SensorScheme: اتوماسیون مدیریت زنجیره تامین با استفاده از شبکه های سنسور بی سیم

خلاصه:

همانگونه که پیشرفت های اخیر با تکنولوژی RFID نشان می دهد، تجارت مدیریت زنجیره تامین می تواند به شدت از اتوماسیون نفع ببرد. انتظار می رود که استفاده از تکنولوژی شبکه سنسور بیسیم بتواند جهش بعدی در کارایی و کیفیت خدمات را به ارمغان بیاورد. با این که، نرم افزار سیستم WSN فعلی هنوز عملکرد، انعطاف و ایمنی مورد نیاز را ارائه نمی دهد. این مقاله یک سناریو را مطرح میکند که نشان می دهد چگونه فناوری WSN می تواند از مدیریت زنجیره تامین سود ببرد، و SensorScheme را که پلت فرم برای تحقق سناریو است ارائه کند. SensorScheme یک پلت فرم WSN همه منظوره است، که محیط اجرایی امنی را برای برنامه های dynamically loaded فراهم می کند. این از عناصر برنامه نویسی سطح بالا مانند ارتباطات marshaled، مدیریت حافظه اتوماتیک و امکانات چند پردازشی استفاده می کند. SensorScheme از حافظه اندک موجود در نود های WSN به طور موثر استفاده میکند تا به برنامه های بزرگتر و پیچیده تر در پی تکنولوژی جدید اجازه ی اجرا بدهد. ما یک اجرای SensorScheme ارائه میکنیم و نتایج تجربی را برای نشان دادن فشردگی، سرعت عمل و بازده انرژی تهیه می کنیم.

مدیریت زنجیره تامین یک تجارت پیچیده است که شامل بسیاری از احزاب و حجم زیادی از کالاها می شود. جای تعجب نیست که بررسی دستی کالا ها منجر به خطای انسانی می شود که باعث کاهش قابل توجه درآمد می شود. اتوماسیون می تواند باعث بهبود زنجیره تامین، کارایی و حجم معاملات بالاتر می شود. براساس یک مطالعه اخیر [13]، کاربرد اخیر فناوری RFID در حال حاضر تاثیر زیادی بر زنجیره تامین خرده فروشی دارد.

دیگر تحولات اخیر در ارتباطات بی سیم کم قدرت، تکنولوژی جدیدی را بوجود آورده است: شبکه های حسگر بی سیم، شامل دستگاه های محاسباتی کوچک مجهز به سنسورهای کوچک و قابلیت های ارتباطات بی سیم. متفاوت از

RFID، این دستگاه ها با باتری عمل می کنند و می تواند با هر وسیله دیگری در نزدیکی ارتباط برقرار کند، محیط خود را حس می کند و به طور مداوم به وضعیت درک شده جهان اطرافشان پی ببرند.

شبکه های حسگر بی سیم یک امید بزرگ برای تجارت مدیریت زنجیره تامین است. گره های WSN را می توان به جعبه ها، کانتینر رول، پالت ها و کانتینر های حمل متصل کرد تا به عنوان دستگاه های ردیابی حمل و نقل فعال عمل کنند. این دستگاه ها می توانند به طور فعال فرایند حمل و نقل را کنترل کنند و به شرایط بررسی مناسب کالاها مانند درجه حرارت برای غذاهای تازه رسیدگی کنند. علاوه بر این، این دستگاه ها می توانند آسیب ناشی از شوک های ناگهانی، یا باز شدن کانتینر ها و دیگر اشکال نقض قرارداد را تشخیص دهند. این باعث بهبود کیفیت خدمات و کارایی بیشتر می شود که به نوبه خود منجر به کاهش هزینه حمل و نقل می شود.

اگرچه سیستم عامل های سخت افزاری شبکه بیسیم در حال حاضر برای دستگاه های ردیابی حمل و نقل فعال مناسب هستند، تکنولوژِ جدید نرم افزار سیستم WSN فاقد مجموعه ای از ویژگی های مناسب است. در این مقاله ما یک پلت فرم به نام SensorScheme ارائه می کنیم که می تواند نیازهای ارائه شده توسط سناریوهای لجستیک پیگیری فعال را برطرف کند. SensorScheme یک مترجم برای اجرای کد برنامهdynamically loaded برای سیستم عامل WSN بر اساس زبان برنامه نویسی Scheme است. این محیط اجرایی امن را ارائه می دهد که در آن برنامه های بدعمل نمی توانند دستگاه را خراب کنند و با امکانات برنامه نویسی سطح بالا مثل جمع آوری زباله، ارتباط با مرتب کردن خودکار آیتم های داده مجهز شده و برای اجرای چندین کنترل از راه دور و فعال سازی مسدود کردن تماس های I / O همکاری می کند. علاوه بر ردیابی فرآیندهای لجستیکی، SensorScheme می تواند در بسیاری از برنامه های کاربردی WSN "سنتی" استفاده کند.

بقیه مقاله به شرح زیر است: بخش 2 یک سناریوی کاربردی را ارائه می دهد، که پس از آن جدیدترین تکنولوژی برای تحقق این برنامه در بخش 3 بررسی می شود.سپس، بخش 4 طراحی SensorScheme را شرح می دهد و به دنبالدر بخش 5 درباره تکنیک های پیاده سازی برای سناریو بحث می شود. سپس عملکرد SensorScheme را در بخش 6 ارزیابی می کنیم و نتیجه گیری می کنیم (بخش 7).

2 سناریو

فن آوری شبکه های حسگر بی سیم می تواند در صنعت مدیریت زنجیره تامین، زمانی که به عنوان دستگاه های ردیابی حمل و نقل فعال (ATTDs) متصل به اقلام حمل و نقل مجدد (RTI) مانند جعبه، کانتینر های rolling ، پالت و کانتینر های حمل و نقل، سودمند باشد. برای نشان دادن استفاده از چگونگی ATTD ما در حال حاضر در مورد یک سناریوی حمل و نقل کوچک بحث می کنیم. یک محموله ی موز را در نظر بگیرید که از مزرعه در نزدیکی ریودوژانیرو، برزیل به یک مرکز توزیع سوپرمارکت در روتردام منتقل می شود. موز در جعبه های بسته بندی شده روی پالت قرار گرفته می شوند، و هر کدام با یک دستگاه ردیابی مجهز شده اند. در اوایل صبح، این پالت ها در کامیون ها (متعلق به شرکت حمل و نقل موز یا BTC) از مزرعه به بندر بارگیری در بندر حمل می شوند، جایی که آنها را به کانتینر های حمل و نقل حمل می کنند که آنها را تا رسیدن به مرکز توزیع زنجیره ای سوپرمارکت حمل می کند. در طی کل سفر موز باید در هوای خنکک بین 10 تا 15 درجه سانتیگراد و از منابع گاز اتیلن مانند دانه های قهوه ای تازه نگهداری شود که بر روند رسیدن تأثیر می گذارد.

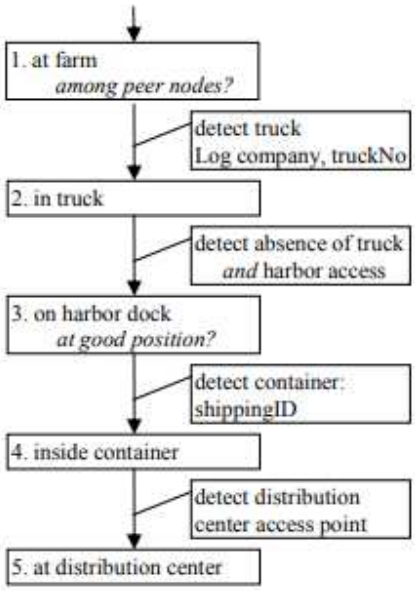
در طی فرآیند حمل و نقل از مزرعه به مرکز توزیع - که ما به آن یک سفر می گوییم - تعدادی از موارد نظارت می شوند:

1. سنسورهای دما بر روی ATTD ها دمای محیط را در فواصل 1 دقیقه اندازه گیری می کنند و دمای اندازه گیری شده را در پرونده ورودی دستگاه ذخیره می کنند. اگر دمای بیش از محدوده مجاز باشد، دستگاه هشدار را سیگنال می کند، بنابراین می توان اندازه گیری ها را بلافاصله انجام داد.
2. هر دستگاه ردیابی پالت با دستگاه های اطراف آن ارتباط برقرار می کند تا اطمینان حاصل شود که آیا هر یک از آنها دانه های قهوه یا سایر محصولات مضر را حمل می کنند. هنگامی که یک پالت به یک کانتینر بارگیری می شود، دستگاه همچنین درخواست می کند که آیا کانتینر می تواند کانتینر دیگری حمل کالاهای مضر را در یک فاصله معین (از 10 متر) پیدا کند. اگر دانه های قهوه در نزدیکی یافت می شوند، دستگاه ردیابی این را در فایل ورودی آن ذخیره می کند و هشدار را نشان می دهد.

3. در طی کل سفر، ATTD در هر پالت پیروی از طرح حمل و نقل را بررسی می شود. در هر مرحله از سفر، تایید می کند که آیا به کامیون صحیح بارگذاری شده است، و در انبار صحیح تخلیه شده است، که در شکل 1 نشان داده شده است. هر انتقال به یک مرحله جدید از سفر تسطیح شده و هر زمانی که حمل و نقل به درستی انجام نمی شود ، و یا در زمان محدودیت دارد، دستگاه ها یک هشدار را به سیگنال در می آورند.

موز های ما از تعدادی مرحله در حین حمل از مزرعه به مرکز توزیع عبور می کند، همانطور که در شکل 1 نشان داده شده است. در هر مرحله، بسته به شرایط محلی، روش متفاوتی برای تأیید باید انجام شود.

شکل 1 نمودار پروسه حمل و نقل



در حالی که یک پالت در مزرعه منتظر بارگیری در کامیون است، تلاش می کند که تایید کند به درستی، در نزدیکی سایر پالت ها که باید به یک کامیون بارگیری شوند، بارگیری شود. این کار را با مقایسه مقصد و محتویات آن با (اکثریت) گره های همتا در سایر پالت های نزدیک انجام می دهد. هنگامی که یک پالت به طور صحیح قرار نگرفته باشد یا هیچ گره همتایی یافت نشود، باید هشدار را به صدا در آورد.

سپس ، پالت ها به کامیون حمل می شوند تا آنها را به بندر منتقل کنند. گره ها می توانند بارگیری را با " شنیدن " دستگاه دیگری را که در داخل کامیون قرار دارد، تشخیص دهد. در کامیون، هر دستگاه پالت از دستگاه کامیون شرکت

ها و شناسه های کامیون ها درخواست می کند و آنها را در فایل ورود به سیستم (همراه با زمان فعلی) ثبت می کند. دستگاه های پالت با اطلاعاتی که قبل از 6 صبح به همراه کامیون های شرکت BTC حمل می شوند، برنامه ریزی می شوند و اگر هر یک از شرایط رخ ندهد، هشدار می دهند.

گرچه در کامیون، گره های پالت نباید هر چیزی را تأیید کنند، زیرا هیچ تغییری در حالت رخ نخواهد داد تا زمانی که آنها خارج شوند. آنها باید تشخیص دهند که از کامیون خارج می شوند، که می توانند به دلیل عدم وجود کامیون و حضور زیرساخت بی سیم (نقطه دسترسی) اسکله بارگیری بندرگاه نتیجه گیری شود. اگر بندر گاه صحیح شناسایی نشده باشد یا قبل از رسیدن پالت، مدت طولانی طول بکشد، باید هشدار داده شود.

هنگامی که در اسکله خالی می شوند، ATTDها دوباره تأیید می کنند که آیا آنها به طور صحیح قرار گرفته اند تا دوباره به کانتینر های حمل و نقل بارگیری شوند .اسکله مجهز به زیرساخت های پیشرفته الکترونیکی است که می تواند موقعیت هر پالت را ردیابی کند و بر اساس این، هر پالت تأیید می کند که آیا در جای درست است. هنگامی که نادرست قرار می گیرد، می تواند به طور مستقیم یک پیام هشدار را به زیرساخت اسکله منتقل کند که به کارگران برای تصحیح آن اطلاع می دهد. بر خلاف کامیون ها، نقاط دسترسی می توانند از اتصال مستقیم رادیویی خارج باشند، فقط از طریق چند هاپ متصل می شوند و به چندین پروتکل ارتباطی چند هاپی متفاوت برای ارتباط برقرار کردن نیاز دارند.

برای آخرین مرحله حمل و نقل، پالت ها به کانتینر ها بارگذاری می شوند. این ها را می توان با یک شناسه حمل و نقل مطابق برنامه ریزی شده در هر کانتینر تشخیص داد. سرانجام، هنگامی که کانتینر به مرکز توزیع می رسد، ATTD های پالت نقطه دسترسی مرکز توزیع و تبدیل حالت را درک می کنند. به عنوان پایگاه اصلی برای این حمل و نقل، نقطه دسترسی مرکز توزیع روتردام با استفاده از پروتکل های اختصاصی بی سیم، فقط اجازه ی دسترسی به کالاهای متعلق به سوپرمارکت زنجیره ای را می دهد. باز هم این نیاز به پروتکل های ارتباطی مختلف برای انجام تایید سفر دارد.

هنگامی که خطاها در طول سفر تشخیص داده می شوند، ATTD ها هشدار را سیگنال می دهند. بسته به مرحله حمل و نقل، روش های مختلفی برای افزایش هشدار مورد نیاز است. در حالی که پالت ها در خارج از کامیون قرار دارند، منتظر هستند تا بارگیری شوند، صدای زنگ و چراغ های چشمک زدن، توجه کارگران را که می توانند مشکل را حل کنند، جلب می کنند. اما در داخل کامیون، هشدار باید به راننده در کابین کامیون اطلاع داده شود. در ابتدای هر مرحله جدید، روش هشدار مناسب انتخاب شده است که توسط هر یک از منابع خطا که می تواند در دستگاه رخ دهد، از جمله فرآیندهای دما و پروسه مجاورت قهوه، استفاده شود.

2.1 برنامه نویسی ATTDs

حالا که ما مشخص کردیم که قابلیت ATTD ها باید نشان داده شود، سوال این است که چگونه دستگاه را برای دستیابی به این رفتار برنامه نویسی می کنند. پیاده سازی ساده ممکن است شامل یک برنامه ثابت تعبیه شده در دستگاه باشد که باید با چند پارامتر مانند محتوا، مقصد، جزئیات حامل حمل و نقل، تکمیل شود. این زمانی که محموله ارسال می شود، برنامه نویسی می شود. برای تعدادی از سناریوهای حمل و نقل محدود این ممکن است کافی باشد، اما قطعا نمی تواند تجارت مدیریت زنجیره تامین در تمام تنوع های آن را برآورده کند. حداکثر انعطاف پذیری با حداکثر آزادی به دست می آید، یعنی، کل برنامه تایید سفر را تعیین کرده و آن را قبل از شروع سفر در هر گره نصب کنند.

این انعطاف پذیری بسیاری از روش های تایید دیگر برای ATTD ها را امکان پذیر می کند. به عنوان مثال، باز کردن درب های کانتینر می تواند از طریق یک تغییر ناگهانی در دمای محیط تشخیص داده شود. متناوبا، با استفاده از سنسورهای شتاب، شوک های ناگهانی می توانند شناسایی و وارد سیستم شوند تا دریابند که کدام حمل کننده مسئول کالا های آسیب دیده ی شکننده مانند لوازم الکترونیکی مصرفی است. علاوه بر این، با پشتیبانی از زیرساخت های محلی، ATTD ها می توانند برای اتصال به خانه از طریق اتصال به اینترنت برای گزارش وضعیت خود برنامه نویسی شوند.

ایمنی نیز باید در نظر گرفته شود. معمولا پالت ها متعلق به یک سازمان کارآزموده است و توسط آن مدیریت می شوند. فقط اگر کاربران (حمل کننده ها) قادر به" break" دستگاه ها نباشند (یعنی تغییر عملکرد نرم افزار خود) این سناریو می تواند یک سناریوی ریالیستیک باشد. در حال حاضر تنها چیز مورد نیاز راهی رسا، محکم و امن برای بیان این برنامه های سفر است.

3. الزامات برنامه و جدیدترین تکنولوژی

از سناریو که در بالا توضیح داده شده است، می توانیم تعدادی از الزامات را که اجرای احتمالی باید با آنها مطابق باشد را به دست آوریم. ما در مورد این ها بحث خواهیم کرد و بررسی میکنیم که تا چه اندازه تکنولوژی جدید به این مسائل توجه کرده است. به عنوان یک پلت فرم پیاده سازی برای دستگاه های ردیابی فعال حمل و نقل، ما یک سخت افزار WSN سهام، مانند mica motes را فر می کنیم [9].

در شروع هر سفر، هر ATTD باید با استفاده از رابط بی سیم برای سفر آینده برنامه نویسی شده باشد. در حال حاضر چندین مکانیسم به روز رسانی کد بی سیم برای پلت فرم های شبکه حسگر بی سیم توسعه یافته است. این کار با جایگزینی کل تصویر برنامه به عنوان یک واحد کامل صورت میگیرد. پلت فرم TinyOS [8] برای شبکه های حسگر شامل دو تکنولوژی از جمله XNP [3] و Deluge [10] است. متاسفانه، این رویکرد به دلایل مختلف برای سناریوی ما مناسب نیست.

اول، تصاویر برنامه معمولا ده ها کیلابایت هستند، و انتقال این اطلاعات چند دقیقه طول می کشد (با توجه به [10]، [14]). این می تواند تا حدودی با استفاده از یکی از الگوریتم های فشرده سازی و دیفرانسیل مختلف بهبود یابد ([RSYNC [11]، MOAP [17]، FlexCup [14]) .

دوم، این مکانیزم های به روز رسانی کد هدف دارد تمام باینری را با یک سیستم جدید، شامل سیستم عامل که برنامه ریزی کار، دسترسی سخت افزاری سطح پایین، پروتکل های شبکه و حتی مکانیسم به روز رسانی کد را کنترل می کند جایگزین کند که نباید توسط کاربران ATTD قابل تغییر باشد. چندین پلت فرم WSN ماژول های قابل بارگذاری

زمانبندی ([4] Contiki، 7] SOS] را ارائه می دهند، اما این هنوز هم موجب دسترسی نامحدود به کل دستگاه را می شود.

یک رویکرد مناسب تر، استفاده از یک مترجم یا ماشین مجازی است. از یک طرف، تنها کد برنامه باید به دستگاه ها منتقل شود، که به طور قابل توجهی حجم کد منتقل شده را کاهش می دهد. علاوه بر این، از آنجا که نماد کد یک متغیر طراحی است، نه یک ویژگی سخت افزار، می توان آن را به طور خاص می تواند برای انواع برنامه های مورد انتظار که برای آن ساخته شده اند فشرده سازی شود.

علاوه بر این، مترجم یا ماشین مجازی به عنوان چیزی که معمولا "جعبه شن " نامیده می شود، عمل می کند و سخت افزار را از برنامه های تفسیری محافظت می کند. بدین ترتیب برنامه های ناسازگار یا باگ دار از تغییر وضعیت هر حالتی در کنار خودشان جلوگیری می شوند و نمی توانند یک دستگاه را خراب کنند و یا عملکرد های مهم دستگاه را مختل کنند.

مشهور ترین معماری ماشین مجازی که اغلب در محاسبات جریان اصلی مورد استفاده قرار میگیرد، ماشین مجازی جاوا Sun است.مشخص شده است به عنوان یک پلت فرم شبکه حسگر استفاده شده است (خورشید 18] SPOTs]). SensorWare [2] یک پلت فرم دیگر مبتنی بر تفسیر و sandboxing برای WSN ، با استفاده از زبان TCL است. با این حال، هر دو از این سیستم عامل نیاز به پلت فرم های غنی از منابع بیشتر از آنچه است که ما برای سناریو برنامه ما در نظر گرفته ایم.

Mat'e / Bombilla [12] یک ماشین مجازی است که به طور خاص برای دستگاه های WSN محدود به حافظه طراحی شده است. متأسفانه، Mat'e تنها می تواند شامل برنامه های خیلی کوچک باشد. برنامه ها در زمینه هایی مربوط به منابع رویداد سازمان یافته اند، که شامل یک آرایه دستور العمل 128 بایت است که در پاسخ به رویدادهای راه انداز اجرا می شود. هر زمینه (6 در مجموع) دارای پشته اپراند خود ، مجموعه ای از 8 متغیر (عدد صحیح) محلی و یک بافر بسته است. با این حال، پیاده سازی سناریو ما نیاز به تخصیص حافظه توسط برنامه بارگذاری شده و dynamic loading روش ها و پروتکل های ارتباطی سفارشی دارد، که در بخش 5 نشان داده شده است. این

محدودیت های پیچیدگی شدید، از اجرای سناریو برنامه ما را روی Mat´e ممانعت میکند. علاوه بر این، مدل همزمانی Matte چندین کار مستقل را انجام نمی دهد که به صورت موازی اجرا شوند.

4. SensorScheme

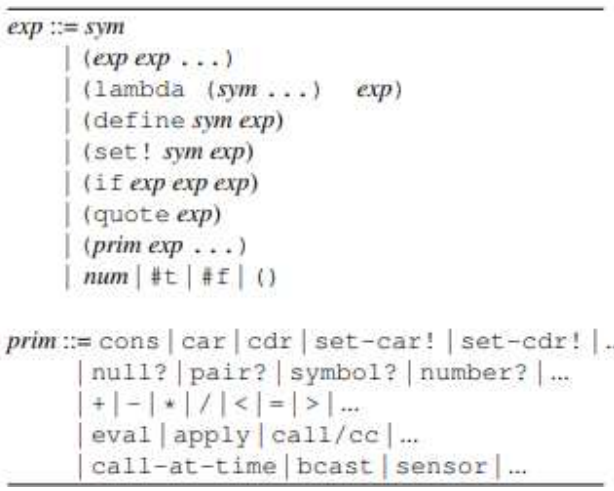
ما SensorScheme را به عنوان یک پلتفرم تفسیری جدید برای WSN پیشنهاد می کنیم که می تواند برای اجرای سناریو برنامه کاربردی ما استفاده شود. SensorScheme برای استفاده در پلت فرم های سخت افزاری WSN، با توجه به محدودیت منابع خود طراحی شده است. مهمتر از همه، حافظه - مخصوصا RAM ، دارای ذخیره ی کم است و مکانیسم هایی مانند حافظه مجازی برای گسترش حافظه به آنچه که عملا در دسترس است وجود ندارد. علاوه بر این، ارتباطات بی سیم اکثریت انرژی را نسبت به محاسبات مصرف می کند و باید تا حد امکان به حداقل برسد.

پلت فرم SensorScheme از معانی اجرای زبان برنامه نویسی Scheme استفاده می کند، همانطور که از نامش مشخص است. با این وجود، این اجرای زبان Scheme نیست، و ایجاد برنامه های SensorScheme نیازی به استفاده از زبان Scheme یا سینتکس ندارد. در عوض، یک سینتکس آشناتر الهام گرفته از C برای کاربران سیستم در دسترس خواهد بود. مثال های کد در بخش های زیر از سینتکس Scheme، به عنوان زبان اسمبلی موتور زمان سنج SensorScheme استفاده می کنند.

4.1 حافظه

SensorScheme به طور خاص برای اندازه کوچک حافظه سیستم عامل WSN طراحی شده است. تمام حافظه به یک مخزن کوچک از سلول های کوچک اندازه یکسان اختصاص داده شده است. این سلول ها برابر " "cons-cells Scheme هستند که هر کدام شامل دو عضو داده می شوند که می توانند برای هر مقدار دیگری مانند یک cons-cell، number، #t) booleans، (#f یا لیست خالی (()) رفرنس باشند. سلول ها می توانند ترکیب شوند تا لیست ها، درختان، لیست های مشارکت و غیره را تشکیل دهند.

مخزن حافظه سراسری اطلاعات برنامه و همچنین کد برنامه و حالت متراکم مانند پشته تماس، وابستگی متغیر محلی و سراسری و صفوف برنامه ریزی را ذخیره می کند. جمع آوری زباله سلول های استفاده نشده در مخزن حافظه را اشغال می کند.



شکل 2. گرامری برای SensorScheme

4.2 نمایش برنامه و معنا شناسی اجرا

برنامه های SensorScheme یک لیست خاص مرتبط از سلول های حافظه را که حاوی درخت سینتکس انتزاعی (AST) این برنامه هستند را شکل می دهد. شکل 2 دستورالعمل یک درخت سینتکس SensorScheme (که در علامت براکت Scheme نوشته شده است) را فهرست می کند. معانی عملیاتی این قوانین همانند برنامه عادی است. اولین قاعده، exp، مجموعه ای از اصطلاحات SensorScheme عادی را توصیف می کند. سه سازه ی اول آن نشان دهنده هسته SensorScheme lambda-calculus است: مرجع متغیر، برنامه کاربردی و جبر لامبدا. چهار ساختار بعدی فرم های خاصی هستند که لازم است برای اجرای یک Scheme حداقل کامل الزامیست: تعریف متغیر سراسری، تخصیص متغیر، ارزیابی مشروط، نقل قول لفظی. سپس فراخوانی روش اولیه و چهار قانون آخر نشان دهنده مرجع ثابت (اعداد، درست، غلط، لیست خالی) است.

مجموعه ای از ابتدایی های تعریف شده که بعضی از آنها توسط قانون دوم ارائه می شوند، شامل بسیاری از اساسی ترین شیوه های اولیه هستند و شامل (خط به خط): دستکاری cons-cell، type predicates ، حساب، جریان کنترل و I / O می باشد

برای توصیف معانی اجرای Scheme ، خواننده را به [5] ارجاع می دهیم.

4.3. برنامه ریزی کار

گره های WSN دارای طبیعت مبتنی بر رویداد ذاتی هستند. این در سیستم عامل های WSN امروز منعکس می شود. اجرای برنامه در تعدادی از کار های کوتاه مدت سازماندهی شده است که می تواند برنامه ریزی شده در پاسخ به برخی رویدادها اجرا شود. به طور کلی، کار ها تا اتمام اجرا می شود، پس از پایان دادن به کار قبلی کار بعدی شروع می شود.

SensorScheme طراحی شده است تا بر روی سیستم های عامل WSN مبتنی بر رویداد مانند TinyOS [8] یا Contiki [4] اجرا شود. SensorScheme مکانیسم برنامه ریزی خود را در بالای OS تعریف می کند. وقتی یک رویداد اتفاق می افتد، یک وظیفه SensorScheme برنامه ریزی می شود. این وظایف به ترتیب FIFO انجام می شوند. انواع حوادثی که می توانند در SensorScheme اتفاق بیفتند عبارتند از: 1) شلیک یک تایمر، 2) دریافت پیام شبکه و 3) حوادث سخت افزاری ناشی از سنسورها.

حوادث تایمر یک محاسبه را در یک لحظه از پیش تعیین شده اجرا می کنند. SensorScheme یک فرایند ابتدایی call-at-time فراهم می کند که به عنوان پارامتر زمان برنامه ریزی شده و محاسبات به عنوان یک تابع صفر-argument عمل می کند. در زمان برنامه ریزی شده، محاسبات به عنوان یک رویداد پردازنده اجرا می شود.

استفاده از رویدادهای تایمر با یک مثال بهتر نشان داده شده است. در نمونه کد در شکل 3 (a) تابع time-loop خود را در فواصل 5 ثانیه برای پخش یک پیام برنامه ریزی می کند.

```
   (define (time-loop)
(a)  (call-at-time (+ (now) 5) time-loop)
     (bcast (list 'gossip 1 2 3)))

   (define-handler (gossip a b c)
(b)  ; react to the gossip message
     ; variable src is bound to ID of sender
     ...)
```

شکل 3. مثال های کد نمونه که استفاده از وقایع تایمر و ارتباطات را نشان می دهد

4.4 ارتباطات

ارتباط شبکه بی سیم یکی از اجزای حیاتی سیستم عامل WSN است. در SensorScheme ارتباطات به گونه ای طراحی شده است که فشرده و استفاده از آن آسان باشد.

تمام داده های SensorScheme در سلول های حافظه مجموعه ای کوچک از انواع داده ها، با یک کد نوع برچسب گذاری شده اند. با استفاده از این نوع اطلاعات زمان اجرا دستگاه ها یک ساختار داده را به نمایش خطی مناسب برای ارتباط شبکه تبدیل می کند. پس از پذیرش، گیرنده می تواند یک ساختار داده ی مشابه از نمایش خطی را (یک کپی از) بازسازی کند. این یک تکنیک آشنا به نام marshalling است، که همچنین در فن آوری های دیگر مانند CORBA یا Java RMI استفاده می شود.

ارتباطات SensorScheme مشابه شیوه پیام فعال TinyOS عمل می کند. یک پیام شامل یک نماد هدر و تعدادی از آیتم های داده است. هدر پیام یک نماد است و به تابع سراسری اشاره دارد که پیام را اداره می کند و آیتم های داده در پیام به عنوان پارامتر به تابع رسیدگی کننده عمل می کند. روش اولیه bcast به سادگی یک پیام را به تمام گره ها در محدوده انتقال می فرستد. یک پارامتر واحد را می پذیرد: یک لیست حاوی محتوای پیام. برای نمونه کد که شامل bcast است، شکل 3 را ببینید. اولیه ی bcast، محتوا پیام را به صورت خطی به یک یا چند بسته فیزیکی بسته به اندازه محتوای پیام کدگذاری می کند.

گیرندگان این پیام محتوای هر بسته را به آیتم های داده مربوطه رمزگشایی می کنند. سپس رسیدگی کننده ی پیام که توسط نماد هدر جستجو و برنامه ریزی شده به عنوان یک رسیدگی کننده ی رویداد اجرا می شود. نمونه کد از

شکل 3 (که در تمام گره ها در WSN لود می شود) نشان می دهد که ارتباط چگونه برقرار می شود. گره ها پیامی حاوی gossip header و سه آیتم داده، مقادیر 1، 2 و 3 را پخش می کنند. دریافت gossip روش زمان بندی، که ID منبع گره ارسال را به عنوان پارامتر مجازی محدود به src فرض می کند و سه عنصر داده را به a، b و c متصل می کند.

ارتباط کد برنامه SensorScheme ساده است: ساختار داده ها شرح می دهند که کد را می توان در داخل پیام SensorScheme ، و روی دریافت 'eval'-ed برای بارگذاری و اجرای بسته بندی کرد. یک روش ابتدایی به نام eval-handler وجود دارد که فقط آن را انجام می دهد، و این باعث می شود که یک گره SensorScheme ''empty راه اندازی شود. ابتدایی eval-handler تعریف شده است:

```
(define-handler (eval-handler sexpr)
(eval sexpr))
```

و می تواند به روش زیر استفاده شود

```
(bcast (list ’eval-handler
’(define sqr (lambda (x) (* x x)))))
```

توجه داشته باشید که رابط ارتباطی SensorScheme محدودیتی در تعداد آیتم های داده یا اندازه هر آیتم داده در یک پیام ندارد. از این رو، نمی توان فرض کرد محتویات پیام در داخل یک بسته استفاده شده توسط رابط شبکه فیزیکی قرار گیرد و باید چندین بسته استفاده شود. ما به دلیل محدودیت فضا، در مورد جزئیات رمزگذاری و باز کردن صحیح بسته بندی این پیام ها بر روی گیرنده در این مقاله بحث نخواهیم کرد.

5. بحث

با استفاده از SensorScheme، ATTD ها می توانند برای انجام وظایف مورد بحث در بخش 2 برنامه ریزی شوند. برای نشان دادن این موضوع، ما در حال حاضر درباره اجرای SensorScheme در بخشی از این سناریو بحث خواهیم کرد. مثال نشان می دهد که چگونه SensorScheme آسان ساختن پروتکل های ارتباطی را ایجاد می کند و ایجاد سلول را مسدود می کند،که مخصوصا برای برنامه های WSN ارتباطی مفید است.

به یاد بیاورید که در حالی که پالت ها با موزها در مزرعه است، منتظر هستند که به کامیون ها بارگیری شود، با هرکدام آنها را بررسی می کنند تا جایگذاری درست را بررسی کنند. اگر مقصد و محتوا پالت ها با همتایان خود مطابق باشد پالت ها به درستی قرار داده می شوند.

کد SensorScheme که در شکل 4 ارائه شده است شامل تعدادی از مراجع تعریف شده در استاندارد Scheme [1] یا یکی از srfi [16] است و ما آنها را بدون اشاره بیشتر به عملیات آنها استفاده می کنیم.

شکل 4 اجرای یک روش SensorScheme را به نام peer-verify را نشان می دهد. این روش لیست اتصال 2 جفت کلید-ارزش را می پذیرد و با تمام همسایگان مستقیم ارتباط برقرار می کند تا ورودی های فرهنگ لغت آن ها را از کلیدهای داده شده پیدا کند. اگر هر یک از مقادیر همسایه ها از پارامترهای داده شده متفاوت باشد، تابع هشدار فعلی فراخوانده می شود(خطوط 2-7 را ببینید).

اکثر کارهای واقعی در روش peer-dict انجام می شود (خطوط 10-21). این یک تماس مسدود کننده است که مقدار کلید و زمان باقیمانده را به عنوان پارامترها در نظر میگیرد و بعد از گذشت ثانیه ها با مقدار های مربوط به همسایگان آن، بر گردانده می شود.

SensorScheme تداوم را فراهم می کند که می تواند برای پیاده سازی یک مکانیسم همزمان سازی کم وزن استفاده شود. این اجازه می دهد تا بدون استفاده از حافظه بیشتر از حد لازم برای در برگرفتن حالت برنامه، تعداد دلخواه از عملیات I / O مسدود کننده همزمان انجام شود. ما نمی توانیم در مورد معناشناسی تداوم و اولیه ی call cc / بحث کنیم، برای توصیف کامل تداوم ها، خواننده را به [6] ارجاع می دهیم.

تابع peer-dict درخواستی را برای تمام همسایگان (خط 12) ارسال می کند که حاوی یک شناسه درخواست منحصر به فرد (ایجاد شده در خط 11) و کلید درخواست شده است، و ID درخواست را در فرهنگ لغت waiting-reqs (خط 13-14) ذخیره می کند. فراخوانی call / cc در خط 15، یک تداوم ایجاد می کند، برای برگداندن تماس گیرنده تابع بعد از وقفه استفاده می شود. در خط 17 یک تایمر برای پایان دادن به وقفه سیگنال تنظیم شده است.

سرانجام، یک تماس برای خروج (خط 21) کار فعلی را قطع می کند، به این ترتیب می تواند رویدادهای دیگر را پردازش کند در حالی که peer-dict مسدود شده است.

پیام پخش شده در خط 12 توسط handler-dict-hdl در تمام گره های دریافتی (خطوط 24-26) انجام می شود. این گره ها به سادگی با یک پیام peer-dict-rpl با شناسه فرستنده، شناسه درخواست اصلی و ارزش فرهنگ لغت سراسری مرتبط با کلید پاسخ می دهند.

پس از پذیرش پیام های peer-dict-rpl در دستگاه درخواست کننده (خطوط 30-33)، شناسه درخواست را در dictionary-waiting-reqs مطرح می کند و لیست ارزش را با مقداری که تازه دریافت شده است (خط 33) گسترش می دهد.

هنگامی که پس از ثانیه های وقفه، تایمر منقضی می شود (خط 18 تا 20)، شناسه درخواست یکبار دیگر جستجو می شود و از فرهنگ لغت حذف می شود. سپس، با فراخوانی به تدوام مرتیط با متغیر k، روش peer-dict ، با لیست ارزش ایجاد شده در فراخوانی های بعدی peer-dict-rpl به عنوان مقدار بازگشتی برگردانده می شود.

عدم بررسی کد خطا عمدی است و یکی از پیامدهای استفاده از SensorScheme را نشان می دهد. به عنوان مثال، در بررسی کننده ی peer-dict-hdl، اگر ورود کلید درخواست شده در فرهنگ لغت رخ ندهد، هیچ پیام پاسخی نباید ارسال شود. این می تواند بدون هیچ گونه شناسایی خطای صریح و یا کد رسیدگی به دست آید: Assoc # f (false) را برمیگرداند و cdr #f را در یک خطا دریافت می کند، که بلافاصله رسیدگی کننده را بدون ارسال پیام، متوقف می کند.

```
(define (peer-verify alist)
  (if (not (every (lambda (kv)
                    (every (lambda (v)
                             (eq? v (cdr kv)))
                           (peer-dict 5 (car kv))))
                  alist))
      (alerter 'itinerary-error 'peer-verify)))

; requests the value of given keys from all neighbors
(define (peer-dict timeout key)
  (let ((reqid (rand)))
    (bcast (list 'peer-dict-hdl reqid key))
    (set! waiting-reqs (cons (cons reqid ())
                             waiting-reqs))
    (call/cc
     (lambda (k)
       (call-at-time (+ (now) timeout)
         (lambda ()
           (k (cdr (assoc-and-remove!
                    reqid waiting-reqs)))))
       (exit)))))

; handler invoked at neighbors
(define-handler (peer-dict-hdl reqid key)
  (bcast (list 'peer-dict-rpl src reqid
               (cdr (assoc key global-dict)))))

; handler receiving values from neighbors
; called at requesting node
(define-handler (peer-dict-rpl dst reqid val)
  (when (= dst id)
    (let ((req (assoc reqid waiting-reqs)))
      (set-cdr! req (cons val (cdr req))))))
```

شکل 4 مثال کد منبع برنامه

6. ارزیابی

ما SensorScheme را بر روی یک پلت فرم سخت افزاری شبکه حسگر اجرا کرده ایم و آن را برای تجزیه و تحلیل استفاده از حافظه، تفسیر مخارج کلی و هزینه انرژی در هنگام اجرای مثال 4 استفاده کردیم.

6.1 پیاده سازی

ما یک پیاده سازی SensorScheme اولیه بر روی یک پلت فرم شبکه سنسور بی سیم بر اساس یک میکروکنترلر MSP430 ساخته ایم که شامل 10 کیلو بایت RAM و 48 کیلو بایت فلش برنامه است. این دستگاه شامل تراشه فرستنده nRF905 Nordic است که با سرعت 50 کیلوبیت در ثانیه ارتباط برقرار می کند.

پیاده سازی سرعت توجه زیادی به سمت بهینه سازی جلب نکرده است، و به نظر می رسد مکان هایی برای بهبود در آمار های زمان اجرا ارائه شده در این بخش وجود دارد.

هر سلول 4 بایت را می گیرد و در آدرس های 4 بایتی مرتب می شوند. فضای آدرس کلی سلول ها در RAM با استفاده از تنها 13 بیت محاسبه می شود. مقادیر SensorScheme در 15 بیت بیان می شوند، که 2 بیت پایین به عنوان نوع برچسب ها برای چهار نوع داده های SensorScheme موجود است: نمادها، شماره های کوتاه، شماره های طولانی وcons cell ها. 2 بیت دیگر در هر cons cell برای تخصیص حافظه و جمع آوری زباله استفاده می شود که یک علامت ساده و جمع کننده رفت و آمد شبیه الگوریتم اصلی Deutsch-Schorr-Waite است [15].

جدول 1 جزئیات استفاده از حافظه را نشان می دهد. محیط زمان اجرای SensorScheme، از جمله اجرای روش های ابتدایی، بسیار کوچک است و تنها KB 7.7 از حافظه برنامه را استفاده می کند. بیشتر 10 کیلوبایت RAM برای مخزن مشترک در دسترس است. بقیه به سیستم عامل و بافر شبکه اختصاص داده شده است.

6.2 اندازه کد و استفاده از حافظه

قبل از اینکه ما در مورد ارزیابی های انجام شده بحث کنیم، ابتدا اندازه کدهای برنامه را که در جدول 2 نشان داده شده است، در نظر می گیریم. برای فعال کردن اجرای برنامه که در شکل 4 ارائه شده است، برخی از توابع استاندارد کتابخانه نیز مثل every و assoc در گره ها دسترس هستند.

جدول 2 نشان می دهد که کد کتابخانه کمی بزرگتر از خود برنامه است. در مقایسه با کد منبع، کدگذاری شبکه فشرده، در طول انتقال در شبکه،به کمتر از یک پنجم را کاهش می دهد. در حافظه، اندازه کد برنامه بزرگتر است، زیرا در سلولهای حافظه قرار دارد و مجموعا 1500 (375 × 4) بایت مصرف می کند. این 1975 سلول دیگر را برای کد برنامه اضافی و برای استفاده در حین اجرای برنامه، با پشته تماس، متغیرهای سراسری و محلی، صف بندی های برنامه ریزی و تایمر و داده های برنامه کاربردی در دسترس قرار میدهد.

به ویژه در هنگام انتقال کد برنامه، SensorScheme یک نمایش بسیار فشرده را تولید می کند که برنامه ریزی سریع و کارآمد را امکان پذیر می سازد.

SensorScheme runtime	7750	bytes Flash
– garbage collector	294	bytes
– cell allocator	122	bytes
– (un)marshaller	1640	bytes
– primitives	3728	bytes
MSP430 memory	10240	bytes RAM
OS and buffers	830	bytes
runtime state	10	bytes
memory pool (2350 cells)	9400	bytes

جدول 1 استفاده حافظه ی اجرای SensorScheme

Code size	program	library	all	
Source code	963	1032	1991	chars
Net-encoded	176	186	362	bytes
In memory	181	194	375	cells
Available			1975	cells

جدول 2. اندازه های کد برنامه نمونه

6.3 عملکرد زمان اجرا و استفاده از انرژی

ما استفاده از حافظه و تاثیر هزینه های کلی ارزیابی و جمع آوری مواد زاید را در کل زمان محاسبات اندازه گیری کرده ایم که در سیستم عامل های WSN کم هستند. استفاده از انرژی یک عامل عملکرد بسیار مهم است، بنابراین ما انرژی مورد استفاده توسط اجرای برنامه های SensorScheme را اندازه گیری می کنیم.

برای اندازه گیری زمان محاسباتی از یک شبیه ساز پردازنده در یک شبکه شبیه سازی شده از 20 گره استفاده کردیم که هر کدام به صورت دوره ای از تابع peer-verify شکل 4 استفاده می کنند. این نشان دهنده یک وضعیت دنیای واقعی است؛ زیرا در یک زمان معین،تنها یک تایید باید صورت گیرد. تمام محاسبات انرژی براساس برگه های اطلاعات اجزای سخت افزاری پلت فرم پیاده سازی ما است.

جدول 3 (a) لیستی از نتایج زمان اجرا را در هر فراخوانی تابع peer-verify را نشان می دهد. برای هر یک از این دوره ها، کد SensorScheme تنها 208 میلی ثانیه زمان اجرا لازم دارد. با یک دوره زمانی 10 ثانیه (حداقل با دو بار تکرار 5 ثانیه) این فقط دو درصد زمان مصرفی CPU است.

کسر بزرگ (حدود 57٪) زمان اجرا برای تخصیص حافظه و جمع آوری زباله صرف می شود. این یک نتیجه منطقی از طراحی SensorScheme است. داده های برنامه و همچنین چارچوب پشته از مخزن حافظه اختصاص داده می شوند. این به سرعت تمام حافظه را مصرف می کند و پس از آن یک چرخه جمع آوری زباله ضروری است. خود جمع آوری زباله باعث توقف موقت برنامه ها به مدت تنها 10 ms می شود، که تاخیر قابل قبولی برای اکثر برنامه های WSN است.

در انتهای هر جمع آوری زباله میانگین تعداد سلول های استفاده شده 395 و حداکثر 429 است. با در نظر گرفتن 375 سلول مورد استفاده برای ذخیره برنامه، بین 20 و 54 سلول برای داده های برنامه و سازه های زمان اجرا اشغال می شود. در کل، کمتر از یک پنجم کل حافظه مورد نیاز این نرم افزار است، و فضای زیادی برای دیگر برنامه های بزرگتر یا پیچیده تر وباق میگذارد.

ارتباطات بخش قابل توجهی از مصرف انرژی در گره WSN را به دست می آورد. جدول 3 (b) تعداد پیام های فرستاده شده و دریافت شده در هر دوره را نشان می دهد و انرژی مصرف شده برای محاسبات اضافی توسط سیستم عامل، براساس برآوردها و مصرف انرژی در هنگام ارسال و دریافت،را نشان میدهد. (قبل از ارسال، نیاز رادیو به وصل نیرو ، 3 ms زمان اضافی صرف میکند، که در زمان پخش نیز موجود است.)

در نهایت، از این سه منبع انرژی با یکدیگر، جدول 3 (c) هزینه نسبی هر یک از آنها را نشان می دهد. این نشان می دهد که اکثر انرژی توسط نیروی رادیویی در طول ارتباطات (89٪) استفاده می شود، در حالی که زمان محاسبات تنها 7٪ از کل مصرف انرژی را صرف می کند. ما منابع دیگری از انرژی مصرفی مانند پروتکل MAC (گوش دادن بیکار) و خواندن سنسورها را که فقط کسر انرژی مصرف شده توسط تفسیر برنامه را کاهش می دهد، حساب نمیکنیم. در نتیجه، تأثیر هزینه های کلی تفسیر بر کل بودجه انرژی حداقل است و بیش از 7 درصد آن نخواهد بود.

(a)

	cycles	ms	mJ
Execution time and energy	1245483	208	1.27
Fraction spent in allocation	25.2%		
Fraction spent in GC	31.4%		
– # collections	6.43		
– execution time / collection	10.1	ms	
– avg. used cells	395	cells	
– max. used cells	429	cells	

(b)

Comm. energy	TX	RX	total	
peer-dict-hdl	2	12.6		msgs
peer-dict-rpl	12.4	107		msgs
OS time	41.4	88.9	130	ms
OS energy	0.25	0.54	0.80	mJ
message size	160	153		bits / msg
air time	89	365	455	ms
radio energy	2.41	14.04	16.45	mJ

(c)

Total energy used			
program execution	1.27	mJ	7%
OS communication	0.80	mJ	4%
radio TX / RX	16.45	mJ	89%
Total	18.52	mJ	

جدول 3: استفاده از زمان و انرژی ارزیابی برنامه نمونه

7. نتیجه گیری و جهت آینده

این مقاله مدیریت زنجیره تامین را به عنوان یک منطقه کاربردی دیگر از شبکه های حسگر بی سیم معرفی می کند. ما امکانات تجهیزات ردیابی حمل و نقل فعال را مشخص کرده ایم که می توانند کارایی و کیفیت خدمات را برای صنعت حمل و نقل افزایش دهد. علاوه بر این، ما مورد استفاده معمولی برای ATTD ها را مورد بحث قرار دادیم و تحلیل کردیم چگونه می توان با استفاده از تکنولوژی WSN فعلی آن را اجرا کرد. نرم افزار سیستم WSN فعلی دارای نیازهای منابع بالایی است و یا قابلیت های بسیار کمی را برای برآورده ساختن نیازهای نرم افزار فراهم می آورد، که عمدتا به دلیل سیستم عامل های WSN است که از حافظه رها شده اند.

SensorScheme پلت فرم پیشنهادی ما برای پیاده سازی قابلیت ردیابی فعال حمل و نقل در سخت افزار WSN است. بر اساس معانی شناسی زبان برنامه نویسی Scheme، مزایای زبان های سطح بالا مانند جمع آوری زباله، همزمان سازی و مرتب سازی خودکار پیام ها را به ارمغان می آورد، که منجر به سایز کوچکتر برنامه می شود. با این حال، اجرای SensorScheme کوچک و کارآمد است. با استفاده بهتر از حافظه کم در دسترس ، SensorScheme

می تواند طیف گسترده ای از قابلیت ها را فراهم کند. SensorScheme به دلیل تفسیر برنامه و جمع آوری زباله باعث صرفه جویی بیشتر انرژی اضافی می شود و تاخیر قابل ملاحظه ایبه دنبال ندارد.

با این حال، چالش های تحقیقاتی باقی می مانند. امنیت بیشتر، مانند عملیات ضد جعل و محرمانه بودن اطلاعات ورود به سیستم، باید راه خود را به پلت فرم SensorSchemeپیدا کنند. علاوه بر این، تخصیص حافظه پویا باعث درجه ای از غیر قابل پیش بینی بودن می شود،که احتمالا باعث کست گره ها در مواقع دلخواه می شود زمانی که حافظه آزاد باقی نمی ماند. روش هایی برای کاهش این مسئله به میزان قابل توجهی قابلیت استفاده از پلت فرم را افزایش می دهد.