIX

Our OS Abstraction Layer provides:

- Abstraction from mutexes/signals
- Abstraction from threads/processes
- Blocking socket interface

Performance measurements on MBED NXP LPC176x @ 100 MHz

Throughput



- TCP Tx: 10,09 Mbit/s
- TCP Rx: 20,23 Mbit/s



CPU time

- Idle state: 45 µs / loop
- Max throughput: 832 µs / loop

1100
1010
0101

RAM Usage

- Dyn. Memory: Min. 1kB Max. 8,3kB Avg. 4,7kB
- Static Memory: 16kB
- Stack: 2kB

Project: picotcp

- Quality oriented development environment

- Agile development (SCRUM)
- Test Driven Development
- Continuous Integration & Automated testing (Virtual testing is possible)
- Code Quality check: Tiobe Tics
Continuous improvement

- Full compliance with IETF TCP/IP standards

RFC 768 User Datagram Protocol (UDP)
 RFC 791 Internet Protocol (IP)
 RFC 792 Internet Control Message Protocol (ICMP)
 RFC 793 Transmission Control Protocol (TCP)
 RFC 816 Fault Isolation and Recovery
 RFC 826 Address Resolution Protocol (ARP)
 RFC 879 The TCP Maximum Segment Size and Related Topics
 RFC 894 IP over Ethernet
 RFC 896 Congestion Control in IP/TCP Internetworks
 RFC 919 Broadcasting Internet Datagrams
 RFC 922 Broadcasting Internet Datagrams in the Presence of Subnets
 RFC 950 Internet Standard Sub-netting Procedure
 RFC 1009 Requirements for Internet Gateways
 RFC 1034 Domain Names Concepts and Facilities
 RFC 1035 Domain Names Implementation and Specification
 RFC 1071 Computing the Internet Checksum
 RFC 1112 Internet Group Management Protocol (IGMP)
 RFC 1122 Requirements for Internet Hosts Communication Layers
 RFC 1191 Path MTU Discovery
 RFC 1323 TCP Extensions for High Performance
 RFC 1337 TIME-WAIT Assassination Hazards in TCP

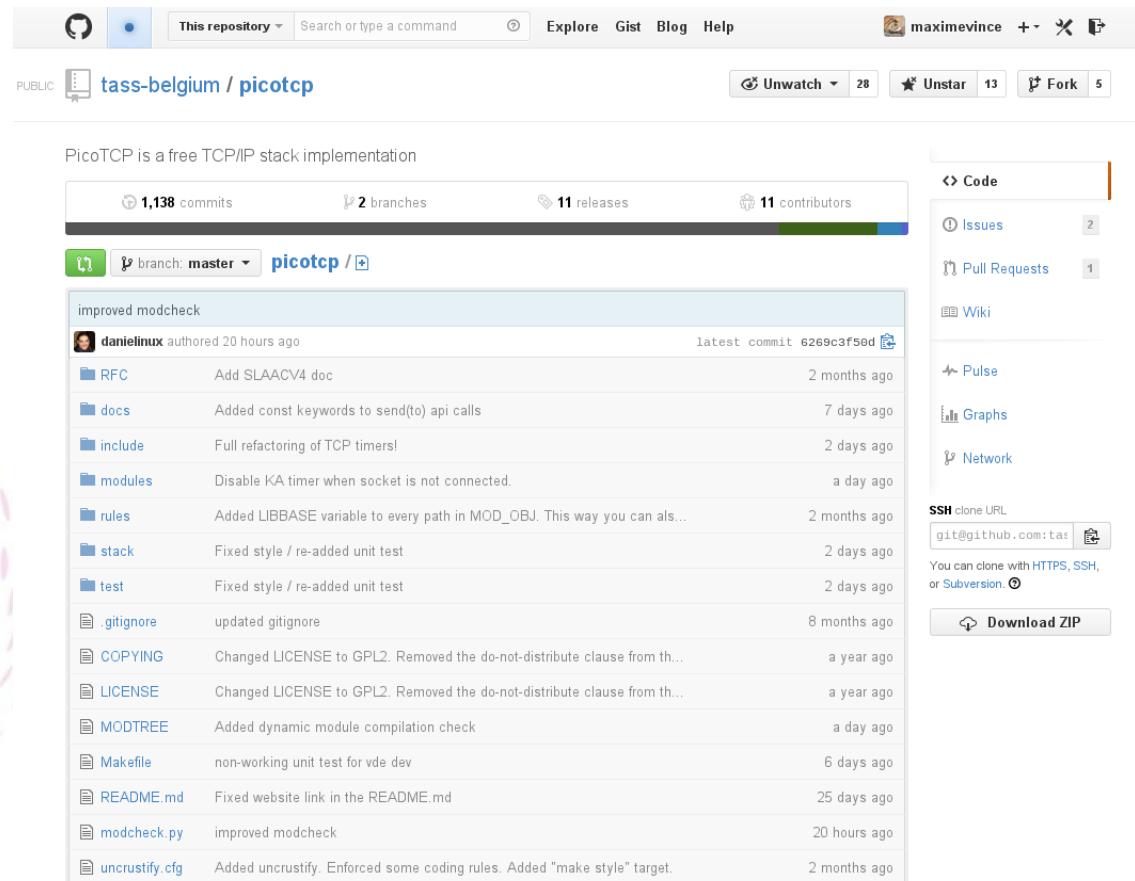
RFC 1534 Interoperation Between DHCP and BOOTP
 RFC 1542 Clarifications and Extensions for the Bootstrap Protocol
 RFC 1812 Requirements for IP Version 4 Routers
 RFC 1878 Variable Length Subnet Table For IPv4
 RFC 1886 DNS Extensions to Support IP Version 6 (1)
 RFC 2018 TCP Selective Acknowledgment Options
 RFC 2131 Dynamic Host Configuration Protocol (DHCP)
 RFC 2132 DHCP Options and BOOTP Vendor Extensions
 RFC 2236 Internet Group Management Protocol, Version 2
 RFC 2460 Internet Protocol, Version 6 (IPv6) Specification (1)
 RFC 2581 TCP Congestion Control
 RFC 2616 Hypertext Transfer Protocol - HTTP/1.1
 RFC 2663 IP Network Address Translator (NAT) Terminology and Considerations
 RFC 3042 Enhancing TCP's Loss Recovery Using Limited Transmit
 RFC 3315 Dynamic Host Configuration Protocol for IPv6 (DHCPv6) (1)
 RFC 3376 Internet Group Management Protocol, Version 3 (2)
 RFC 3517 A Conservative Selective Acknowledgment (SACK)-based Loss Recovery Algorithm for TCP
 RFC 3782 The NewReno Modification to TCP's Fast Recovery Algorithm
 RFC 4291 IP Version 6 Addressing Architecture (1)
 RFC 6691 TCP Options and Maximum Segment Size (MSS)

Code Quality	
picotcp master	TASS
Higher Quality	Measured by TiCS
A	C
B	
C	
D	
E	
F	
Lower Quality	
TIOBE Quality Indicator (based on TQI definition 1.1)	63.20%
Measurement performed: 30-Jan-2014	
Code Coverage	A B C D E F
Abstract Interpretation	A B C D E F
Code Complexity	A B C D E F
Compiler Warnings	A B C D E F
Coding Standards	A B C D E F
Code Duplication	A B C D E F
Fan Out	A B C D E F
Dead Code	A B C D E F

This product has been tested with the utmost care against the TIOBE Quality indicator definition. The definition can be found at www.tiobe.com.



- Community driven: We're on GitHub
 - Full source code freely available
 - Public issue-tracking system
 - GPLv2 licensed
 - Dual licensing possible



The screenshot shows the GitHub repository page for `tass-belgium / picotcp`. The page title is "PicoTCP is a free TCP/IP stack implementation". Key statistics displayed include 1,138 commits, 2 branches, 11 releases, and 11 contributors. A list of recent commits is shown, along with links to issues, pull requests, and other repository details.

Commit	Description	Date
danielinux	authored 20 hours ago	latest commit 6269c3f50d
RFC	Add SLAACV4 doc	2 months ago
docs	Added const keywords to send(to) api calls	7 days ago
include	Full refactoring of TCP timers!	2 days ago
modules	Disable KA timer when socket is not connected.	a day ago
rules	Added LIBBASE variable to every path in MOD_OBJ. This way you can als...	2 months ago
stack	Fixed style / re-added unit test	2 days ago
test	Fixed style / re-added unit test	2 days ago
.gitignore	updated gitignore	8 months ago
COPYING	Changed LICENSE to GPL2. Removed the do-not-distribute clause from th...	a year ago
LICENSE	Changed LICENSE to GPL2. Removed the do-not-distribute clause from th...	a year ago
MODTREE	Added dynamic module compilation check	a day ago
Makefile	non-working unit test for vde dev	6 days ago
README.md	Fixed website link in the README.md	25 days ago
modcheck.py	improved modcheck	20 hours ago
uncrustify.cfg	Added uncrustify. Enforced some coding rules. Added "make style" target.	2 months ago

- Ohloh says PicoTCP ...
 - has a young, but established codebase
 - is maintained by a large development team
 - with stable Y-O-Y commits
 - took an estimated 7 years of effort (COCOMO model)
 - starting with its first commit in October, 2012

Ohloh | BLACK DUCK Open HUB

PROJECTS PEOPLE ORGANIZATIONS BETA TOOLS CODE 

Follow @ Ohloh Ohloh Meta SIGN IN JOIN NOW

Projects Search... 

PicoTCP  1 

In a Nutshell, PicoTCP...

... has had 1,139 commits made by 23 contributors representing 27,638 lines of code

... is mostly written in C with a low number of source code comments

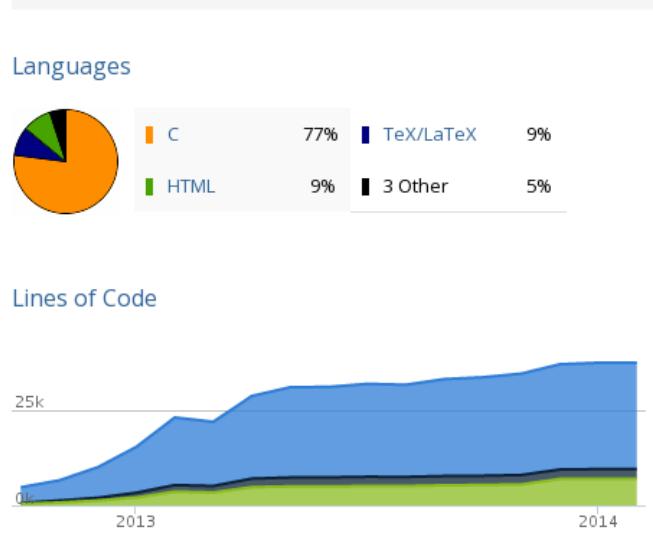
... has a young, but established codebase maintained by a large development team with stable Y-O-Y commits

... took an estimated 7 years of effort (COCOMO model) starting with its first commit in October, 2012 ending with its most recent commit 2 days ago

Languages

Language	Percentage
C	77%
HTML	9%
TeX/LaTeX	9%
3 Other	5%

Lines of Code



The chart shows a steady increase in lines of code over time. The y-axis represents lines of code in thousands, with major ticks at 0k and 25k. The x-axis represents years, with major ticks for 2013 and 2014. The data series is represented by a blue line with a light blue shaded area underneath, showing a consistent upward trend.

How to port to platform XYZ?

- Easy to port!
- You need:
 - **Memory management:**
 - Malloc/free functions
 - **Timing functions:**
 - Returning the amount of elapsed milliseconds / seconds
 - **A driver for your hardware**
 - Your way to the physical layer

- Memory management:
 - use malloc() and free() from your C-lib
 - Use memory allocation from your RTOS
 - Or our own memory manager which is on it's way
- Timing functions:
 - Timer ISR every millisecond, incrementing a 64-bit integer
 - PICO_TIME_MS() returning this integer as uint64_t
 - PICO_TIME() returning this integer / 1000 as uint64_t

```
extern volatile uint32_t lpc_tick;
extern volatile pico_time full_tick;

#define pico_free(x) free(x)

static inline void *pico_zalloc(size_t size)
{
    void *ptr = malloc(size);
    if(ptr)
        memset(ptr, 0u, size);
    return ptr;
}

static inline pico_time PICO_TIME_MS(void)
{
    if ((full_tick & 0xFFFFFFFF) > lpc_tick) {
        full_tick += 0x100000000ULL;
    }
    full_tick = (full_tick & 0xFFFFFFFF00000000ULL) + lpc_tick;
    return full_tick;
}

static inline pico_time PICO_TIME(void)
{
    return PICO_TIME_MS() / 1000;
}

static inline void PICO_IDLE(void)
{
    uint32_t now = lpc_tick;
    while(now == lpc_tick);
```

- Generic device driver API to PicoTCP:

- `struct pico_device * pico_dev_create(void)`
- `struct pico_device`
`->send = pico_dev_send;`
`->poll = pico_dev_poll;`
`->destroy = pico_dev_destroy;`
- `int pico_dev_send(struct pico_device *dev, void *buf, int len)`
- `int pico_dev_poll(struct pico_device *dev, int loop_score)`
- `void pico_dev_destroy(struct pico_device *dev)`

```
#ifndef PICO_DEV_MBED_H
#define PICO_DEV_MBED_H

#include "pico_config.h"
#include "pico_device.h"

void pico_mbed_destroy(struct pico_device *vde);
struct pico_device * pico_mbed_create(char *name);

#endif /* PICO_DEV_MBED */
```

```

struct pico_device * pico_mbed_create(char *name)
{
    uint8_t mac[PICO_SIZE_ETH];
    struct pico_device_mbed *mb =
        pico_zalloc(sizeof(struct pico_device_mbed));
    if (!mb)
        return NULL;
    ethernet_address(mac);
    if( 0 != pico_device_init((struct pico_device *)mb, name, mac))
    {
        dbg ("Loop init failed.\n");
        pico_loop_destroy(mb);
        return NULL;
    }

    mb->dev.send = pico_mbed_send;
    mb->dev.poll = pico_mbed_poll;
    mb->dev.destroy = pico_mbed_destroy;
    mb->bytes_left_in_frame = 0;

    if(ethernet_init() != 0) {
        dbg("Failed to initialize hardware.\n");
        pico_device_destroy((struct pico_device *)mb);
        return NULL;
    }

    dbg("Device %s created.\n", mb->dev.name);
    return (struct pico_device *)mb;
}

```

```

static int pico_mbed_poll(struct pico_device *dev, int loop_score)
{
    int len;
    struct pico_device_mbed *mb = (struct pico_device_mbed *) dev;

    while(loop_score != 0)
    { /* check for a new frame */

        if(mb->bytes_left_in_frame == 0)
        {
            if((len = ethernet_receive()) > 0)
                mb->bytes_left_in_frame = len;
        }

        if(mb->bytes_left_in_frame == 0)
            return loop_score; /* read a frame */

        len = ethernet_read((char *)buf), ETH_MAX_FLEN);

        if(len != mb->bytes_left_in_frame)
            return -1;

        dbg("MBed received %u new bytes\n",len);
        pico_stack_recv(dev, buf, len);

        loop_score--;
    }
    return loop_score;
}

```

```
static int pico_mbed_send(struct pico_device *dev, void *buf, int len)
{
    int ret, sent;
    struct pico_device_mbed *mb = (struct pico_device_mbed *) dev;

    if (len > ETH_MAX_FLEN)
        return -1;

    dbg("MBed sending %d bytes ...\\n",len);

    /* Write buf content to dev and return amount written */
    ret = ethernet_write((const char *)buf, len);
    sent = ethernet_send();
    dbg("MBed sent %d bytes\\n",ret);

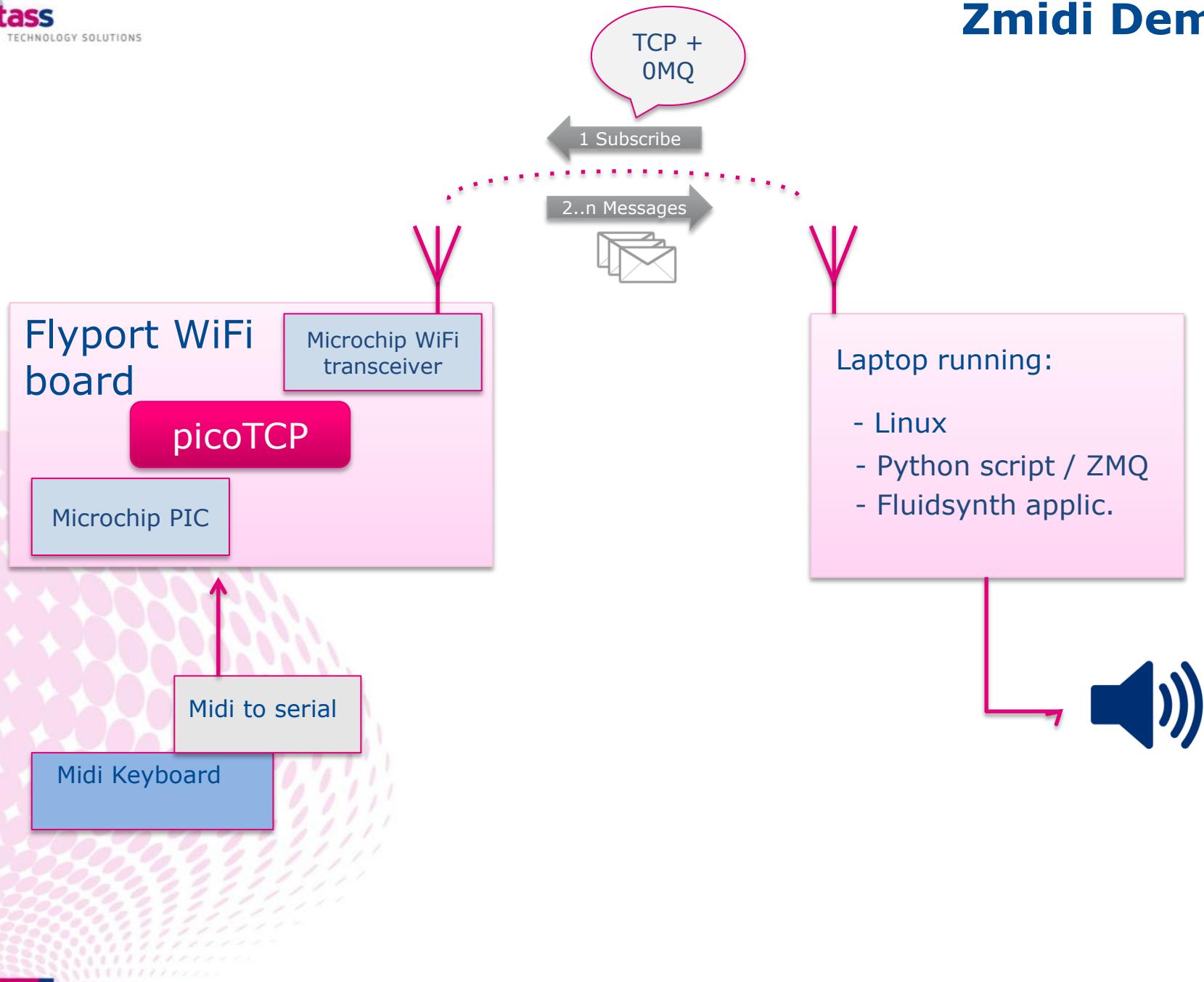
    if (len != ret || sent != ret)
        return -1;
    else
        return ret;
}
```

Demo Time!



Zmidi Demo





Check us out!



Check us out on GitHub:

<https://github.com/tass-belgium/picotcp.git>

