# ساده‌سازی برنامه‌نویسی در سیستم عامل TinyOS مورد استفاده در شبکه حس‌گر بی‌سیم

سیدمیثم خضری، مهدی آقا صرام و فضل‌الله ادیب‌نیا

**چکیده: سیستم عامل TinyOS [۱] به‌عنوان پرکاربردترین سیستم عامل گره حس‌گر بی‌سیم، دارای مدل برنامه‌نویسی مبتنی بر رویداد است. برنامه‌نویسی مبتنی بر رویداد مستلزم استفاده از ماشین حالات است که برنامه‌نویس را ملزم به مدیریت دستی پشته برنامه می‌کند. به‌همین دلیل پیاده‌سازی پردازش‌های طولانی در سیستم‌های مبتنی بر رویداد مانند TinyOS دشوار می‌باشد. در این مقاله سعی کرده‌ایم با ایجاد تغییراتی در زمان‌بند TinyOS، انتزاع برنامه‌نویسی جدیدی برای این سیستم عامل ارائه کنیم که پیاده‌سازی پردازش‌های طولانی را در آن ساده‌تر می‌کند و به توسعه‌دهنده برنامه کاربردی امکان کدنویسی رویه‌ای و چندریسه‌ای را می‌دهد. تغییرات در زمان‌بند TinyOS به‌نحوی انجام شده که با برنامه‌های کاربردی قبلی نیز سازگاری داشته باشد. نتایج ارزیابی در یک کاربرد نمونه نشان می‌دهد که از نظر توان مصرفی مدل پیشنهادی تفاوت چندانی با مدل قبلی ندارد، هرچند سربار حافظه مصرفی و سربار پردازشی آن نسبت به مدل قبلی بیشتر است.**

***کلید واژه:*** **برنامه‌نویسی چندریسه‌ای، برنامه‌نویسی مبتنی بر رویداد، سیستم عامل، گره حس‌گر بی‌سیم، TinyOS.**

## ۱- مقدمه

شبکه حس‌گر بی‌سیم، متشکل از تعداد زیادی گره حس‌گر بی‌سیم است که در یک محیط پراکنده شده‌اند. در این شبکه هر گره، خودمختار بوده و توانایی انجام عملیات پردازشی، جمع‌آوری اطلاعات حس‌گرها و برقراری ارتباط رادیویی با سایر گره‌ها را دارد. تغییر شرایط محیطی مانند دما، رطوبت، صدا، فشار و ... توسط حس‌گر تشخیص داده می‌شود و داده‌های به‌دست آمده پس از جمع‌آوری توسط فرستنده رادیویی گره حس‌گر در فضای اطراف منتشر شده و به کمک سایر گره‌های حس‌گر به سمت ایستگاه مرکزی هدایت می‌شوند. امروزه شبکه‌های حس‌گر بی‌سیم در زمینه‌های نظامی، پزشکی، محیط زیست، کشاورزی، کنترل صنعتی، کنترل ترافیک و ... کاربرد دارند.

سکوی سخت‌افزاری گره حس‌گر عموماً از یک میکروکنترلر، حافظه جانبی Flash، فرستنده گیرنده بی‌سیم، انواع حس‌گرها و منبع تغذیه به‌صورت باتری تشکیل می‌شود. در شبکه‌های حس‌گر اولیه به دلیل توانایی‌های بسیار محدود سخت‌افزاری و سادگی کاربردها، نرم‌افزار لازم مخصوص هر کاربرد نوشته می‌شد ولی با پیچیده‌ترشدن برنامه‌های کاربردی نیاز به نرم‌افزاری سیستمی احساس شد تا مدیریت منابع سیستم را راحت‌تر کرده و روند اجرای برنامه را قابل پیش‌بینی کند. نرم‌افزار سیستمی که سرویس‌های پایه را برای توسعه‌دهنده برنامه کاربردی فراهم می‌کند، سیستم عامل گره حس‌گر نامیده می‌شود. معماری سیستم عامل گره حس‌گر بر میزان استفاده از حافظه، الگوی مصرف توان و نحوه استفاده از پردازنده برای اجرای وظایف موجود تأثیرگذار است. از آنجا که سیستم عامل گره حس‌گر موظف به ارائه سرویس به برنامه‌های کاربردی است، تنها نمی‌تواند روی صرفه‌جویی در توان متمرکز شود. مقادیر خوانده‌شده توسط حس‌گرها و پیام‌های دریافتی از سایر گره‌ها باید به شکل مناسبی اداره شوند.

بسیاری از سیستم عامل‌های خاص گره حس‌گر از مدل اجرای مبتنی بر رویداد برای پشتیبانی چندوظیفگی استفاده می‌کنند. در این مدل از اجرای برنامه تنها یک پشته برای ذخیره‌سازی حالت اجرا وجود دارد. به‌دلیل محدودیت‌های پردازشی و حافظه در سخت‌افزار گره حس‌گر در اغلب موارد این مدل نسبت به گزینه‌های دیگر ارجحیت دارد. اما مدل مبتنی بر رویداد چندان مناسب پیاده‌سازی سرویس‌های پیشرفته‌تر نیست. چنانچه برنامه‌نویس بخواهد محاسبات طولانی را به این روش پیاده‌سازی کند مجبور است آن را به پردازش‌های کوتاه‌تر تقسیم کند تا پاسخ‌گویی به سایر رویدادهای هم‌زمان با مشکل مواجه نشود. فشرده‌سازی داده‌ها نمونه‌ای از این دست محاسبات طولانی است که می‌تواند در کاربردهایی نظیر لرزه‌نگاری [۲] یا انتقال تصاویر به گره مرکزی [۳] مفید واقع شود. در [۴] نشان داده شده است که فشرده‌سازی داده‌ها می‌تواند به میزان قابل توجهی انرژی مصرفی گره‌های حس‌گر را کاهش دهد و بر طول عمر شبکه حس‌گر بی‌سیم بیافزاید.

در این مقاله سعی کرده‌ایم که با ایجاد تغییراتی در سیستم‌عامل TinyOS پیاده‌سازی پردازش‌های طولانی در این سیستم عامل را برای برنامه‌نویسان ساده‌تر کنیم. ما با ارائه مفهوم پردازشی جدیدی به نام Job، زمان‌بند این سیستم‌عامل را طوری تغییر داده‌ایم که از برنامه‌نویسی رویه‌ای پشتیبانی شود. در ادامه مقاله ابتدا انواع معماری‌های موجود برای سیستم عامل‌های گره حس‌گر و نحوه پاسخ‌گویی به رویدادها در گره حس‌گر معرفی شده‌اند. در بخش دوم مدل هم‌زمانی TinyOS تشریح شده است. بخش سوم به کارهای انجام‌شده در این زمینه اختصاص دارد. مدل پیشنهادی در بخش ۴ توصیف شده و در بخش ۵ ارزیابی شده است. بخش نهایی مقاله حاوی نتیجه‌گیری و دربرگیرنده پیشنهاداتی برای ادامه تحقیقات در این زمینه است.

### ۱-۱ معماری سیستم عامل گره حس‌گر

در حال حاضر اکثر سیستم عامل‌های موجود برای گره حس‌گر از مدل مبتنی بر رویداد برای اداره محاسبات خود استفاده می‌کنند [۱] و [۵]. در این دسته از سیستم عامل‌ها، هر عملی که سیستم عامل انجام می‌دهد در

این مقاله در تاریخ ۹ مهر ماه ۱۳۸۷ دریافت و در تاریخ ۲۴ اردیبهشت ماه ۱۳۸۹ بازنگری شد.

سیدمیثم خضری، دانشکده مهندسی برق و کامپیوتر، دانشگاه یزد، صفاییه – بلوار دانشگاه– خیابان پژوهش یزد، (email: khezri@stu.yazduni.ac.ir).

مهدی آقا صرام، دانشکده مهندسی برق و کامپیوتر، دانشگاه یزد، صفاییه – بلوار دانشگاه– خیابان پژوهش یزد، (email: mehdi.sarram@yazduni.ac.ir).

فضل‌الله ادیب‌نیا، دانشکده مهندسی برق و کامپیوتر، دانشگاه یزد، صفاییه – بلوار دانشگاه– خیابان پژوهش یزد، (email: fadib@yazduni.ac.ir).

پاسخ به یک رویداد ناشی از وقفه‌های سخت‌افزار (مانند وقفه تایمر، رسیدن یک بسته جدید از کانال رادیویی و ...) است. پس از آن که سیستم عامل وظایف مورد نظر در قبال رویدادها را انجام داد، گره برای صرفه‌جویی در مصرف توان به حالت استراحت می‌رود تا زمانی که وقفه سخت‌افزاری بعدی رخ دهد. سیستم عامل TinyOS یکی از این سیستم عامل‌هاست که در آن وظایف منتسب به رویدادها به‌ترتیب زمانی (FCFS) و پشت سر هم تا زمانی اجرا می‌شوند که خاتمه بیابند. این مدل اجرای وظایف مزایای متعددی برای گره حس‌گر با منابع محدود دارد. اولاً به‌دلیل آن که در هر لحظه از زمان تنها یک وظیفه تا پایان اجرا می‌شود، نیازی به عمل پرهزینه تعویض متن نیست. ثانیاً از آنجایی که وظایف تا زمان خاتمه سرویس خود اجرا می‌شوند لازم نیست برای هر وظیفه پشته مخصوص در نظر گرفته شود. پس از پایان وظیفه دیگر نیازی به پشته آن نیست، در نتیجه تمام وظایف می‌توانند از یک پشته مشترک استفاده کنند. این امر موجب صرفه‌جویی قابل توجهی در میزان مصرف حافظه خواهد شد. از طرفی اجرای انحصاری وظایف نسبت به هم، توسعه‌دهنده برنامه کاربردی را از مواجهه با مشکلاتی چون به‌وجود آمدن شرایط رقابتی در دسترسی به متغیرهای مشترک دور نگه می‌دارد.

در مقابل سیستم عامل‌هایی وجود دارند که از مدل هم‌زمانی چندریسه‌ای استفاده می‌کنند. سیستم عامل به‌طور هم‌زمان چندین ریسه را اجرا می‌کند. هر ریسه پشته مخصوص به خود را دارد و زمان‌بند سیستم به محض پایان مهلت زمانی اختصاص داده شده به یک ریسه عمل تعویض متن را انجام می‌دهد و پشته ریسه دیگری را جایگزین پشته ریسه فعلی می‌کند. هرچند سیستم عامل‌های مبتنی بر معماری چندریسه‌ای مانند uC/OS [۶] و FreeRTOS [۷] در بسیاری از سیستم‌های تعبیه‌شده[1] با موفقیت پیاده‌سازی شده‌اند اما به‌دلیل سربار حافظه مربوط به ذخیره پشته هر ریسه و سربار پردازشی ناشی از عمل تعویض متن استفاده از آنها در گره حس‌گر بی‌سیم با منابع محدود چندان موفقیت‌آمیز نبوده است. سیستم‌عامل MANTIS [۸] و RETOS [۹] بر اساس این مدل برای گره حس‌گر بی‌سیم طراحی شده‌اند و هدف آنها ساده‌سازی توسعه برنامه‌های کاربردی مخصوص شبکه حس‌گر بی‌سیم بوده است.

### ۲-۱ پیاده‌سازی برنامه کاربردی برای گره حس‌گر

گره حس‌گر بی‌سیم می‌بایست در برابر محرک‌های مختلف شامل رویدادهای فیزیکی و پیام‌های دریافتی از سایر گره‌ها واکنش نشان دهد. از آنجا که محیط شبکه‌های حس‌گر غیر ساخت‌یافته و دائماً در حال تغییر می‌باشد، مدل رایج محاسباتی معمولاً مبتنی بر رویداد و ناهم‌زمان است. در ساده‌ترین حالت، مدل مبتنی بر رویداد متشکل از تعدادی فعالیت یا سرویس است که بسته به رویداد اتفاق‌افتاده اجرا می‌شوند. اتفاق یک رویداد جدید می‌تواند باعث رفتن به حالت دیگری از برنامه شود و به حالت فعلی آن بستگی دارد. اگرچه این مدل از نظر مفهومی ساده است اما با روش معمول برنامه‌نویسی مغایرت دارد. توسعه‌دهنده مجبور است برنامه خود را در قالب یک ماشین حالت پیاده‌سازی کند و حالت برنامه را بین اجرای چند وظیفه مختلف حفظ کند.

روش معمول برای اجرای هم‌زمان چند کد مختلف استفاده از چندبرنامگی است که به دو صورت Preemptive و Cooperative انجام می‌شود. برخلاف چندریسگی Preemptive که در آن سیستم عامل پردازنده را از ریسه در حال اجرا می‌گیرد، در چندریسگی Cooperative ریسه در حال اجرا به‌صورت داوطلبانه پردازنده را در اختیار سیستم عامل قرار می‌دهد. استفاده از چندریسگی Cooperative یک انحصار ضمنی برای برنامه اجرایی ایجاد می‌کند و مشکل بروز شرایط رقابتی را که در چندریسگی Preemptive اتفاق می‌افتد از بین می‌برد.

در این مقاله با استفاده از چندریسگی Cooperative یک انتزاع برنامه‌نویسی جدید در سیستم‌عامل مبتنی بر رویداد TinyOS معرفی کرده‌ایم که به برنامه‌نویس اجازه می‌دهد کد خود را به‌صورت کلاسیک و رویه‌ای بنویسد. توسعه‌دهنده برنامه می‌تواند به کمک یک سری توابع، عملیات پایه محاسباتی مانند خواندن مقدار حس‌گر، دریافت و ارسال بسته و ... را به‌صورت بلوکه‌شونده انجام دهد.

## ۲- سیستم عامل TINYOS

برنامه‌های کاربردی TinyOS و همچنین بخش غالب این سیستم عامل به زبان nesC [۱۰] که برگرفته از زبان برنامه‌نویسی C می‌باشد، نوشته می‌شوند.

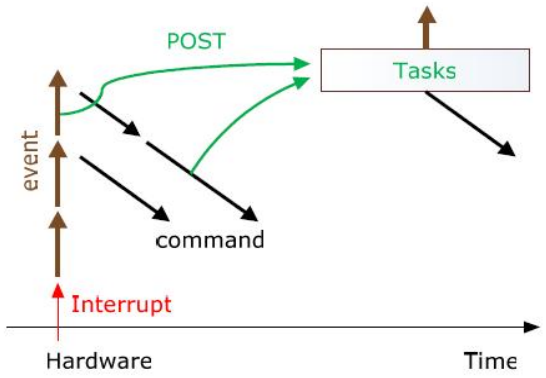
مدل اجرایی زبان nesC از دو نوع پردازش پشتیبانی می‌کند: یکی در قالب اجرای وظایف (Tasks) و دیگری اجرای اداره‌کننده‌های وقفه که به‌صورت ناهم‌زمان توسط سخت‌افزار فعال می‌شوند. Task یک نوع فراخوانی با تعویق تابع است به این معنی که یک module می‌تواند Task را به زمان‌بند TinyOS پست کند تا مدتی بعد زمان‌بند بر اساس سیاست زمان‌بندی تعیین‌شده Task را به اجرا درآورد. اجرای Taskها با سیاست اجرا تا اتمام انجام می‌شود یعنی تا تکمیل عملیات یک Task، زمان‌بند Task دیگری را اجرا نمی‌کند. اجرای انحصاری Taskها نسبت به همدیگر باعث ساده‌سازی کد می‌شود چرا که خطری از جانب اجرای ناگهانی یک کد دیگر و دست‌کاری داده‌ها، برنامه را تهدید نمی‌کند. البته اداره‌کننده‌های وقفه می‌توانند اجرای Task را با وقفه روبه‌رو کنند.

پردازنده تا زمانی که یک وقفه سخت‌افزاری روی دهد در حالت استراحت باقی می‌ماند. با وقوع وقفه سخت‌افزاری، ریزکنترل‌کننده از حالت استراحت خارج شده و اداره‌کننده وقفه را اجرا می‌کند. اداره‌کننده وقفه یک اداره‌کننده رویداد را فراخوانی می‌کند (فلش‌های به سمت بالا). کد اداره‌کننده رویداد می‌تواند فرامینی را صادر کند (فلش‌های مورب) یا یک Task به زمان‌بند ارسال[2] کند. همچنین کد یک فرمان نیز می‌تواند این عمل را انجام دهد. چنانچه اداره‌کننده وقفه یک یا چند Task به زمان‌بند پست کرده باشد، زمان‌بند مجدداً اجرای Taskها را تا خالی‌شدن صف مربوطه انجام می‌دهد. پس از اجرای تمام Taskها پردازنده مجدداً به حالت استراحت خواهد رفت. همان‌طور که در شکل ۱ دیده می‌شود اجرای کد درون Task به‌صورت انحصاری در یک زمان آغاز شده و مدتی بعد به پایان می‌رسد. در زمان اجرای کد Task هیچ Task دیگری اجرا نخواهد شد و تنها این اداره‌کننده‌های رویداد هستند که می‌توانند در خلال اجرای Task اجرا شوند. این مسئله می‌تواند به بروز شرایط رقابتی بین کد درون Task و کد اداره‌کننده رویداد شود. زبان nesC برای پیشگیری از این مشکل تدابیری اندیشیده است.

از آنجا که Taskها به‌صورت انحصاری پردازنده را در اختیار می‌گیرند، چنانچه Task فعلی که پردازنده را در اختیار دارد به محاسبات طولانی نیاز داشته باشد، اجرای سایر Taskها که ممکن است اولویت بیشتری نسبت به Task فعلی داشته باشند برای مدت طولانی به تعویق می‌افتد.

1. Embedded Systems

2. Post




شکل ۱: رویدادها و فرامین می‌توانند Task را به زمان‌بند ارسال کنند.

## ۳- کارهای انجام‌شده

برای رفع مشکلات مربوط به مدل هم‌زمانی و همچنین ساده‌سازی برنامه‌نویسی در سیستم عامل TinyOS تا به حال روش‌های مختلفی پیشنهاد شده است که در این بخش به‌طور مختصر آنها را شرح می‌دهیم:

TinyMos [۱۱] چارچوبی را برای توسعه‌دهنده فراهم می‌کند که به کمک آن می‌توان چند برنامه nesC را به‌طور موازی با هم اجرا کرد. طراحان TinyMos توانسته‌اند TinyOS را به‌صورت یک نخ روی زمان‌بند سیستم عامل MANTIS اجرا کنند. سربار ناشی از TinyMos مربوط به کد زمان‌بند سیستم اصلی، حافظه مورد نیاز برای پشته هر نخ و سربار ناشی از عمل تعویض متن است.

کتابخانه TinyThread [۱۲] یک سری توابع API را در اختیار برنامه‌نویس قرار می‌دهد که از عملیات Blocking I/O پشتیبانی می‌کند. زمان‌بند TinyThread در قالب یک Task در سیستم عامل TinyOS اجرا می‌شود و عمل زمان‌بندی را به‌صورت Cooperative انجام می‌دهد. کتابخانه TinyThread شامل یک سری توابع API جهت ایجاد نخ، عملیات ورودی خروجی Blocking از قبیل ارسال و دریافت بسته‌ها و توابع سنکرون‌سازی نخ‌ها با همدیگر است.

زمان‌بند اولویت‌دار [۱۳] توسط گروهی از محققین دانشگاه Cork ایرلند برای TinyOS طراحی شده است. زمان‌بند اولویت‌دار فوق ۵ سطح اولویت برای Taskها تعریف می‌کند که در هر سطح عمل زمان‌بندی به‌صورت FIFO انجام می‌شود. Taskهایی که در سطح اولویت بالاتر قرار دارند می‌توانند جایگزین هر Task در حال اجرا از سطوح اولویت پایین‌تر از خود شوند. یکی از مشکلاتی که در زمان‌بند اولویت‌دار طراحی‌شده دیده می‌شود، عدم تبعیت آن از مدل هم‌زمانی nesC است. ویژگی Preemptiveبودن این زمان‌بند می‌تواند باعث بروز مشکل شرایط رقابتی بین Taskها در سطوح مختلف اولویت شود.

سیستم عامل Contiki [۵] یک سیستم عامل مبتنی بر رویداد برای گره حس‌گر بی‌سیم می‌باشد که ترکیبی از یک هسته سبک‌وزن مبتنی بر رویداد به همراه یک کتابخانه توابع است که از Preemptive Multithreading پشتیبانی می‌کند. برنامه‌هایی که نیازمند اجرا به‌صورت چندریسه‌ای هستند می‌توانند از توابع موجود در این کتابخانه استفاده کنند. طراحان سیستم‌عامل Contiki مفهوم برنامه‌نویسی جدیدی به نام Protothread [۱۴] معرفی کرده‌اند که نوشتن برنامه‌های مبتنی بر رویداد را ساده‌تر می‌کند؛ بدون آن که سربار زیادی از نظر حافظه به سیستم تحمیل کند. هرچند Protothread تعداد حالت‌های موجود در ماشین حالات برنامه را کاهش می‌دهد اما برنامه‌نویس همچنان باید از این مدل جهت برنامه‌نویسی استفاده کند.

```
01 module SenseAndForward {
02   uses {
03     interface AMSend;
04     interface Timer;
05     interface Read
06     ... }  }
07 implementation {
08   message_t packet;
09   bool RadioBusy;
10   uint16_t data;
11
12   event void Boot.booted() {
13      call Timer.startPeriodc(1024);
14   }
15   event void Timer.fired() {
16      call Read.read();
17   }
18   event void Read.readDone (...) {
19      if (!RadioBusy) {
20          RadioBusy = TRUE;
21          data = sensorData;
22          post sendData();
23      }
24   }
25   task void SendData() {
26      pReading = call Packet.getpayload(&packet,NULL);
27      pReading->data = data;
28      if (call AMSend.send(...) != SUCCESS { RadioBusy = FALSE; }
29   }
30   event void AMSend.sendDone (...) {
31      if (&packet == bufptr) { RadioBusy = FALSE; }
32   }
33 }
```

شکل ۲: پیاده‌سازی برنامه SenseAndForward به روش مبتنی بر رویداد.

سیستم عامل RETOS [۹] از پایه برای پشتیبانی از توابع API نخ‌ها در گره حس‌گر طراحی شده است. هرچند پیاده‌سازی RETOS از استاندارد POSIX در این زمینه، با آن سازگاری کاملی ندارد اما به اندازه کافی برنامه‌نویسی را برای کاربردهای شبکه حس‌گر ساده می‌کند.

## ٤- سبک برنامه‌نویسی پیشنهادی

برای توسعه یک برنامه کاربردی ساده در nesC برنامه‌نویس ناچار است دسترسی به حافظه مشترک بین Task و رویدادهای ناهم‌زمان را کنترل کرده و درخواست‌های پایه خود را به چندین عملیات دومرحله‌ای تقسیم کند.

به‌عنوان نمونه شکل ۲ کد خلاصه‌شده یک برنامه کاربردی ساده به‌نام SenseAndForward در TinyOS را نشان می‌دهد که وظیفه آن خواندن مقدار حس‌گرها و انتشار داده به‌دست آمده در فضای اطراف است. هر بار که رویداد Timer.fired گزارش داده می‌شود، فرمان خواندن داده از حس‌گر صادر می‌شود. پس از آماده‌شدن مقدار خوانده‌شده، رویداد Read.readDone گزارش داده می‌شود و اداره‌کننده این رویداد Task مربوط به ارسال داده را به زمان‌بند پست می‌کند. متغیر منطقی RadioBusy تنها در صورتی اجازه ارسال می‌دهد که رادیو ارسال داده قبلی را به پایان رسانده باشد. در این کد وظیفه اصلی خواندن و ارسال داده به چهار قسمت مجزا شکسته شده است.

هدف ما این است که یک انتزاع ساده و سرراست برای برنامه‌نویس TinyOS فراهم کنیم. برای مثال روش ساده‌تر برای خواندن مقدار حس‌گر این است که به شکل یک فرمان بلوکه‌شونده، تقاضای خواندن مقدار حس‌گر صادر شده و داده خوانده‌شده در قالب مقدار خروجی برگردانده شود. در روش پیشنهادی ما عملیات مجزای درخواست و پاسخ به شکل یک درخواست واحد در می‌آید و به برنامه‌نویس این اجازه را می‌دهد که کد خود را به‌صورت رویه‌ای پیاده‌سازی کند. شکل ۳ نشان می‌دهد چگونه برنامه کاربردی فوق با روش پیشنهادی ما پیاده‌سازی شده است.

```
01 module SenseAndForward {
02   uses {
03     interface Socket;
04      interface BlockingRead;
05      interface msleep;
06      interface Job;
07      ... }}
08 implementation {
09   message_t packet;
10   uint16_t data = 0;
11   uint8_t stack[STACK_SIZE];
12
13   event void Boot.booted() {
14     call Socket.init();
15     call Job.postJob(STACK_TOP(stack));
16   }
17
18   event void Job.runJob() {
19     while (1) {
20         call BlockingRead.read(&data);
21         pReading = call Packet.getPayload(&packet, NULL);
22         pReading->data = data;
23         call Socket.send (...);
24         call msleep.sleep(1024);
25     }
26   }
27 }
```

شکل ۳: پیاده‌سازی برنامه SenseAndForward به روش پیشنهادی.

```
1: interface Job {
2:     async command error_t postJob();
3:     event void runJob();
4:     command void yield();
5: }
```

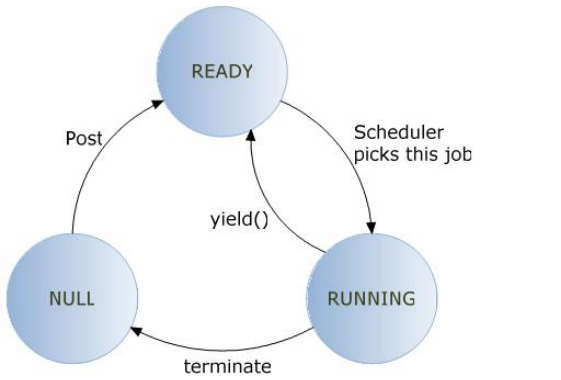
شکل ۴: رابط کاربری Job.

برای رسیدن به این هدف یک مفهوم محاسباتی جدید به نام Job به مدل هم‌زمانی TinyOS اضافه کرده‌ایم که روال اجرای برنامه در آن، پس از صدور تقاضا متوقف شده و با حاضرشدن پاسخ مربوطه ادامه می‌یابد. رابط کاربری Job (شکل ۴) امکان چندریسگی Cooperative را فراهم می‌کند. فرمان ()yield این امکان را به برنامه‌نویس می‌دهد که پردازنده را داوطلبانه در اختیار سیستم عامل قرار دهد.

مدل اجرایی Job دارای دیاگرام حالاتی است که در آن Jobها می‌توانند در یکی از ۳ وضعیت NULL، READY و RUNNING باشند (شکل ۵). با صدور فرمان ()postJob، زمان‌بند شناسه Job مربوطه را در صف مخصوص آن قرار می‌دهد و حالت Job از وضعیت اولیه NULL به READY تغییر می‌کند. هر Job یک پشته مخصوص خود دارد که حالت محاسبات را در آن نگهداری می‌کند. در این تغییر حالت پشته Job با مقادیر اولیه مقداردهی می‌شود. پس از آن که زمان‌بند TinyOS تمام Taskهای موجود را اجرا کرد به سراغ صف نگهداری Jobها رفته و از ابتدای آن یک Job را برای اجرا انتخاب می‌کند. در اینجا حالت Job از READY به RUNNING تغییر کرده و زمان‌بند رویداد ()runJob را گزارش می‌دهد، پشته Job جایگزین پشته سیستم عامل شده و عمل تعویض متن انجام می‌شود.

برنامه‌نویس به‌عنوان استفاده‌کننده Job موظف به پیاده‌سازی رویداد ()runJob است. چنانچه در حین اجرای کد فرمان ()yield صادر شود، عمل تعویض متن انجام شده و پشته سیستم عامل جایگزین پشته Job خواهد شد و حالت آن از RUNNING به READY تغییر می‌کند. روند فوق تا زمانی تکرار می‌شود که عملیات Job به پایان برسد.

با استفاده از زمان‌بند جدید و مجموعه‌ای از توابع کمکی، می‌توانیم رابط‌های کاربری TinyOS را بازنویسی و عملیات دوقسمتی تقاضا و پاسخ را در قالب یک دستور بلوکه‌شونده خلاصه کنیم. شکل ۶ پیاده‌سازی




شکل ۵: حالات Job و گذر بین آنها.

```
01 module BlockingReadP {
02  provides interface BlockingRead;
03  uses interface Read;
04 }
05 implementation {
06   uint16_t val;
07   uint8_t job_id;
08   error_t res;
09
10   command error_t BlockingRead.read(uint16_t* data)
11   {
12    job_id = id;
13    call Read.read();
14    SUSPEND(job_id); // yield processor to scheduler
15    *data = val;     // next time job starts from here
16    return res;
17   }
18
19   event void Read.readDone(error_t result, uint16_t data)
20   {
21      if (result==SUCCESS) {
22          val = data;
23      }
24      res = result;
25      RESUME(job_id);  // put the job to READY list
26   }
27 }
```

شکل ۶: پیاده‌سازی رابط کاربری BlockingRead.

رابط کاربری میانی BlockingRead برای خواندن حس‌گرها را نشان می‌دهد که جایگزین رابط کاربری Read شده است. فرمان بلوکه‌شونده ()BlockingRead.read با ترکیب فرمان ()Read.read، رویداد ()Read.readDone و استفاده از قابلیت چندریسگی Cooperative که Job فراهم کرده است، عملیات بلوکه‌شونده خواندن حس‌گر را در اختیار کاربر می‌گذارد. پس از صدور فرمان ()Read.read کنترل پردازنده به سیستم عامل داده می‌شود و مدتی بعد با آماده‌شدن مقدار خوانده‌شده، اداره‌کننده رویداد ()Read.readDone مقدار خوانده‌شده را در یک متغیر محلی قرار داده و سپس Job مربوطه را مجدداً در صف READY قرار می‌دهد. با شروع مجدد اجرای Job مقدار متغیر محلی حاوی داده خوانده‌شده به‌عنوان پارامتر خروجی برگردانده می‌شود.

## ۵- ارزیابی

برای ارزیابی مدل پیشنهادی برنامه کاربردی SenseAndForward را به هر دو روش پیاده‌سازی کردیم. دو برنامه فوق برای سکوی سخت‌افزاری Mica۲ کامپایل شده و با استفاده از شبیه‌ساز Avrora و ابزارهای جانبی دیگر میزان حافظه مصرفی، انرژی مصرفی و درصد فعالیت پردازنده در اجرای هر برنامه با هم مقایسه شدند. نرم‌افزار Avrora [۱۵] شبیه‌سازی با معماری باز است که ریزکنترل‌کننده AVR به‌کار رفته در خانواده Mica را شبیه‌سازی می‌کند. این شبیه‌ساز با دقتی در حد

جدول ۱: سربار حافظه مصرفی در برنامه SENSEANDFORWARD بر حسب بایت.

| | Blocking | Non-Blocking | Difference |
|---|---|---|---|
| ROM | ۱۲۱۴۶ | ۱۱۰۶۸ | ۱۰۷۸ (۸٪) |
| RAM | ۳۸۰ | ۲۲۵ | ۱۵۵ (۴۰٪) |

جدول ۳: مقایسه درصد فعالیت پردازنده در برنامه SENSEANDFORWARD.

| | Blocking | Non Blocking | Difference |
|---|---|---|---|
| Active | ۸٫۹۱۷۷٪ | ۸٫۴۳۵۸٪ | ۰٫۴۸۱۹٪ |

سیکل ساعت پردازنده قادر است کد برنامه کاربردی قابل اجرا روی سخت‌افزار واقعی گره حس‌گر را اجرا کند. همچنین ابزار دقیقی برای اندازه‌گیری کمیت‌های مختلف در اختیار توسعه‌دهنده قرار می‌دهد تا بتواند رفتار برنامه خود را پیش از نصب در محیط واقعی مورد بررسی قرار دهد. شبیه‌ساز Avrora به زبان شیءگرای Java نوشته شده است و این مسئله به توسعه‌دهنده کمک می‌کند به‌راحتی ابزار اندازه‌گیری جدیدی به آن اضافه کند.

جدول ۱ سربار حافظه مصرفی مدل پیشنهادی فوق را نشان می‌دهد. اضافه‌شدن کد برنامه که در حافظه ROM قرار می‌گیرد، به‌دلیل استفاده از زمان‌بند جدید و رابط کاربری‌های میانی وابسته به آن است. حافظه داده به این دلیل بیشتر مصرف شده است که مقداری از فضای RAM به پشته Job اختصاص داده شده است. در اینجا برای آن که روند اجرای برنامه با مشکل مواجه نشود، حافظه مربوط به نگهداری پشته را به اندازه کافی بزرگ در نظر گرفته‌ایم. میزان حافظه تخصیص داده شده به پشته Job برابر ۱۲۸ بایت است.

برای اندازه‌گیری میزان سربار انرژی مصرفی دو گره حس‌گر را در کنار هم قرار دادیم. یک گره حس‌گر برنامه کاربردی ذکرشده را اجرا می‌کند یعنی در هر ثانیه عمل نمونه‌برداری از حس‌گر خود را انجام داده و داده خوانده‌شده را در فضای اطراف منتشر می‌کند. گره دیگر به‌عنوان ایستگاه مرکزی عمل کرده و این داده‌ها را دریافت می‌کند. نتایج ارزیابی به مدت ۱۰ دقیقه جمع‌آوری شده‌اند. همان‌طور که در جدول ۲ دیده می‌شود تفاوت چندانی از نظر مصرف انرژی بین این دو پیاده‌سازی وجود ندارد.

برای مقایسه سربار پردازشی مربوط به مدل پیشنهادی، سناریوی قبل را تکرار کردیم و به کمک ابزار مونیتورینگ شبیه‌ساز Avrora درصد زمان فعالیت پردازنده را در هر دو حالت اندازه گرفتیم. نتایج مقایسه در جدول ۳ نشان داده شده است. به‌دلیل پردازش‌های اضافی مربوط به زمان‌بند جدید و همچنین عملیات تعویض متن، سربار پردازشی مدل پیشنهادی ما از مدل مبتنی بر رویداد بیشتر است.

## ۶- نتیجه‌گیری و کارهای آینده

در این مقاله نشان دادیم چگونه می‌توان با تغییر زمان‌بند TinyOS و استفاده از مفهوم پردازشی Job عملیات چندقسمتی معمول این سیستم عامل را به‌صورت یک فرمان بلوکه‌شونده در اختیار برنامه‌نویس قرار داد. هرچند استفاده از مدل برنامه‌نویسی مبتنی بر رویداد به دلیل سربار پردازشی کمتر در حافظه و پردازش توصیه می‌شود اما برای پیاده‌سازی پردازش‌های پیچیده‌تر مانند فشرده‌سازی یا رمزنگاری، استفاده از مدل پیشنهادی ما برنامه‌نویسی را ساده‌تر می‌کند زیرا زمان‌بند به‌طور خودکار مدیریت پشته را بر عهده می‌گیرد و لازم نیست برنامه‌نویس حالت جاری محاسبات را در متغیرهای محلی ذخیره کند. ساده‌ترشدن برنامه‌نویسی خوانایی برنامه را افزایش می‌دهد و به تولید برنامه‌های کاربردی پیشرفته‌تر

جدول ۲: مقایسه انرژی مصرفی در برنامه SENSEANDFORWARD (بر حسب ژول).

| | Operation Mode | Blocking | Non-Blocking | Difference |
|---|---|---|---|---|
| CPU | Active | ۱٫۲۱۴ | ۱٫۱۴۸ | ۰٫۰۶۶ (۵٪) |
| | Idle | ۵٫۴۸۱ | ۵٫۵۱ | |
| | Power Down | ۰ | ۰ | |
| | Power Save | ۰ | ۰ | |
| | Total | ۶٫۶۹۵ | ۶٫۶۵۹ | ۰٫۰۳۶ (۰٫۵٪) |
| Radio | Power Off | ۰ | ۰ | |
| | Receive (Rx) | ۱۷٫۰۷۴ | ۱۷٫۰۶۹۸ | ۰٫۰۰۴۲ (~۰٪) |
| | Transmit (Tx) ۰ | ۰٫۰۰۴ | ۰٫۰۰۴ | ۰ |
| | Transmit (Tx) ۱۵ | ۰٫۱۹۸ | ۰٫۲۰۳ | |
| | Total | ۱۷٫۲۷۶ | ۱۷٫۲۷۷ | |
| Total | | ۲۳٫۹۷۱ | ۲۳٫۹۳۶ | ۰٫۰۳۵ (۰٫۱٪) |

کمک می‌کند. مزیت دیگر مدل پیشنهادی ما سازگاری آن با مدل فعلی مبتنی بر رویداد در TinyOS است. از آنجا که سیستم عامل TinyOS از سوی محققین به‌عنوان استاندارد در زمینه تحقیقات کاربردی روی شبکه حس‌گر شناخته می‌شود قابل اجرا روی سکوهای مختلف سخت‌افزاری بوده و زیرسیستم‌هایی نظیر مسیریابی، پروتکل‌های انتشار و جمع‌آوری اطلاعات برای آن توسعه داده شده‌اند. زمان‌بند پیشنهادی همچنان از Taskها و رویدادها پشتیبانی می‌کند و کد مبتنی بر رویداد TinyOS نسبت به کد پردازش طولانی Job دارای اولویت اجرایی است. در نتیجه احتیاجی به تغییر برنامه‌های کاربردی فعلی نیست.

تحقیقات انجام‌شده در این مقاله را می‌توان از چندین جنبه ادامه داد. اول آن که لازم است مدل پیشنهادی فوق در کاربردهای واقعی شبکه حس‌گر بی‌سیم که نیاز به انجام محاسبات پیچیده دارند (مانند رمزنگاری یا فشرده‌سازی داده‌ها) به‌کار گرفته شده و قابلیت‌های آن نسبت به مدل مبتنی بر رویداد سنجیده شود. در قدم بعد باید روش‌هایی برای کاهش سربار مدل پیشنهادی ارائه کرد. به‌عنوان مثال یک راه حل برای کاهش سربار حافظه مصرفی، استفاده از ابزاریست که برای تخمین پشته برنامه در TinyThread به‌کار رفته است [۱۲]. تعمیم مدل پیشنهادی به سایر سیستم عامل‌های مبتنی بر رویداد که در سیستم‌های تعبیه‌شده استفاده می‌شوند و توسعه کتابخانه توابع بلوکه‌شونده برای برنامه‌نویس می‌تواند گام‌های بعدی در توسعه این مدل پیشنهادی باشد.

کد مربوط به تغییرات داده‌شده در سیستم عامل TinyOS توسط نویسندگان این مقاله در آدرس اینترنتی [۱۶] قرار داده شده است.

## مراجع


[1] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, and K. Pister, "System architecture directions for networked sensors," *SIGOPS Oper. Syst. Rev.*, vol. 34, no. 5, pp. 93-104, Dec. 2000.

[2] G. Werner-Allen, K. Lorincz, J. Johnson, J. Lees, and M. Welsh, "Fidelity and yield in a volcano monitoring sensor network," in *Proc of OSDI*, vol. 6, pp. 381-396, Nov. 2006.

[3] J. Hicks, J. Paek, S. Coe, R. Govindan, and D. Estrin, "An easily deployable wireless imaging system," in *Proc of ImageSense 08*, vol. 6, pp. 20-25, Nov. 2008.

[4] C. Sadler and M. Martonosi, "Data compression algorithms for energy constrained devices in delay tolerant networks," in *Proc of SenSys 06*, vol. 18, pp. 265-278, Nov. 2006.

[5] A. Dunkels, B. Gronvall, and T. Voigt, "Contiki - a lightweight and flexible operating system for tiny networked sensors," in *Proc. 29th Annual IEEE Int. Conf. on Local Computer Networks, LCN'04*, vol. 10, pp. 455-462, Nov. 2004.

[6] Micrium.uc/os-ii, *The Realtime Kernel*, Available at http://www.micrium.com/page/products/rtos/os-ii/.
[7] R. Barry, *FreeRTOS, A FREE Open Source RTOS for Small Embedded Realtime Systems*, Available at http://www.freertos.org.
[8] H. Abrach, S. Bhatti, J. Carlson, H. Dai, J. Rose, A. Sheth, B. Shucker, and R. Han, "MANTIS: system support for multimodal networks of in-situ sensors," in *Proc. 2nd ACM Int. Workshop on Wireless Sensor Networks and Applications*, vol. 2, pp. 50-59, Sep. 2003.
[9] H. Cha, S. Choi, I. Jung, H. Kim, H. Shin, J. Yoo, and C. Yoon, "RETOS: resilient, expandable, and threaded operating system for wireless sensor networks," in *Proc of IPSN 07*, vol. 6, pp. 148-157, Apr. 2007.
[10] D. Gay, P. Levis, R. von Behren, M. Welsh, E. Brewer, and D. Culler, "The NesC language: a holistic approach to networked embedded systems," in *Proc. Conf. on Programming Language Design and Implementation, ACM Press*, vol. 3, pp. 1-11, New York, USA, Jun. 2003.
[11] E. Trumpler and R. Han, "A systematic framework for evolving TinyOS," in *Proc. IEEE Workshop on Embedded Networked Sensors, EmNets2006*, vol. 3, pp. 61-65, May 2006.
[12] W. P. McCartney and N. Sridhar, "Abstractions for safe concurrent programming in networked embedded systems," in *Proc. of the 4th Int. Conf. on Embedded Networked Sensor System*, vol. 4, pp. 167-180, Oct. 2006.
[13] C. Duffy, U. Roedig, J. Herbert, and C. J. Sreenan, "Adding preemption to TinyOS," in *Proc. of the Fourth Workshop on Embedded Networked Sensors, EmNets2007*, vol. 4, pp. 88-92, Cork, Ireland. ACM Press, Jun. 2007.
[14] A. Dunkels, O. Schmidt, and T. Voigt, "Using protothreads for sensor node programming," in *Proc. of the Workshop on Real-World Wireless Sensor Networks*, vol. 1, pp. 47-51, Stockholm, Sweden, Jun. 2005.
[15] B. L. Titzer, D. K. Lee, and J. Palsberg, "Avrora: scalable sensor network simulation with precise timing," *Information Processing in Sensor Networks*, vol. 4, no. 1, pp. 477-482, Apr. 2005.
[16] *TinyOS 2.x Index of Contributed Code*, Available at http://docs.tinyos.net/index.php/TinyOS_2.x_index_of_contributed_code

**سیدمیثم خضری** در سال ۱۳۸۳ مدرک کارشناسی مهندسی نرم‌افزار را از دانشگاه آزاد اسلامی واحد قزوین دریافت نمود و در سال ۱۳۸۷ کارشناسی ارشد خود را در رشته مهندسی فناوری اطلاعات (گرایش شبکه‌های کامپیوتری) در دانشگاه یزد به پایان رساند. زمینه‌های تحقیقاتی مورد علاقه وی سیستم‌عامل، مدیریت شبکه‌های کامپیوتری و امنیت آن می‌باشد.

**مهدی آقا صرام** در سال ۱۳۵۴ مدرک کارشناسی مهندسی صنایع خود را از دانشگاه صنعتی شریف و در سال ۱۳۵۶ مدرک کارشناسی ارشد مهندسی کنترل و فناوری تست سیستم‌ها را از دانشگاه ویلز انگلستان اخذ نمود. سپس در سال ۱۳۵۸ مقطع دکترای خود را در زمینه خطایابی خودکار در سیستم‌های آنالوگ، در دانشگاه ویلز انگلستان به پایان رسانید. بین سالهای ۱۳۶۰ تا ۱۳۶۵ در موسسه علوم کامپیوتر استرالیا در زمینه مهندسی سیستم و طراحی سیستم اطلاعات یک زیردریایی مدرن فعالیت نموده است. دکتر صرام فعالیت آکادمیک خود را از سال ۱۳۶۹ در بدو تأسیس دانشگاه یزد در دانشکده مهندسی برق و کامپیوتر شروع نموده و هم‌اکنون نیز عضو هیأت علمی این دانشکده می‌باشد. ایشان در بین سالهای ۱۳۷۹ تا ۱۳۸۲ در دانشگاه‌های سیدنی, مک‌کواری و دانشگاه وسترن سیدنی به تدریس و تحقیق مشغول بوده‌اند. زمینه‌های علمی مورد علاقه نام‌برده مهندسی نرم افزار در مقیاس وسیع, شبکه‌های کامپیوتری و شبکه‌های حس‌گر بی‌سیم می‌باشد.

**فضل‌الله ادیب‌نیا** تحصیلات خود را در مقاطع کارشناسی و کارشناسی ارشد سخت افزار کامپیوتر به‌ترتیب در سالهای ۱۳۶۵ و ۱۳۶۸ در دانشگاه صنعتی اصفهان و صنعتی شریف به پایان رساند. سپس به دوره دکترای مهندسی کامپیوتر در دانشگاه برمن آلمان وارد گردید و در سال ۱۳۷۸ موفق به اخذ درجه دکترا از دانشگاه مذکور گردید. دکتر ادیب‌نیا هم‌اکنون استادیار دانشکده مهندسی برق و کامپیوتر دانشگاه یزد می‌باشد. زمینه‌های تحقیقاتی مورد علاقه ایشان شامل موضوعاتی مانند شبکه‌های کامپیوتری ، امنیت شبکه، سیستم‌های توزیعی و سیستم‌عامل می‌باشد.