باسمه تعالی

گردآورنده و مترجم : علیرضا احتشام فر

موسسه آموزش عالی صدرالمتالهین

موضوع : سیستم عامل متن باز کونتیکی برای دستگاه های نسل بعدی اینترنت اشیا

کلمات کلیدی :

Contiki-NG

Internet of Things

Resource-Constrained Devices

چکیده :

کونتیکی نسل جدید یک سیستم عامل متن باز و چند پلتفرمی برای دستگاه های بی سیم تعبیه شده با محدودیت های شدید است.

تمرکز اصلی این فناوری بر روی ارتباطات و پروتکل های ایمن، قابل اعتماد و کم مصرف و در عین حال متکی به خود، مانند موارد زیر میباشد:

6LoWPAN,IPv6,6TISCH,RPL و CoAP .

اهداف اصلی این سیستم عامل، تسهیل نمونه سازی سریع و ارزیابی ایده های تحقیقاتی اینترنت اشیا، کاهش زمان لازم برای ورود نرم افزار های مرتبط با اینترنت اشیا به بازار و همچنین فراهم کردن یک پلتفرم قابل درک و ساده برای تدریس دوره های مرتبط با سیستم های تعبیه شده در آموزش عالی میباشد.

نسل جدید کونتیکی به عنوان یکی از انشعابات سیستم عامل کونتیکی شروع به کار کرد و بسیاری از امکانات اصلی آن را درون خود حفظ میکند.

در ادامه به دلیل و هدف ساخت نسل جدید کونتیکی (جدید ترین نسخه کنونی 4.7) و همچنین به بررسی تاثیرات آن توسط مثال های خاصی میپردازیم.

1. انگیزه و اهمیت

تحقیقات در زمینه شبکه های حسگر بیسیم تقریبا بیست سال پیش آغاز شد (1) . در سال های نخست، فعالیت های تحقیقاتی دستگاه های بسیار محدود از نظر منابع (کیلوبایت یا کمتر از آن روی رم، رابط نرم افزار برنامه نویسی سطح بیت) را هدف قرار داد، که استفاده از برنامه های بسیار ویژه مرتبط با کاربرد نرم افزار در آن زمان امری عادی بود. (2)

بعنوان مثالی از این رویکرد سیستم عامل TinyOS از یک مدل کاملا مبتنی بر رویداد و زبان برنامه نویسی انحصاری به نام نسک استفاده کرد.

1.1 تاریخچه Contiki

سیستم عامل کونتیکی (3) در سال 2006 برای عموم قابل استفاده شد، ولی از اوایل سال 2003 توسعه آن شروع شده بود. این سیستم عامل برای سنسور های بی سیم با امکانات محدود با حافظه کد 100 کیلوبایت و حافظه خالی کمتر از 10 کیلوبایت طراحی شده بود.
کونتیکی یک قدم مهم و بزرگ در راستای تحول سیستم عامل های اینترنت اشیا مدرن بود که در ادامه به نقاط قوت آن اشاره میکنیم :

استفاده از زبان برنامه نویسی استاندارد C; توسعه دهندگان دیگر نیازی به یادگیری زبان های سفارشی مانند nesC نداشتند.

هسته مبتنی بر رویداد. در ارتباط با کد حلقه اصلی خاص پلتفرم (برای برخی از پلتفرم ها اجرا شده است) کونتیکی زمان واکنش سریع به رویداد های خارجی و اجرای کارآمد انرژی برای اجرا را میسر مینمود.

همکاری چند رشته API مبتنی بر فرایند اجرا. کونتیکی با کاهش نیاز به قفل ها و ماشین های صریح تا حد زیادی برنامه نویسی را ساده کرد (ساده شدن در مقایسه با API مبتنی بر رویداد و رشته پیشگیرانه به ترتیب نگارش شده بیان گردیده).

پشتیبانی استاندارد اولیه برای پروتکل های شبکه مانند پروتکل اینترنت و IPV6از طریق پشته uIP .کونتیکی همچنین تعدادی از پیاده سازی های تازه تاسیس IPv6 روی شبکه های بیسیم کم جریان خصوصی و از پروتکل های روتینگ برای شبکه های کم توان Lossy پشتیبانی میکرد.

توزیع رسمی کونتیکی شامل Cooja نیز بود، که یک شبیه ساز برای دستگاه های شبکه ای در حال اجرا مبتنی بر فریم وورک کونتیکی (IEEE 802.15.4) میباشد.

کونتیکی در کنارCooja و دیگر ابزار های مرتبط با آن به ابزاری پرکاربرد به منظور تحقیقات در زمینه سنسور های بیسیم شبکه تبدیل شد.

به نشانه نفوذ کونتیکی، نویسندگان این مطلب بالای 2000 مقاله بررسی شده تحت عنوان کونتیکی و سیستم عامل را در دیتابیس های Scopus شناسایی کرده. که در میان این مقالات حدود 350 عدد از آنها در سال 2018 منتشر شده است! طبق گفته این منبع به یکی از اولین مقالات تحت عنوان کونتیکی بیش از 1300 بار اشاره شده و در مقالات دیگر از آن به عنوان مرجع استفاده شده است. (3)

1.2 از کونتیکی به کونتیکی نسل جدید

از زمانی که سیستم عامل کونتیکی گسترش پیدا کرد و موارد استفاده از آن گسترده تر شد محدودیت هایی آشکار شدند :

- عقبه بزرگی از پلتفرم های قدیمی با منابع بسیار محدود. میکروکنترلر های 8 و 16 بیتی کم مصرفی که در ابتدای کار کونتیکی برای آنها طراحی شده بود پس از گذشت زمان منسوخ شدند.

در حال حاظر دستگاه های Cortex-M با 32 بیت رم با حالت و مفهوم پیچیده تری از کم مصرف بودن برای همین امر مطرح هستند. (6)

- پشتیبانی از پروتکل های غیر استاندارد. هنگامی که تحقیقات در حوزه سنسور های بیسیم شبکه گسترش یافت و به یکی از تکنولوژی های کلیدی در اینترنت اشیا تبدیل شد، قابلیت همکاری و استاندارد ها به طرز زیادی دارای اهمیت شدند. درکنار استاندارد های سازگار پیاده سازی پروتکل ها، مدل کد محور کونتیکی قدیمی تر بنظر میرسید، بصورت تجربی، پروتکل های غیراستاندراد در ابتدا تحت عنوان مصنوعات تحقیقات تلقی میشدند. یکی از مثال های آن Rime Stack میباشد.

ترکیب پلتفرم قدیمی و پشتیبانی از آنها برای پروتکل های غیر استاندارد شبکه، سختی نگهداری و مدیریت را بیشتر کرد و مانع تکامل کدها و سیستم عامل شد.

نسل جدید کونتیکی برای اولین بار در نووامبر سال 2017 انتشار یافت. هدف آن از بین بردن بعضی از محدودیت های کونتیکی و همینطور آسان تر کردن نگهداری و سریعتر کردن امکان توسعه آن بود. اهداف نسخه اصلی آن عبارتند از :

- پروتکل های استاندراد . بعضی از پروتکل های استاندارد قابل استفاده و تحت پوشش توسط نسل جدید کونتیکی عبارتند از :

IEEE 802.15.4 TSCH, 6LoWPAN, 6TISCH, RPL, CoAP, MQTT ,LWM2M

- پشتیبانی از پلتفرم های سخت افزاری مدرن.

- قابل اعتماد بودن (خود محور و ایمن بودن) که اثبات این امر از طریق شیوه های توسعه مدرن ادغام مداوم به وسیله شبیه سازی و بستر فیزیکی آزمایشی و همینطور تکنیک های تست امنیتی انجام شد.

اولین نسخه منتشر شده (نسخه 4.0) یکی از زیرشاخه های سیستم عامل کونتیکی بود. در کنار تغییرات اساسی و متمرکز فوق الذکز سیستم عامل، کونتیکی نسل جدید پیکربندی و سیستم لوگینگ جدیدی اضافه کرد، یک آر پی ال جدید، کم حجم و قابل اتکا تحت عنوان RPL-Lite و یک پوسته مدیریتی شبکه جدید نیز به آن اضافه شد . نسل جدید کونتیکی همراه خود پاکسازی گسترده ای از کد های قدیمی، پلتفرم ها و پروتکل ها و سرویس ها بهمراه داشت. بدلیل جلوگیری از محدودیت در توسعه های آینده، تاریخچه تمامی تغییرات نسل جدید کونتیکی با چند کلیک به سادگی قابل مشاهده میباشند.

2. پروژه کونتیکی نسل جدید

کونتیکی نسل جدید شامل قابلیت های جدید زیادی است ولی در عین حال تعداد زیادی از قابلیت های سیستم عامل قدیمی کونتیکی را با یا بدون تغییرات نیز استفاده میکند، برای مثال زمان بندی، کرنل اتفاق محور، ساختار اعمال دستکاری اطلاعات کتابخانه ها و محل ذخیره.

نسل جدید کونتیکی علاوه بر موارد بالا با تغییرات کمی از تجهیزات نرم افزار های شبکه محور بسیاری نیز استفاده میکند. برای مثال از پیاده سازی های اورجینال 6LoWPAN , RPL

نسل جدید کونتیکی بطور خاص پلتفرم های Cortx-M را هدف قرار میدهد، نسخه اصلی آن پشتیبانی برای موارد زیر را ارائه میدهد:

Nordic Semi-Conductor, NXP, OpenMote, Texas Instruments, Zolertia

تمامی پلتفرم های فوق توسط M4- یا Crotex-M3 ساخته شده اند.

خارج از محل ذخیره اصلی تعداد بسیار زیادی شاخه های نسل جدید کونتیکی وجود دارند که به کمک سخت افزار های دیگر آمدند. برای مثال پلتفرم هایی که توسط چیپ های میکروالکترونیک ساخته شده اند، مانند آنهایی که در بستر آزمایش اینترنت اشیا به کار میروند.

با توجه به مطالعات و اندوخته های فعلی نمیتوان گفت چرا نسل جدید کونتیکی روی مدل های صفر کورتکس کار نکنند. در آخر باید گفت

نسخه رسمی از معماری 16 bit MSP430 نیز پشتیبانی میکند، که عموما داخل شبیه ساز Cooja بکار میرود.

در مورد افزودن کمک برای سخت افزار های دیگر، عمده تلاش ما تحت پیاده سازی معماری میکروکنترلر جدید و پیرامون چیپ میشود ، از جمله آن ها گیرنده های رادیویی میباشند. زمانی که این مورد اضافه شود، اتصال نسل جدید کونتیکی به یک برد جدید خیلی کمتر طاقت فرسا خواهد بود. (در صورت تمایل به کسب اطلاعات در این مورد، راهنمای مرحله به مرحله اتصال کونتیکی نسل جدید به یک سخت افزار جدید در ویکی پدیا یافت میشود.)

سیستم عامل نسل جدید کونتیکی خود را در دسته باقی سیستم های عامل برای دستگاه های تعبیه شده قرار داده است. برای مثال :

RIOT, ZEPHYR, Arm Mbed, Apache Mynewt, Tiny OS, FreeRTOS (7)

با پیاده سازی RPL-Lite و TSCH, RPL-Classic نویسندگان این مقاله حس میکنند نسل جدید کونتیکی جایگاه مش های شبکه با ظرفیت

محدود را حفظ میکند. در مقایسه همه جانبه کمی یا کیفی موارد فوق در رابطه با سیستم عامل های اینترنت اشیا، خواننده این مقاله را به مقالاتی که از قبل در خصوص اینترنت اشیا منتشر شده اند ارجاع میدهیم. (8و9)

به منظور اختصار، مابقی این بخش بر توصیف جنبه های فنی و غیر فنی، تکنولوژی هایی که کاملا جدید یا پس از انتشار پروژه نسل جدید کونتیکی دستخوش کمی تغییر شده اند تمرکز میکند، توصیف کامل و با جزئیات سیستم عامل هدف اصلی نویسنده این مقاله میباشد.

2.1 معماری و قابلیت های کونتیکی نسل جدید

در بیان کلی نسل جدید کونتیکی را میتوان به دو بخش کلی تقسیم کرد. بخش اول بدون سخت افزار و بخش دوم سخت افزار محور.

هاست های سابق قابل انتقال، کراس پلتفرم، قابل پیاده سازی کامپوننت های سیستم عامل شامل کارنل، زمان بندی های نرم افزار، معماری کتابخانه های اطلاعات محور و پروتکل های شبکه بودند.

در حالت دیگر برای ادامه کار سیستم عامل بر روی دستگاه های دیگر نیاز به دانش کد زنی داریم. این مورد شامل درایور برای کامپوننت های سخت افزاری ماند تایمر، رابط های رادیویی و باقی لوازم جانبی رو یا خرج از برد میشود. برای مثال :

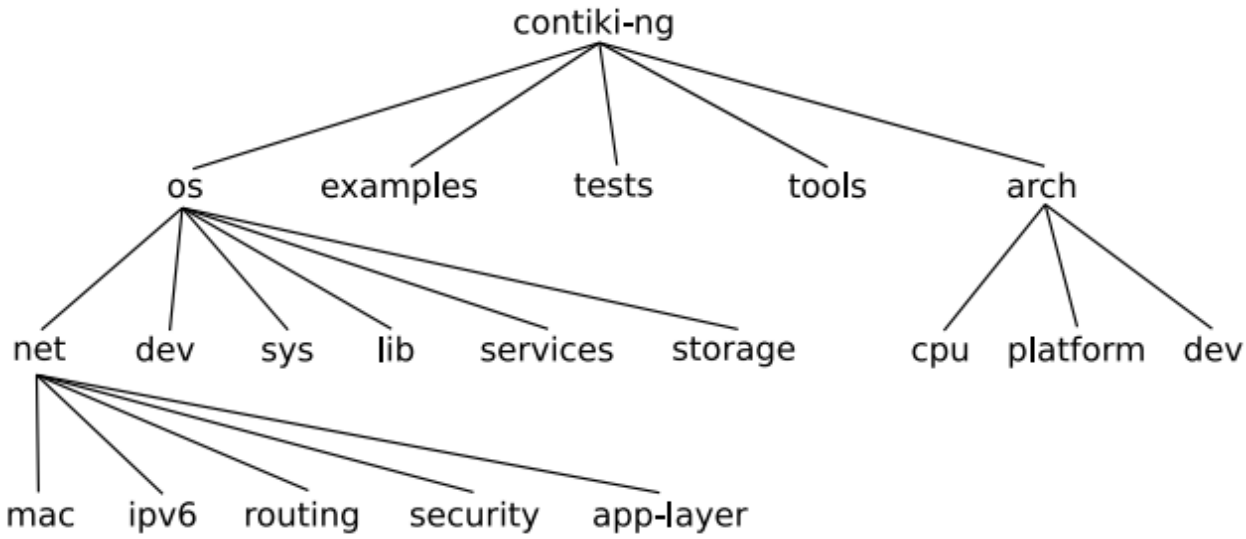
Universal Asynchornous Receiver/Transmitter (UART0), Serial Peripheral interface (SPI), Inter-integrated Circuit (12c), LEDs, User buttons and sensing elements

برای اینکه افزودن پشتیبانی و تحت شعاع قرار دادن دستگاه های بیشتر به راحتی هر چه تمام تر انجام شود و برای سهل الانتقال بودن کد ها، نسل جدید کونتیکی لایه های انتزاعی سخت افزاری برای رابط های ساده تعریف میکند، از جمله ورودی/خروجی همه منظوره و SPI.

یک لایه انتزاعی سخت افزاری برای 12C بخشی از یک نقشه راه کوتاه است. این لایه های انتزاعی سخت افزار، عملیات های سخت افزار محور برای این پورت جدید سخت افزار را که توسط برنامه نویس اجرا شده اند را اعلام میکند. این امر به برنامه نویسان جدید الورود امکان تمرکز بالاتر روی اجرا کردن بخش های سخت افزار محور API های شناخته شده کنند، بدون اینکه زمان خود را صرف تدبیر اندیشی برای رابط های جدید برنامه نویسی کنند. زمانی که پیاده سازی های چیپ محور توسعه یافت، اضافه کردن پشتیبان برای پلتفرم های مختلف روی چیپ های مشابه، دیگر به سختی گذشته نبود و برای این امر باید پیکربندی ناچیزی انجام میدادیم. در میان لایه های انتزائی سخت افزار هر جایی که امکانش وجود داشته باشد، نسل جدید کونتیکی فانکشن هایی مستقل از پلتفرم ارائه میکند که با استفاده از آنها ما این امکان را داریم که به المان های سخت افزاری خود دسترسی داشته باشیم. قابلیت مذکور بر روی تمامی پلتفرم های تحت پوشش سیستم عامل بدون زحمت اضافه قابل استفاده میباشد. برای مثال یک برنامه نویس میتواند()Spi_transfer را استفاده کند تا یک سکانس از بایت ها را به یک رابط پیرامونی ارسال نماید. این عملیات روی تمامی پلتفرم های سخت افزاری که بخش های سخت افزار محور رابط پیرامونی پشت سر هم را پیاده سازی میکنند یک رفتار شناخته شده است.

2.1.1 ساختار محل ذخیره (Repository)

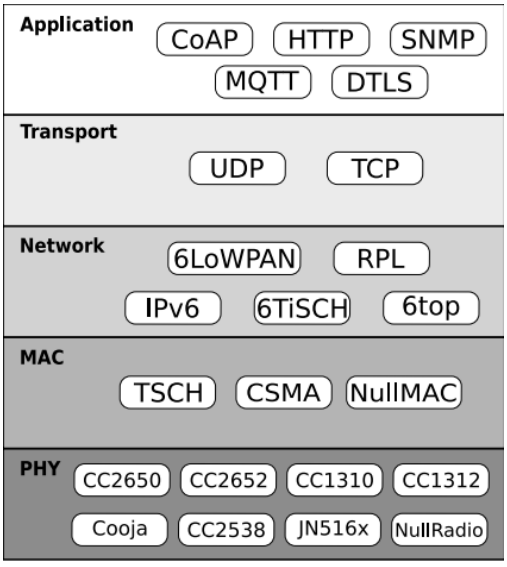
تصویر شماره 1 فهرست راهنمای ساختار پایگاه کد نسل جدید کونتیکی را به نمایش میگذارد. تمامی کد های پلتفرم محور زیر /os و تمامی درایور های سخت فزار محور زیر /Arch * قابل مشاهده هستند، فهرست راهنمای تست ها* ادغام تست سوئیت های مداوم را ذخیره میکند در حالیکه ابزار ها * خدمات و آپشن های کمکی مانند اکسریپت های استفاده شده برای آپلود سیستم عامل برای دستگاه های پشتیبانی شده و ایجاد مستندات را در خود نگهمیدارد.



تصویر 1

2.2 پشتیبانی شبکه

همانگونه که در بخش 1.2 اشاره کردیم، یکی از اهداف اصلی نسل جدید کونتیکی این است که برای دستگاه های بیسیم جاسازی شده با محدودیت های شدید منابع، خدمات سازگار با استاندارد و کار شبکه ای کم قدرت ارائه کند. در تصویر 2 به پشته شبکه ای نسل جدید کونتیکی اشاره میکنیم.



تصویر 2

در ادامه این بخش، یک اشاره کلی به اجرای پروتکل های منتخب پشتیبانی شده و مشخصات فنی نسل جدید کونتیکی میکنیم.

TSCH & 6TISCH 2.2.1

زمان شکاف کانال پرش یا به اختصار (TSCH) یک لایه اکسس کنترل (MAC) تعریف شده در استاندارد IEEE 802.15.1-2015امیباشد. (10) هدف این تکنولوژی اینترنت صنعتی اشیا و مواردی است که قابلیت اطمینان بیشتر، تاخیر و انرژی کمتری را خواهان هستند.
(11) 6TISCH ترکیبی از IETF استاندارد های موجود یا جدیدی هستند که هدف آنها تعریف کامل یک TSCH که IPV6 فعال در شبکه برای استفاده در موارد فوق دارند میباشد. کونتیکی نسل جدید، TSCH، IPv6 را بیشتر از TSCH IEEE 802.15.4e با راه اندازی های حداقلی پشتیبانی میکند. (10) (در نسخه اصلی مقاله، مقالات مشابه دیگری اشاره شده و توضیحات تکمیلی بابت مثال ها داده نشده، برای اختصار و قابل فهم بودن اشاره های مذکور را در خط بعد خواهیم داشت و شماره مقالات را برای عزیزانی که تمایل به کسب اطلاعات بیشتر در این زمینه دارند قرار میدهیم.)
(6tisch) minimal configuration (12), the 6top protocol (6p) (13), RPL (14), THE TISCH AND 6TISCH implementations (15), Orchestra (16), the minimal scheduling function (MSF) (17)
ارکستر یک تابع زمان بندی کاملا مستقل است که بدون نیاز به ترافیک سیگنالینگ میتواند پیکربندی لینک های TSCH را انجام دهد. از سوی دیگر MFS از 6P برای تخصیص های TSCH استفاده میکند و برنامه آن را طبق تغییرات ترافیکی تغییر میدهد.

RPL-Classic & RPL-Lite 2.2.2

RPL پروتکلی است که توسط IETF RFC6550 برای مسیریابی بر روی شبکه های کم مصرف و با تلفات تعریف شده است. به طور خلاصه، گره ها یک توپولوژی گراف غیر چرخشی جهت دار چند جهشی (DAG) می سازند، که مسیریابی را به سمت ریشه (در امتداد یک گرادیان) یا به سمت هر گره دیگری (به دنبال جدول های مسیریابی یا از طریق مسیریابی منبع) امکان پذیر می‌کند. سیستم‌عامل Contiki یکی از اولین پیاده‌سازی‌های RPL باز را در سال 2010 ارائه کرد («RPL-Classic»)، که نشان داده شد با نسخه RPL توزیع‌شده با TinyOS قابل همکاری است. Contiki-NG این پیاده سازی را پذیرفت و نسخه جدیدی به نام "RPL-Lite" ارائه کرد. RPL-Lite به دو چیز دست می یابد:

(i) تنها یک زیرمجموعه انتخابی از حالت‌های عملکرد مربوطه را از استاندارد بسیار انعطاف‌پذیر، بر اساس سال‌ها تجربه RPL-Classic حفظ می‌کند،

و (ii) یک فاکتورگیری مجدد کامل از یک مبنای کد ارائه می‌دهد که مقدار قابل توجهی بدهی فنی در طول سال‌ها انباشته شده بود.

به این ترتیب، RPL-Lite تنها از یک DAG واحد در یک زمان، یک «نمونه RPL» و تنها حالت غیر ذخیره‌سازی عملکرد را پشتیبانی می‌کند. این انتخاب‌ها مقدار حالت حفظ شده در هر گره محدود در شبکه را به حداقل می‌رساند و امکان عملکرد قوی‌تر را فراهم می‌کند. RPL-Lite به موازات تحقیقات در مورد RPL فوق‌العاده قابل اعتماد پیاده‌سازی شد و از بسیاری از مکانیسم‌های قابلیت اطمینان که به عنوان بخشی از این تحقیق ابداع شده‌اند بهره می‌برد.

2.2.3. پشتیبانی چندپخشی

هسته Contiki-NG از ارسال چندپخشی IPv6 در LoWPAN6 از طریق یک API پشتیبانی می‌کند که امکان افزودن آسان موتورهای ارسال چندپخشی جدید را فراهم می کند.

Contiki-NG به اجرای استانداردهای پروتکل چندپخشی برای شبکه های کم توان و با اتلاف (MPL) کمک می‌کند: یک پروتکل ارسال چندپخشی پیشنهاد شده توسط گروه ویژه مهندسی اینترنت (IETF). پشتیبانی MPL دو موتور چندپخشی را که از سیستم عامل اصلی Contiki اقتباس شده اند، همراهی می‌کند: (i) ارسال بدون حالت چندپخشی (SMRF) RPL و (ii) SMRF پیشرفته (ESMRF).

2.2.4. CoAP و LWM2M

پروتکل برنامه محدود (CoAP) یک پروتکل لایه کاربردی مشابه HTTP است، اما به گونه ای طراحی شده است که برای محیط های محدود مناسب تر باشد. پیاده‌سازی CoAP Contiki-NG از بسیاری از ویژگی‌های کلیدی CoAP پشتیبانی می‌کند، از جمله: (i) انتقال‌های بلوکی برای انتقال بلوک‌های بزرگ داده که نمی‌توانند در یک بسته واحد قرار بگیرند، و (ii) مشاهدات CoAP. پیاده‌سازی و API مربوطه اضافه کردن منابع CoAP جدید ثبت‌شده در یک مسیر خاص و همچنین کنترل‌کننده‌های منابع پیچیده‌تر را که می‌توان در تمام درخواست‌های CoAP فراخوانی کرد، آسان می‌کند. پیاده سازی CoAP با libcoap قابل همکاری است. ابزارهای خط فرمان، و همچنین با node-coap قابلیت همکاری با اولی به صورت خودکار به عنوان بخشی از گردش کار CI اقدامات GitHub ما (بخش 2.5) برای جلوگیری از رگرسیون آزمایش می‌شود. این پیاده‌سازی همچنین با افزونه‌های مرورگر «کروم» و « برای کروم «(Cu4Cr) قابل همکاری است. Contiki-NG همچنین از نسخه 1.0 مشخصات Open Mobile Alliance (OMA) Lightweight M2M (LWM2M) با فرمت‌های داده ساده، نمادگذاری شی جاوا اسکریپت (JSON) و Type-LengthValue (TLV) پشتیبانی می‌کند. این پیاده سازی از حالت امنیتی LWM2M با کلیدهای از پیش مشترک و سرور و اشیاء امنیتی برای پیکربندی امنیت پشتیبانی می‌کند. پیاده سازی موتور LWM2M با اجرای اشیاء LWM2M IP" برای اشیاء هوشمند" (IPSO) همراه است، که به عنوان نمونه ای از نقطه شروع توسط توسعه دهندگانی که مایل به استفاده از LWM2M برای کار خود هستند، استفاده می شود. این پیاده سازی با Eclipse Leshan و همچنین Eclipse Wakama قابل همکاری است. قابلیت همکاری با Leshan نیز به عنوان بخشی از گردش کار CI ما آزمایش می شود (بخش 2.5). سیستم عامل اصلی Contiki از CoAP و همچنین LWM2M پشتیبانی می‌کند، اما هر دو پیاده‌سازی توسط پروژه Contiki-NG به‌طور گسترده بازسازی و تکامل یافته‌اند.

2.2.5. MQTT

Contiki-NG دارای یک پیاده‌سازی مشتری سبک از Message Queuing Telemetry Transport (MQTT)، یک پروتکل باز برای انتشار/اشتراک است. این پیاده سازی از نسخه های MQTT 3.1 (اقتباس شده از سیستم عامل Contiki) و نسخه 3.1.1 (مشارکت توسط ContikiNG) پشتیبانی می کند. پشتیبانی از MQTTنسخه 5 که قبلاً در شاخه توسعه ادغام شده است و در نسخه بعدی گنجانده خواهد شد. Contiki-NG همچنین یک نمونه مشتری MQTT مستقل از پلتفرم را ارائه می‌دهد که با کارگزار Eclipse Mosquitto MQTT و همچنین با پلتفرم IBM Watson IoT قابل همکاری است. قابلیت همکاری با Mosquitto به عنوان بخشی از گردش کار CI ما (بخش 2.5) برای جلوگیری از رگرسیون آزمایش می‌شود.

2.2.6. محدودیت‌ها

در حال حاضر، Contiki-NG به طور رسمی از موارد روبرو پشتیبانی نمی کند: (i) حالت گوش دادن نمونه هماهنگ IEEE (CSL) 802.15.4. (ii) 6LoWPAN Neighbor Discovery. (iii) عملیات "Mesh-Under6LoWPAN" (فقط "Route-Over" پشتیبانی می شود). و (IV) پیام های ایمن در RPL. پیاده‌سازی متن‌باز برای Contiki-NG در مخازن آینه‌ای وجود دارد و افزودن پشتیبانی رسمی یکی از جاه‌طلبی‌های بلندمدت این پروژه است.

2.3. افشای مسئول و مشاوره های امنیتی

Contiki-NG تمرکز خود را بر امنیت نرم افزار از طریق تست فازی و روش های دیگر افزایش می‌دهد. این پروژه دارای یک آدرس ایمیل اختصاصی است که می تواند برای افشای مسئولیت پذیری آسیب پذیری‌های امنیتی استفاده شود. در نتیجه فرآیندهای آزمایش داخلی و گزارش های جامعه، این پروژه اخیرا اولین توصیه های امنیتی خود را منتشر کرد.

2.4. مستندات

Contiki-NG یک بانک اطلاعاتی ذخیره شده در GitHub است که شامل فهرست گسترده‌ای از راهنماها و آموزش ها برای مبتدیان و همچنین برای کاربران پیشرفته تر میباشد. علاوه بر این، سورس کد نسل جدید کونتیکی با نظرات Doxygen حاشیه‌نویسی شده که برای تولید اسناد API مبتنی بر HTML استفاده می‌شود. اسناد API را می‌توان توسط کاربران به صورت لوکال در رایانه هایشان ساخته و مشاهده کرد، اما در عین حال به صورت آنلاین در "Read the Docs" نیز قابل دسترسی میباشد. این فضای آنلاین که بطور خودکار به روز می شود، چندین نسخه از اسناد API را در خود نگهمیدارد که شامل : یک نسخه از تمای نسخه های منتشر شده ی نسل جدید کونتیکی (از نسخه 4.2 که این ویژگی برای اولین بار معرفی شد)، و همچنین یک نسخه از آخرین انتشار و آپدیت توسط برنامه نویسان.

2.5. تست یکپارچگی پیوسته

کونتیکی نسل جدید به طور خودکار با استفاده از یکپارچه سازی مداوم یا به اختصار (CI) در GitHub Actions بصورت مداوم آزمایش می‌شود. این گردش کار و نحوه انجام آزمایشات با انتشار نسخه v4.6 جایگزین مجموعه آزمایشی قدیمی تر در پلتفرم Travis-CI16 شد. هر بار که یک تغییر کد در یکی از شاخه‌های git اصلی نسل جدید کونتیکی اعمال می‌شود و یا یک درخواست pull باز یا آپدیت میشود، مجموعه آزمایشی بطور خودکار فعال می‌شود. همه pull request ها باید تمام تست‌های CI را پشت سر بگذارند تا بتوانند در سورس کد اصلی در نظر گرفته شوند. تمام تغییرات کد منبع باید از طریق یک pull request انجام شود و این مورد برای تیم اصلی نگهداری نسل جدید کونتیکی نیز صدق میکند و آنها نیز از این قاعده مستثنی نیستند. این استراتژی امکان بررسی دقیق توسط همتایان را اعمال می کند، بنابراین کیفیت کلی کد را افزایش می‌دهد و احتمال ایجاد خطا را کاهش می‌دهد. مجموعه تست نسل جدید کونتیکی شامل هفده کار است که هر کار چندین تست CI را اجرا می‌کند. مجموعه آزمایشی المان کدهای زیر را پوشش می دهد:

• کامپایل موفقیت آمیز نمونه های کد برای پلتفرم‌های سخت افزاری متعدد تحت چندین پیکربندی مختلف.

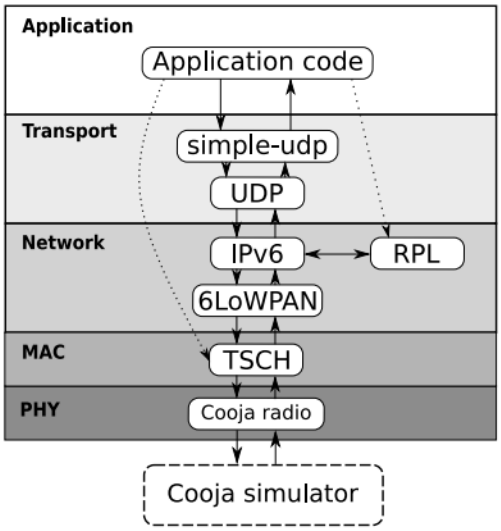
• اعتبارسنجی عملکرد صحیح عناصر مختلف زیرسیستم شبکه از طریق شبیه سازی‌های متعدد Cooja و سناریوهای اجرای کد محلی.

• گردآوری موفقیت آمیز مستندات API doxygen (بخش 2.4 به آن اشاره کردیم).

کاربرانی که می‌خواهند کد خود را با گردش کار Contiki-NG CI اعمال کنند، می‌توانند با فعال کردن GitHub Actions در مخزن فورک خود، این کار را به‌طور ساده انجام دهند. از سوی دیگر، تصویر داکر Contiki-NG همه ابزارهای مورد نیاز برای اجرای مجموعه آزمایشی به صورت لوکال را در رایانه‌هایشان فراهم می‌کند.

2.6. ساخت‌های شبانه

علاوه بر CI که به ازای مشارکت انجام میشود و در بالا توضیح داده شد، ما اجرای خودکار بستر آزمایشی شبانه را روی سخت‌افزار واقعی انجام می‌دهیم: یک بستر آزمایشی سفارشی با 25 گره که در RISE SICS نصب شده است. هر گره هم از یک دستگاه کنترل (Raspberry PI) و هم مجموعه ای از حسگرها (در زمان نوشتن، Zolertia Firefly و دانگل JN516x) تشکیل شده است. هر شب، یک cron job چهار آزمایش 2 ساعته را برنامه ریزی می‌کند که هر کدام با پیکربندی متفاوتی از استک شبکه هستند. گره های کنترلی در تمام ۲۵ دستگاه Firefly تبدیل به یک سیستم افزار مستقل میشوند که ارتباط پاسخ-درخواست را بین گره ریشه و هر گره دیگر در شبکه انجام می‌دهد. هر آزمایش شامل بیش از 10 هزار مبادله بسته ای در شبکه رفت و برگشت بر روی چند جهش با استفاده از دستگاه‌های صرفه جویی در انرژی است. یک اسکریپت پس از پردازش کردن یک فایل گزارش، معیارهای کلیدی را از هر اجرا استخراج می‌کند: نسبت تحویل رفت و برگشت سر به سر، تأخیر، چرخه وظیفه رادیویی، تعداد پرش و فرکانس تغییرات توپولوژی شبکه از آن موارد هستند. فایل‌های گزارش خام و آمار پردازش شده به طور خودکار به یک وب سایت عمومی برای تجسم و نظارت فرستاده می‌شوند. مزایای ساخت شبانه دوچندان است. در مرحله اول، آنها برخلاف شبیه سازی/همانندسازی مورد استفاده در CI، تست خودکار را بر روی سخت افزار واقعی ارائه می‌دهند. ثانیا، آنها به توسعه دهندگان بازخورد می‌دهند و توانایی مقایسه عملکرد پیکربندی‌های مختلف پشته شبکه و تشخیص کاهش عملکرد را نیز دارند.



تصویر 3

3. مثال گویا

برای نشان دادن یک مثال کاربردی نسل جدید کونتیکی، ما نسخه دمو هماهنگ سازی زمان در سطح شبکه را انتخاب کردیم. هدف از انتخاب این مثال این است که نشان دهیم یک توسعه دهنده نرم افزار چگونه میتواند از هماهنگ کننده ی زمان که توسط TSCH ارائه شده برای تعیین تاخیر بین بسته منشا توسط یک منبع شبکه و زمان هندل و پردازش تا node مقصد استفاده کند، که این موضوع به عنوان زمان پردازش سورس شبکه نیز حساب میشود. این مثال بدلیل حفظ سادگی کد های نرم افزاری در زمان نشان دادن چندین رابط انتخاب شده است.(تصویر 3)، برای درک بهتر این موضوع به خطوطی که در ادامه داریم دقت کنید:

پردازش اولیه کونتیکی نسل اول (خطوط 23-27)

```
#include "contiki.h"                                                    1
#include "net/ipv6/simple-udp.h"                                        2
#include "net/mac/tsch/tsch.h"                                          3
#include "lib/random.h"                                                 4
#include "sys/node-id.h"                                                5
                                                                        6
#define UDP_PORT                8765                                    7
#define SEND_INTERVAL          (60 * CLOCK_SECOND)                      8
                                                                        9
PROCESS(node_process, "RPL Node");                                      10
AUTOSTART_PROCESSES(&node_process);                                     11
simple_udp_callback rx_callback; /* Defined in the file                 12
    rx_callback.c */
                                                                        13
PROCESS_THREAD(node_process, ev, data)                                  14
{                                                                       15
  static struct simple_udp_connection udp_conn;                         16
  static struct etimer periodic_timer;                                  17
  uip_ipaddr_t dst;                                                     18
                                                                        19
  PROCESS_BEGIN();                                                      20
                                                                        21
  /* Initialization; 'rx_callback' is the function for                  22
     packet reception */
  simple_udp_register(&udp_conn, UDP_PORT, NULL, UDP_PORT               23
     , rx_callback);
  etimer_set(&periodic_timer, random_rand()   if(node_id                24
     == 1) {  /* Running on the root? */
    NETSTACK_ROUTING.root_start();                                      25
  }                                                                     26
                                                                        27
  /* Main loop */                                                       28
  while(1) {                                                            29
    PROCESS_WAIT_EVENT_UNTIL(etimer_expired(&                           30
     periodic_timer));
    if(NETSTACK_ROUTING.node_is_reachable()                             31
       && NETSTACK_ROUTING.get_root_ipaddr(&dst)) {                     32
      /* Send network uptime timestamp to the network                  33
     root node */
      uint64_t network_uptime =                                         34
     tsch_get_network_uptime_ticks();
      simple_udp_sendto(&udp_conn, &network_uptime,                     35
     sizeof(uint64_t), &dst);
    }                                                                   36
    etimer_set(&periodic_timer, SEND_INTERVAL);                         37
  }                                                                     38
                                                                        39
  PROCESS_END();                                                        40
}                                                                       41
```

انتظار برای اتفاقات روی تایمر (خط 31)

ارسال بسته های شبکه (خط 36)

پذیرش بسته ها (در سورس کد کامل با عملکرد مربوطه پروتوتایپ نشان داده شده، خط 12)

و در آخر اینکه چگونه با روتینگ و پروتکل های TSCH به وسیله لایه نرم افزاری تعامل داشته باشیم.

این مثال تعداد زیادی از فانکشن های پیشرفته و اساسی نسل جدید کونتیکی را در خود پنهان کرده، بعضی از جزئیات در خصوص اینکه نسل جدید کونتیکی چگونه بسته های شبکه را پردازش میکنند در تصویر 3 آمده است. اگر نحوه نگرش خود را صرفا به ارسال و دریافت ترافیک تغییر دهیم، نرم افزار تنها یک رابط مستقیم با لایه انتقالات توسط simple_udp_API دارد (منظور ما در این مثال خاص پیاده سازی UDP است). ولی اپلیکشن یک ارتباط غیرمستقیم با RPL و TSCH نیز دارد.

در مورد قبل، NETSTACK_ROUTING API وظیفه تشخیص اینکه آیا دستگاه قبل از انجام عملیات های انتقال به درستی در شبکه پیاده سازی شده را بر عهده دارد. در مورد بعدی tsch_get_network_uptime_tiks() وظیفه دریافت دقیق اطلاعات زمانی را دارد.

در مسیر ریسیور با توجه به پیکربندی پشته شبکه، پذیرش فریم ها یا از طریق وقفه ها و یا از طریق کشش های انجام گرفته توسط رادیو درایور داخل بازه های زمانی TSCH RX انجام میگیرد. در روش قدیمی پردازش های وقفه محور، وقفه ها توسط فانکشن های پردازشگر با معماری خاص انجام میگیرند. برای رادیو هایی که روی چیپ سیستم سوار میشوند، زمانی یک وقفه رادیویی خاص اتفاق میافتد که یک فریم بطور کامل و بدون هیچگونه خطایی دریافت شده باشد. برای رادیو های SPI، پردازنده ی قاب یک وقفه ی GPIO ایجاد میکند. وقفه ای متناظر کنترل کننده عملا پردازش اصلی رادیو درایور را میگیرد و در لحظه برمیگرداند، به بیان دیگر زمان صرف شده توسط کد درحال اجرا را

داخل یک وقفه متنی کوتاه میکند. پردازش رادیو درایور متعاقبا بیرون از وقفه فراخوانده شده، از بافر سخت افزاری رادیو خوانده میشود و داخل رم اصلی در کنار ویژگی های دریافت کننده فریم مربوطه مانند ناشنگر قدرت سیگنال دریافتی یا به اختصار (RSSI) قرار میگیرد .

در صورتی که رادیو TSCH غیرفعال باشد. با استفاده از NETSTACK_RADIO API در زمان صحیح در طول بازه زمانی پردازش، پیاده ساز مستقل از پلتفرم TSCH رادیو درایو را از از فریم ها میکشد. بدین ترتیب وظیفه دقت در زمان بندی را بر عهده TSCH میباشد.

فریم ها در حالیکه به سمت استک های شبکه حرکت میکنند توسط پیاده ساز مورد بحث پروتکل پردازش میشوند و کد های اپلیکیشن توسط یک قابلیت پاسخ به تماس از این امر مطلع میشوند.

زیر بخش exaples/ کونتیکی تعداد بسیار زیادی پروژه به عنوان مثال استفاده کرده که همگی را میتوان به عنوان قدم های اولی با کونتیکی نسل جدید استفاده کرد یا در صورت تمایل بعضی کاربران از آنها برای نقطه شروعی بابت ساخت اپلیکشن های خودشان استفاده کنند.

سه مثال تمامی المان های پشته (stack) شبکه و تمامی انتزاعات سخت افزاری را در این مقاله توضیح میدهند. برای امتحان کردن مثال بیان شده در این بخش یا باقی مثال های ارائه شده، ما استفاده از تصویر با لینک زیر (در خصوص محل نگهداری داکر نسل جدید کونتیکی میباشد) و در کنار آن کمک گرفتن از ویکی پدیا را پیشنهاد میکنیم. این تصویر تمامی کامپایلر ها و ابزار های مورد نیاز را در خود دارد و همچنین شامل شبیه ساز Cooja نیز میباشد که امکان انجام آزمایشات روی کونتیکی نسل جدید را بدون نیاز به سخت افزار های پیشنیاز میدهد.(https://hub.docker.com/r/contiker/contiki-n)

تاثیرات پس از انتشار

از زمان انتشار منبع باز آن در سال 2006، سیستم عامل اصلی Contiki توسط پروژه‌های تحقیقاتی متعددی که توسط سازمان‌های مختلفی تامین می‌شود، استفاده شده است، به عنوان مثال: کمیسیون اروپا (Ec) تحت لیسانس Horizon 2020 (H2020)، Horizon Europe نیز مانند برنامه های چارچوب قبلی، نهادهای مالی مختلف ملی تحقیقات، شورای تحقیقات مهندسی و علوم فیزیکی بریتانیا EPSRC یا بنیاد دانش سوئد.

ارزش افزوده و مزایای Contiki-NG را می توان به شرح زیر خلاصه کرد:

• انتقال ساده به پلتفرم های سخت افزاری جدید به دلیل حذف کدهای قدیمی و به دلیل راه اندازی اولیه سیستم جدید مستقل از پلت فرم، کد حلقه اصلی و HAL ها.

• توسعه ساده ویژگی های جدید به دلیل بهبود اسناد و نمونه های معقول تر.

• افزایش کیفیت کد به دلیل شیوه های توسعه مدرن از جمله گردش کار git-flow ،یکپارچه سازی مداوم، اجراهای شبانه در بستر آزمایشی، و یک اسکریپت برای آزمایش ساخت بیش از 1200 پروژه برای همه پلتفرم ها.

با توجه به این ویژگی‌ها، Contiki-NG هم یک ابزار بی‌سیم برای ساخت شبکه‌های بی‌سیم چند هاب، کم مصرف و محدود با قابلیت اطمینان پنج- نه و همچنین یک پلتفرم تحقیقاتی برای نوآوری درفناوری بی‌سیم کم‌مصرف است. سیستم های تعبیه شده در تمام سطوح پشت شبکه، به عنوان مثال TiSCH 6زمان بندی، مسیریابی، امنیت و لایه های MACصرفه جویی در مصرف انرژی بکار رفته اند.

علیرغم تاریخچه نسبتاً کوتاه خود،در خصوص Contiki-NG تحقیقات قدیمی تر و آنهایی که قبلا درخصوص کونتیکی منتشر و بررسی شده اند کار ما را تسهیل کردند، از جمله مقالاتی که توسط تیم هایی که به پروژه وابسته نیستند، برای مثال:

Contiki-NGبه ویژه در فعال کردن تحقیقات جدید در شبکه های IEEE 802.15.4 TSCHو IETF 6TiSCHموفق بوده است. Contiki Contiki-NG /یکی از دو پیاده‌سازی منبع باز IEEE 802.15.4 TSCHرا برای سخت‌افزار واقعی ارائه می‌کند، دیگری که توسط OpenWSNارائه شده است. نویسندگان این مقاله OpenWSNرا به عنوان یک رقیب مستقیم در نظر نمی گیرند، زیرا یک پشته شبکه است، نه یک سیستم عامل کامل.

Contiki-NGتحقیقات آینده را در بسیاری از جهات، نه محدود به امنیت اینترنت اشیا را امکان پذیر می کند. برای مثال: برداشت انرژی؛ تحرک شبکه؛ ارتباطات اینترنت اشیا چند پروتکل/چند رادیویی/باند چند فرکانسی.

فراتر از انتشارات علمی، Contiki-NGشروع به تأثیرگذاری از طریق موارد بیان شده می کند: (برای حمایت از کارهای انجام شده به عنوان بخشی از پروژه های تحقیقاتی تأمین شده در رشته های مختلف مانند: اینترنت اشیاء صنعتی، سلامت دیجیتال، شهرهای هوشمند برای تدریس در آموزش عالی میباشد.)

جدول 2 تعداد کمی از پروژه های تحقیق و توسعه تامین شده را که به طور گسترده از Contiki-NGاستفاده کرده اند فهرست

می کند. ارائه فهرست جامعی از این گونه پروژه ها بسیار طولانی خواهد بود و خارج از محدوده این مقاله تلقی می شود.

Contiki-NGبه عنوان پایه چندین محصول تجاری استفاده می شود. این فهرست شامل برنامه‌های مصرف‌کننده/منزلی، برای مثال در سیستم‌های گرمایش خانه هوشمند و لامپ‌های هوشمند است. این فهرست همچنین شامل برنامه های صنعتی مانند نظارت/ردیابی دارایی و کشاورزی هوشمند می شود.

نتیجه گیری

در این مقاله به سیستم عامل نسل جدید کونتیکی برای دستگاه های تعبیه شده بیسیم با محدودیت های شدید پرداختیم. امیدواریم با مطالعه این مقاله خواننده به درکی از ساز و کار، معماری و برخی قابلیت های غیرفنی این سیستم عامل پی برده باشد. ما سعی کردیم که در این مقاله به برتری نسل جدید کونتیکی نسبت به جد قدیمی خود بپردازیم و به ارزشی که این سیستم عامل به جامعه پژوهشی اضافه کرده اشاره کنیم.

در آخر باید گفت سیستم عامل نسل جدید کونتیکی با ورود به جوامع علمی شاخص هایی را به آن اضافه نموده است.

Funded research projects using Contiki-NG.

Project title	Funder	Ref.
F-Interop	H2020	
5G-CORAL	H2020	[34]
SPHERE	UK EPSRC	
EurValve	H2020	[32]
Vessedia	H2020	[35]
E-care@home	Swedish Knowledge Foundation	[36]
aSSIsT	Swedish Foundation for Strategic Research	
SYNERGIA	Innovate UK	

کونتیکی نسل جدید شامل نقشه راه و راهنمایی است که برای علاقمندان در گیت-هاب قابل دسترسی میباشد. آپدیت هایی که اکنون در حال پردازش هستند تحت عنوان *roadmap* علامت گذاری شده اند و تغییراتی که قرار است در آینده ای نه چندان دور روی آنها پردازش اعمال شود تحت عنوان *roadmap/long-term* علامت گذاری شده اند.

اعلامیه منافع رقابتی

نویسندگان این مطلب اعلام میکنند که هیچگونه منافع مالی یا روابط شخصی که بتواند در روند کار مقاله تاثیری داشته باشد نداشته اند.

سپاسگذاری ها

نویسندگان این مطلب افرادی هستند که در حال حاظر در تیم نگهداری و توسعه نسل جدید کونتیکی فعالیت دارند و در حالی که روی سورس کد نسل جدید کونتیکی فعالیت داشته اند و بخشی از آن را نوشته اند، روی آن ادعای مالکیت ندارند.

این پروژه توسط بنیاد تحقیقات استراتژیک سوئد حمایت شده.

سخن آخر

ترجمه و گردآوری این مقاله توسط علیرضا احتشام فر دانشجو رشته مهندسی نرم افزار از موسسه آموزش عالی صدرالمتالهین با نظارت و راهنمایی های گرانقدر استاد ارجمند جناب آقای دکتر امیر ماروی انجام پذیرفته است.

منابع و مراجع

1. Akyildiz IF, Su W, Sankarasubramaniam Y, Cayirci E. Wireless sensor networks: a survey. Comput Netw 2002;38(4):393–422.

2. Levis P, Madden S, Polastre J, Szewczyk R, Whitehouse K, Woo A, et al. TinyOS: An operating system for sensor networks. In: Ambient intelligence. Springer; 2005, p. 115–48.

3. Dunkels A, Gronvall B, Voigt T. Contiki - a lightweight and flexible operating system for tiny networked sensors. In: 29th Annual IEEE international conference on local computer networks. 2004, p. 455–62.

4. Dunkels A, Schmidt O, Voigt T, Ali M. Protothreads: Simplifying eventdriven programming of memory-constrained embedded systems. In: Proceedings of the 4th international conference on embedded networked sensor systems. New York, NY, USA: ACM; 2006, p. 29–42.

5. Dunkels A. Full TCP/IP for 8-bit architectures. In: Proceedings of the 1st international conference on mobile systems, applications and services. MobiSys '03, New York, NY, USA: ACM; 2003, p. 85–98.

6. Kim H-S, Andersen MP, Chen K, Kumar S, Zhao WJ, Ma K, et al. System architecture directions for post-soc/32-bit networked sensors. In: Proceedings of the 16th ACM conference on embedded networked sensor systems. ACM; 2018, p. 264–77

7. Baccelli E, Hahm O, Günes M, Wählisch M, Schmidt TC. RIOT OS: TOwards an OS for the internet of things. In: 2013 IEEE conference on computer communications workshops. IEEE; 2013, p. 79

8. Javed F, Afzal MK, Sharif M, Kim B-S. Internet of things (IoT) operating systems support, networking technologies, applications, and challenges: A comparative review. IEEE Commun Surv Tutor 2018;20(3):2062–100.

9. 80Silva M, Cerdeira D, Pinto S, Gomes T. Operating systems for internet of things low-end devices: Analysis and benchmarking. IEEE Internet Things J 2019;6(6):10375–83.

10. IEEE. 802.15.4-2015 - IEEE Standard for low-rate wireless networks. 2016, IEEE Std 802.15.4-2015.

11. Thubert P. An architecture for IPv6 over the time-slotted channel hopping mode of IEEE 802.15.4 (6tisch). 2021, RFC 9030.

12. Vilajosana X, Pister K, Watteyne T. Minimal IPv6 over the TSCH mode of IEEE 802.15.4e (6tisch) configuration. 2017, RFC 8180.

13. Wang Q, Vilajosana X, Watteyne T. 6TiSCH operation sublayer (6top) protocol (6P). 2018, RFC 8480.

14. Alexander R, Brandt A, Vasseur J, Hui J, Pister K, Thubert P, et al. RPL: IPV6 routing protocol for low-power and lossy networks. 2012, RFC 6550.

15. Duquennoy S, Elsts A, Al Nahas B, Oikonomou G. TSCH And 6tisch for contiki: challenges, design and evaluation. In: Proc. IEEE DCOSS. IEEE; 2017, p. 11–3.

16. Duquennoy S, Nahas BA, Landsiedel O, Watteyne T. Orchestra: Robust mesh networks through autonomously scheduled TSCH. In: Proceedings of the international conference on embedded networked sensor systems. Seoul, South Korea; 2015.

17. Chang T, Vučinić M, Vilajosana X, Duquennoy S, Dujovne D. 6TiSCH minimal scheduling function (MSF). 2021, RFC 9033.

18. Ko J, Eriksson J, Tsiftes N, Dawson-Haggerty S, Terzis A, Dunkels A, et al. ContikiRPL and TinyRPL: Happy together. In: Workshop on extending the internet to low power and lossy networks (IP+ SN). Vol. 570. Citeseer; 2011.

19. Duquennoy S, Eriksson J, Voigt T. Five-nines reliable downward routing in RPL. 2017, arXiv.

20. Hui JW, Kelsey R. Multicast protocol for low-power and lossy networks (MPL). 2016, RFC 7731.

21. Oikonomou G, Phillips I. Stateless multicast forwarding with RPL in 6lowpan sensor networks. In: Proc. IEEE international conference on pervasive computing and communications workshops. Lugano, Switzerland: IEEE; 2012, p. 272–7.

22. Oikonomou G, Phillips I, Tryfonas T. IPV6 multicast forwarding in RPLbased wireless sensor networks. Wirel Pers Commun 2013;73(3):1089– 116.

23. Abdel Fadeel KQ, El Sayed K. ESMRF: Enhanced stateless multicast RPL forwarding for IPv6-based low-power and lossy networks. In: Proc. 2015 workshop on IoT challenges in mobile and industrial systems. IoT-Sys '15, New York, NY, USA: ACM; 2015, p. 19–24.

24. Shelby Z, Hartke K, Bormann C. The constrained application protocol (CoAP). 2014, RFC 7252.

25. Bormann C, Shelby Z. Block-wise transfers in the constrained application protocol (CoAP). 2016, RFC 7959.

26. Hartke K. Observing resources in the constrained application protocol (CoAP). 2015, RFC 7641.

27. Tomasic I, Khosraviani K, Rosengren P, Jörntén-Karlsson M, Lindén M. Enabling IoT based monitoring of patients' environmental parameters: Experiences from using OpenMote with openwsn and contiki-NG. In: 2018 41st International convention on information and communication technology, electronics and microelectronics. IEEE; 2018, p. 0330–4.

28. Algora CMG, Reguera VA, Fernández EMG, Steenhaut K. Parallel rendezvous-based association for IEEE 802.15.4 tsch networks. IEEE Sens J 2018;18(21):9005–20.

29. Yang G, Urke AR, Øvsthus K. Mobility support of IoT solution in home care wireless sensor network. In: 2018 Ubiquitous positioning, indoor navigation and location-based services. IEEE; 2018, p. 475–80.

30. Cheng X, Sha M. Cracking the TSCH channel hopping in IEEE 802.15.4e. In: Proceedings of the 2018 ACM SIGSAC conference on computer and communications security. ACM; 2018, p. 2210–2

31. Lee S-B, Kim E-J, Lim Y. Contiki-NG-based IEEE 802.15.4 TSCH throughput evaluation. In: Proceedings of the Korean institute of information and commucation sciences conference. The Korea Institute of Information and Commucation Engineering; 2018, p. 577–8.

32. Elsts A, Pope J, Fafoutis X, Piechocki R, Oikonomou G. Instant: A TSCH schedule for data collection from mobile nodes. In: Proc. 2019 international conference on embedded wireless systems and networks. EWSN, 2019.

33. Watteyne T, Vilajosana X, Kerkez B, Chraim F, Weekly K, Wang Q, et al. OpenWSN: a standards-based low-power wireless development environment. Trans Emerg Telecommun Technol 2012;23(5):480–93.

34. Li C-Y, Chien H-T, editors. Communication, dissemination, standardisation and exploitation achievements of Y1 and plans for Y2. 2018, 5G-CORAL Deliverable D5.1.

35. Peyrard A, Kosmatov N, Duquennoy S, Raza S. Towards formal verification of contiki: Analysis of the AES–ccm* modules with frama-c. In: Proc. workshop on recent advances in secure management of data and resources in the IoT. RED-IOT, Madrid, Spain; 2018.

36. Alirezaie M, Renoux J, Köckemann U, Kristoffersson A, Karlsson L, Blomqvist E, et al. An ontology-based context-aware system for smart homes: E-care@home. Sensors 2017;17(7):1586.