دانشگاه صنعتی اصفهان

دانشکده مهندسی برق و کامپیوتر

پلتفرم زمان بندی مصرف انرژی الکتریکی کاربران خانگی

گزارش پروژه ی کارشناسی

محمد مهدی امینی

استاد

دکتر منشئی

بهمن ۱۳۹۶

فهرست مطالب

عنوان **صفحه**

فهرست مطالب . . . . . دو
فهرست تصاویر . . . . . چهار
چکیده ۱

فصل اول : مقدمه **۲**
۱ـ۱ پروتکل MQTT ۲
۱ـ۱ـ۱ لایه ی انتقال ۳
۱ـ۱ـ۲ ساختار پیام ۳
۱ـ۱ـ۳ الگوی Publish/Subscribe ۳
۱ـ۱ـ۴ جلسه ی MQTT ۵
۱ـ۱ـ۵ امنیت ۶
۱ـ۲ رمزنگاری Homomorphic ۶
۱ـ۲ـ۱ کاربردها ۷
۱ـ۳ الگوریتم رمزنگاری Paillier ۸
۱ـ۴ معماری Microservice ۸
۱ـ۴ـ۱ Microservice چیست؟ ۸
۱ـ۵ معایب یا پیچیدگی ها ۱۱
۱ـ۶ OAuth2 . . . . . ۱۱
۱ـ۷ استاندارد OAuth2 . . . . . ۱۲
۱ـ۸ شمای فرایند اجرای پروتکل ۱۲

فصل دوم : مساله ی بهینه سازی مصرف انرژی الکتریکی کاربردهای خانگی **۱۴**
۲ـ۱ تعریف مساله ۱۴
۲ـ۲ اهمیت زمان ۱۵
۲ـ۳ بهینه سازی انرژی مصرفی و هزینه مصرف ۱۶
۲ـ۴ نیازمند های امنیتی ۱۷
۲ـ۴ـ۱ حریم شخصی و محرمانگی ۱۷
۲ـ۴ـ۲ احراز هویت ۱۷

فصل سوم : روش پیشنهادی برای بهینه سازی مصرف انرژی الکتریکی کاربردهای خانگی ۱۸
۳ـ۱ راه حل پیشنهادی ۲۰
۳ـ۱ـ۱ جمع آوری اطلاعات ۲۰
۳ـ۱ـ۲ موجودیت ها ۲۰
۳ـ۲ پلتفرم پیاده سازی شده ۲۲
۳ـ۲ـ۱ موجودیت های مفهومی پلتفرم ۲۴
۳ـ۲ـ۲ قطعات نرم افزاری معماری و وظایف آن ها ۲۴
۳ـ۳ سناریوی کامل استفاده از پلتفرم ۲۵
۳ـ۴ الگوریتم بهینه سازی ۲۹
۳ـ۵ نو آوری ها ۳۰
۳ـ۵ـ۱ استفاده از الگوریتم رمزنگاری Paillier برای حفظ حریم شخصی ۳۰
۳ـ۵ـ۲ طراحی و پیاده سازی یک روش احراز هویت بر پایه ی Token برای پروتکل MQTT ۳۱
۳ـ۶ وضعیت آزمایشی ۳۲

فصل چهارم : نتیجه گیری ۳۳

فصل پنجم : ضمیمه ۳۵
۵ـ۱ کد منبع ۳۵
۵ـ۲ توضیح پروتکل CoAP ۳۵
۵ـ۲ـ۱ مقایسه با HTTP ۳۶
۵ـ۲ـ۲ پیام ها ۳۷
۵ـ۲ـ۳ ساختار پیام ها ۳۸
۵ـ۲ـ۴ امنیت ۳۹
۵ـ۲ـ۵ اینترنت اشیاء ۳۹

فهرست تصاویر

۱ـ۱ شمای Header با طول ثابت ۳
۱ـ۲ نمای کلی از مدل Pub/Sub در قالب یک مثال از MQTT ۴
۱ـ۳ شمای پیامی که در MQTT منجر به Publish می شود ۵
۱ـ۴ شمای پیامی که در MQTT منجر به Subscribe می شود ۵
۱ـ۵ مثال استفاده از jPaillier ۸
۱ـ۶ فرایند اجرای پروتکل OAuth2 . . . . . ۱۳

۳ـ۱ موجودیت های دخیل در مساله با نگاه سطح بالا ۲۱
۳ـ۲ قطعات نرم افزاری پلتفرم ۲۳
۳ـ۳ لیست دستگاه ها در Application موبایل ۲۷
۳ـ۴ صفحه ی نمایش اطلاعات دستگاه اتو ۲۸
۳ـ۵ صفحه ی وارد کردن درخواست مصرف ۲۹

۵ـ۱ لایه های Coap به صورت انتزاعی ۳۷
۵ـ۲ ارسال پیام به صورت Reliable ۳۷
۵ـ۳ مدل Separate-Request/Respsone ۳۸
۵ـ۴ مدل Non-Comfirmable-Request/Respsone ۳۸
۵ـ۵ ساختار پیام های CoAP ۳۹

چکیده

در این پروژه به پیاده سازی یک روش پیشنهادی برای بهینه سازی زمان بندی مصرف انرژی الکتریکی کاربران خانگی، در قالب یک بسته ی نرم افزاری (که شامل سرویس ابری، نرم افزار تلفن همراه، نرم افزار ارتباط با دستگاه های مصرف کننده ی انرژی و نرم افزار بهینه سازی زمان بندی مصرف است)، پرداخته ایم.

طبق [۱، ۲، ۳] هزینه ی مصرف برق در ساعات مختلف روز متفاوت است و در ساعات اوج مصرف بالاترین مقدار را دارد. همچنین در ساعات اوج مصرف فشاری که به شبکه ی توزیع برق وارد می شود، افزایش می یابد. از این رو اگر کاربران دستگاه های الکتریکی ای را که در زمان استفاده از آن ها انعطاف دارند، در ساعات مختلف روز استفاده کنند (به طوری که روند مصرف کاربران در طول روز به صورت یکنواخت باشد)، هم هزینه ی مصرف کاربران کاهش می یابد هم شبکه ی توزیع برق فشار کمتری متحمل می شود. در این پروژه پیاده سازی روش پیشنهادی در [۱، ۲] برای بهینه سازی زمان بندی مصرف، با اعمال اندکی تغییر و نو آوری انجام شده است.

در هر خانه تعدادی دستگاه الکتریکی وجود دارد که کاربران در زمان استفاده از آن ها انعطاف دارند (مثل اتو یا ماشین ظرف شویی).برای اجرای الگوریتم بهینه سازی زمان بندی کاربران باید زمان هایی را مشخص کنند که قصد داردند از این دستگاه ها استفاده کنند.(و یا زمان هایی که برای استفاده از این دستگاه مانعی ندارند).این داده ها توسط کاربران به کمک نرم افزار تلفن همراه و یا توسط خود دستگاه ها اعلام می شوند. الگوریتم در هر روز توسط یک کامپیوتر کوچک که در خانه های کاربران قرار دارد، اجرا می شود و برای هماهنگی با بقیه ی خانه ها داده هایی را به صورت رمزنگاری شده با سرویس ابری مبادله می کند. در نهایت به ازای هر دستگاه یک بردار شامل زمان بندی مناسب برای استفاده از آن، تولید می شود.

در پرتوکل MQTT که یکی از پرکاربرد ترین پرتوکل های اینترنت اشیاء است، به طور پیش فرض احراز هویت بر اساس ارسال Username و Password به صورت آشکار در بسته های مبادله ای است. این راهکار از نظر امنیتی قابل قبول نیست و همچنین در سناریو های اینترنت اشیاء معمولا دستگاهایی از این پروتوکل استفاده می کنند که مستقیما در اختیار کاربر نیستند اما باید از جانب کاربر پیام ارسال کنند.از طرفی کاربران مایل به ذخیره اطلاعات کاربری خود در این دستگاه ها نیستند. در این پروژه برای رفع این مشکل یک روش احراز هویت بر اساس Token برای پروتوکل MQTT طراحی و پیاده سازی شد.

یکی از داده هایی که الگوریتم بهینه سازی به آن نیاز دارد، مجموع میزان مصرف درخواستی همه کاربران است. اگر کابران مجبور باشند، میزان انرژی مصرفی مورد نظر خود را افشا کنند، حریم شخصی آن ها به خطر می افتد. از این رو در این پروژه با استفاده از قابلیت های Homomorphic الگوریتم رمز نگاری Paillier بستری ایجاد کردیم که کاربران بدون نگرانی نبست به حریم شخصی خود، مجموع انرژی مصرفی درخواستی خود را در اختیار دیگر کاربران بگذارند.

واژه‌های کلیدی: ۱ـحریم شخصی، ۲ـ Homomorphic ۳ـ MQTT ۴ـبهینه سازی مصرف انرژی

فصل اول

مقدمه

در این فصل به معرفی تکنولوژی ها و مفاهیم علمی ای که در صورت مساله و طراحی و پیاده سازی راه حل این پروژه دخیل بوده است، می پردازیم. درباره ی هر تکنولوژی یا مفهوم توضیح مجزا و مستقل داده شده است و سعی شده فقط جزییاتی که برای درک این پروژه لازم است، بیان شود. اگر با پروتکل پیام رسانی MQTT ، الگوریتم های رمزنگاری Homomorphic ، الگوریتم رمزنگاری Paillier ، پروتکل احراز هویت OAuth2 و معماری Microservice آشنایی دارید، می توانید این فصل را مطالعه نکنید.

۱ـ۱ پروتکل MQTT

MQTT یک پروتکل استاندارد (ISO/IEC PRF 20922) ارتباطی است که به صورت اختصاصی برای کاربردهای اینترنت اشیاء طراحی شده است. این پروتکل به دلیل طراحی خاص خود در محیط های با پهنای باند محدود به خوبی کار می کند. از ویژگی های اصلی آن میتوان به سایز کوچک بسته ها و توان مصرفی پایین اشاره کرد. این پروتکل همچنین برای محیط های Wireless با ارتباط های غیر قابل اتکا و دارای تاخیر متغیر مناسب است. به علاوه، این پروتکل به شکلی طراحی شده تا پیاده سازی آن شامل پیچیدگی نباشد. کاربردهای این پروتکل اساسا به اینترنت اشیاء مربوط می شود. معروف ترین کاربرد آن ارسال داده های تولید شده توسط سنسورها به

یک Server جمع آوری مقادیر است.

۱ـ۱ـ۱ لایه ی انتقال

درلایه ی انتقال MQTT از پروتکل TCP استفاده می کند. البته یک پروتکل به نام MQTT-SN از این پروتکل مشتق شده است که امکانات MQTT را در محیط های فاقد TCP/IP مثل ZigBee ارایه می کند. اگرچه در Quality of service, TCP وجود ندارد، اما در MQTT علی رغم سادگی اش، Quality of service به عنوان یک ویژگی قابل اتکا در لایه ی Application طراحی شده است.

۱ـ۱ـ۲ ساختار پیام

پیام های MQTT به صورت Binary منتقل می شوند. هر بسته ی پیام MQTT از سه بخش تشکیل شده. بخش اول Header با طول ثابت است که ۲ byte طول دارد. بخش دوم Header اختیاری است با طول متغییر. و بخش سوم Payload است که می تواند تا ۲۵۶ مگابایت داده را شامل شود. فقط بخش اول است که در تمام انواع مختلف پیام های این پروتکل وجود دارد.

bit	7	6	5	4	3	2	1	0
byte 1	Message Type				DUP flag	QoS level		RETAIN
byte 2	Remaining Length							

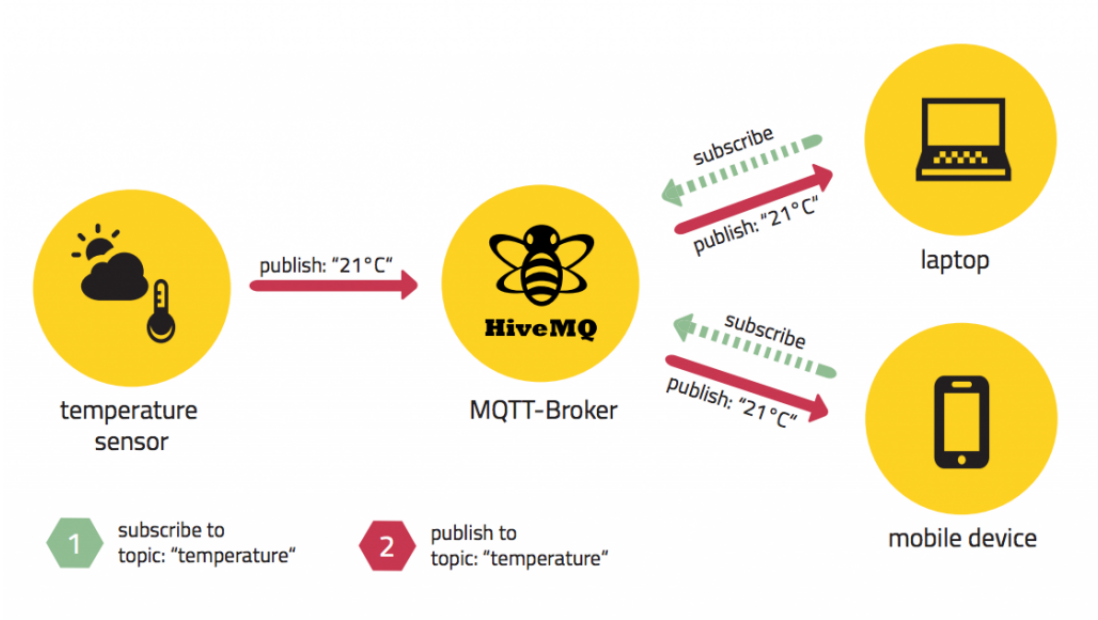
شکل ۱ـ۱: شمای Header با طول ثابت

۱ـ۱ـ۳ الگوی Publish/Subscribe

الگوی Publish/Subscribe یک روش ارتباطی جایگزین برای مدل سنتی Client/Server است. در روش Client/Server، Client به صورت مستقیم با یک نقطه ی نهایی(End-Point) که در واقع Server است ارتباط بر قرار می کند. اما در مدل Pub/Sub طرفین ارتباط از وجود یکدیگر خبر ندارند. بعضی از موجودیت ها ارسال کننده ی پیام (Publisher) و بعضی دیگر دریافت کننده (Subscriber) هستند. یک موجودیت در این مدل، می تواند به صورت همزمان هم دریافت کننده و هم ارسال کننده ی پیام باشد. در معماری این مدل یک موجودیت سوم نیز وجود دارد که هم Publisher ها و هم Subscriber ها از وجود آن آگاه هستند و با آن ارتباط مستقیم بر قرار می کنند. این موجودیت Broker نام دارد. Broker پیام ها را از Publisher ها دریافت می کند. Subscriber ها هر زمان که به Broker متصل شدند، اگر پیامی برایشان در دسترس بود، پیام را دریافت می کنند. در واقع Broker وظیفه ی مدیریت و توزیع پیام ها را دارد و به نوعی شبیه یک مجموعه ی بزرگ از Buffer ها است.

در این الگو دریافت کننده های پیام نیاز ندارند نسبت به دانستن زمانی که اطلاعات جدیدی در دسترس قرار می گیرد نگران باشند. هر زمان که پیام (یا داده ی جدیدی) در دسترس باشد با کمترین میزان سربار از آن مطلع خواهند شد. در مدل سنتی Client Client/Server، ها مجبور بودند برای بروز بودن و آگاهی از آخرین داده ها، هر چند وقت یکبار مثلا هر ده ثانیه یکبار به Server یک درخواست ارسال کنند و درباره ی وجود پیام (یا داده ی جدید) سوال کنند که باعث به Header رفتن منابع مختلف Server و Client ها در میزان بالا می شد.

با توجه به مستقل بودن Entity ها و مستقل بودن فرایند دریافت پیام از فرایند ارسال آن، امکان استفاده از مزایای موازی سازی کاملا وجود دارد. با در نظر گرفتن Event-driven بودن معماری می توان نتیجه گرفت که این مدل دارای قابلیت Scalability به میزان قابل توجهی است.



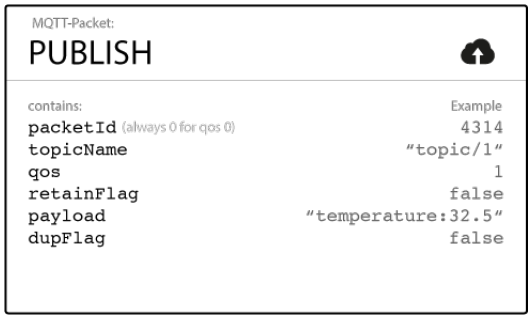
شکل ۱ـ۲: نمای کلی از مدل Pub/Sub در قالب یک مثال از MQTT

Client

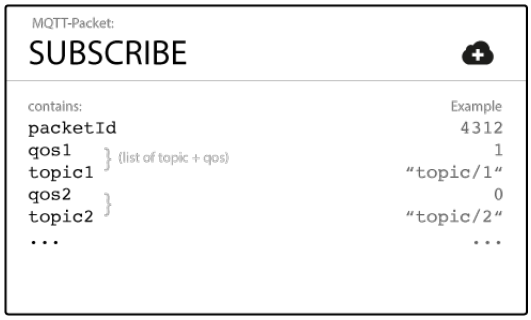
در فضای این پروتکل هر موجودیتی که توانایی Publish یا Subscribe از طریق اتصال به یک Server مرکزی (Broker) را داشته باشد، Client نامیده می شود. لازم به ذکر است که هر دو موجودیت Client و Server توانایی Publish و Subscribe کردن دارند. Client ها به ۲ دسته ی Transient و Persistent تقسیم می شوند. Client های Persistent جلسه ی ارتباطی خود با Server مرکزی (Broker) را حفظ می کنند. اما ارتباط Client های Transient توسط Server مرکزی دنبال نمی شود(Session یا جلسه ای برای Client نگه داری نمی شود.).

Topic

یک نقطه ی اتصال نهایی است که Client ها به آن متصل می شوند. Topic در واقع به عنوان یک Hub مرکزی برای توزیع پیام ها عمل می کند. اگر چه باید Topic ها قبل از اتصال موجودیت ها به آنها ساخته شوند ولی در صورت عدم وجودشان، به محض اینکه یک موجودیت درخواست اتصال به آن را ارسال کند، بدون وقفه Topic مورد نظر ساخته می شود. Topic ها با آدرسشان که یک ساختار درختی مثل ساختار آدرس فایلها دارد، شناخته می شوند. برای مثال `building1/room1/temperature` آدرسی است که تاپیک مربوط به مدیریت پیام های سنسورهای دماسنج اتاق۱ در ساختمان Building را مشخص می کند. دریافت پیام از یک Topic که در واقع Subscirbe کردن آن است به کمک آدرس آن انجام می شود. برای مثال `building1/#/` آدرس Subscirbe کردن تمام Topic های زیر شاخه ی `Building1` است.



شکل ۱ـ۳: شمای پیامی که در MQTT منجر به Publish می شود



شکل ۱ـ۴: شمای پیامی که در MQTT منجر به Subscribe می شود

۱ـ۱ـ۴ جلسه ی MQTT

هر جلسه ی MQTT از چهار فاز Connection, Authentication, Communication, Termination تشکیل می شود. Client ابتدا با ایجاد یک ارتباط با Broker شروع می کند(فاز Connection). ممکن است Broker جلسه ی جدیدی برای Client ایجاد نکند و آخرین جلسه ای که Client با آن به Broker متصل بوده را ادامه دهد. اتصال می تواند از طریق پورت های استاندارد ۱۸۸۳ برای ارتباط های عادی و ۸۸۸۳

برای ارتباط های SSL/TLS انجام گیرد. همچنین امکان تنظیم یک پورت دلخواه در بروکر برای ایجاد ارتباط وجود دارد.

سپس ،Client با بررسی گواهینامه ی Server به اهراز هویت آن می پردازد. همچنین این امکان به صورت اختیاری برای Client وجود دارد که گواهینامه ی خود را به Server ارایه کند(فاز Authentication).

پس از اهراز هویت ،Server امکان Publish پیام به یک topic خاص یا Subscribe کردن پیام های آن وجود دارد(فاز Communication).

Server و Client ها می توانند به ارتباط TCP ای که با یکدیگر دارند، خاتمه دهند(فاز Terminition).

۱ـ۱ـ۵ امنیت

با توجه به هدف طراحی پروتکل MQTT ساده و کم سربار بودن است، وجود برخی صعف های امنیتی در آن، دور از انتظار نیست. این پروتکل معمولا برای ارسال داده های سنسورها و ارسال دستور به عملگرها استفاده می شود، در نتیجه احراز هویت و محرمانگی از نیازهای مشترک اکثر کاربردهای آن است. احراز هویت در این پروتکل به ۲ روش قابل دستیابی است.

۱ـ ارسال Username و Password به صورت clear-text . در این روش Client اطلاعات کاربری خود را در پیام های MQTT به صورت شفاف قرار می دهد. این روش بسیار ابتدایی و به راحتی قابل دور زدن است. در نتیجه استفاده از آن پیشنهاد نمی شود.

۲ـ استفاده از پروتکل SSL/TLS . در این روش Client موظف است گواهینامه ی Server را بررسی کند و ارایه ی گواهینامه توسط Client به صورت اختیاری امکان پذیر است. متاسفانه استفاده از این روش باعث اعمال پیچیدگی و سرباری می شود که با هدف ساده بودن MQTT در تضاد است.

۱ـ۲ رمزنگاری Homomorphic

در یک دسته بندی، سیستم های رمز نگاری به ۲ دسته ی Homomorphic و غیر Homomorphic تقسیم می شوند. اگر یک سیستم رمزنگاری، در واقع یک الگوریتم رمزنگاری (چه متقارن چه نامتقارن) Homo-morphic باشد، (به طور خلاصه می توان گفت) امکان انجام عملیات ریاضی روی داده های رمزنگاری شده را دارد. به طور دقیق تر، مثلا چند عدد داریم و آن ها را توسط یک الگوریتم رمزنگاری Homomorphic به صورت رمز در می آوریم. اکنون می توان این اعداد را همانطور که در حالت رمزنگاری شده هستند، با هم جمع کرد. حاصل جمع بدست آمده هنوز یک عدد رمزنگاری شده است. اگر این حاصل را رمزگشایی کنیم، نتیجه

برابر با حاصل جمع اعداد اولیه خواهد بود.
از این الگوریتم ها زمانی استفاده می شود که نیاز باشد عملیاتی روی داده ها انجام شود، بدون که انجام دهنده ی عملیات از داده ها و نتیجه ی عملیات آگاهی پیدا کند.
از معروف ترین الگوریتم های رمزنگاری Homomorphic می توان به RSA ، ElGamal و Paillier اشاره کرد.

عملیات های مختلفی مثل ضرب و مرتب سازی و حتی عملیات های سفارشی، را می توان توسط الگوریتم های Homomorphic مختلف روی داده های رمزنگاری شده انجام داد. بعضی از این الگوریتم ها Partially Homomorphic هستند. یعنی فقط توانای محاسبه ی نتیجه ی یک عملیات خاص مثل جمع را دارند. بعضی دیگر Fully Homomorphic هستند. یعنی توانایی محاسبه ی نتیجه ی بیش از یک نوع عملیات را دارند (مثلا ضرب و جمع).
از ترکیب الگوریتم های Homomorphic مختلف با توانایی انجام عملیات های مختلف می توان یک کامپیوتر سطح بالا ساخت که داده های رمزنگاری شده را به عنوان ورودی دریافت کند و مجموعه ای از عملیات ها را روی آن ها انجام دهد.

۱ـ۲ـ۱ کاربردها

تقریبا در هر موقعیتی که نیاز به انجام عملیات محاسباتی روی داده هایی باشد که باید بر اساس ضرورت هایی مثل حریم شخصی محرمانه بمانند،می توان از الگوریتم های رمزنگاری Homomorphic استفاده کرد. پر رنگ ترین مثال درباره ی کاربرد این الگوریتم ها، فرایند رای گیری است که رای هر رای دهنده باید برای شمارنده های رای مخفی بماند و در عین حال نتیجه ی رای گری توسط عملیاتی شبیه جمع مشخص شود.

معمولا شرکت هایی که با حجم زیادی از داده ها و محاسبات سر و کار دارند، از سرویس های ابری برای ذخیره ی داده هایشان و انجام عملیات های مختلف روی آن ها استفاده می کنند. با توجه به این که این داده ها گاها باید محرمانه بمانند، سرویس دهنده ی ابری می تواند روی نسخه ی رمزنگاری شده ی داده ها بدون نیاز به دسترسی به نسخه ی اصلی آن ها، عملیات های مورد نظر را انجام دهد و نتیجه را نیز به صورت رمزنگاری شده برای سرویس گیرنده ارسال کند.

۱ـ۳ الگوریتم رمزنگاری Paillier

الگوریتم رمزنگاری نامتقارن Paillier یک الگوریتم Partially Homomorphic است که امکان اجرای عملیات جمع روی داده های رمز نگاری شده را می دهد. این الگوریتم در سال ۱۹۹۹ ابداع شده است. در این الگوریتم برای محاسبه ی حاصل جمع دو عدد، باید حاصل ضرب رمزشده ی آن ها را حساب کرد و نتیجه را رمزگشایی کرد.

همچنین اگر یک عدد مثل A را توسط این الگوریتم به رمز E(A) تبدیل کنیم و حاصل را به توان عدد K برسانیم و سپس رمزگشایی کنیم، A*K را محاسبه کرده ایم. در واقع می توان حاصل ضرب یک عدد رمزنگاری شده در یک عدد مشخص را به کمک این الگوریتم حساب کرد.

کد موجود در تصویر زیر یک مثال کوتاه درباره ی استفاده از این الگوریتم در زبان برنامه نویسی جاوا به واسطه ی کتابخانه ی jPaillier است.

```
BigInteger plainA = BigInteger.valueOf(102);
BigInteger plainB = BigInteger.valueOf(203);

BigInteger encryptedA = publicKey.encrypt(plainA);
BigInteger encryptedB = publicKey.encrypt(plainB);

BigInteger encryptedProduct = encryptedA.multiply(encryptedB).mod(publicKey.getnSquared());

BigInteger additionResult = keypair.decrypt(encryptedProduct);

// additionResult = 102 + 203 = 305
```

شکل ۱ـ۵: مثال استفاده از jPaillier

۱ـ۴ معماری Microservice

چگونه یک سیستم نرم افزاری در اندازه ی بزرگ بسازیم که به راحتی قابل تغییر باشد در عین حال نگه داری از آن آسان باشد؟ آن را به زیر سیستم های کوچک تر تقسیم کنیم.

۱ـ۴ـ۱ Microservice چیست؟

Microsevice یک معماری نرم افزار است که برای ساخت سیستم های نرم افزاری با اندازه بزرگ مناسب است. در این معماری یک نرم افزار که عملیات های مختلف و use-case های مختلف را پشتیبانی می کند به قطعات نرم افزاری کوچک تر تقسیم می شود و هر قطعه به طور کاملا مجزا و مستقل از قطعات دیگر طراحی، پیاده سازی و اجرا می شود. قطعا این قطعات برای انجام فعالیت های مورد نظر خود به برقراری ارتباط با دیگر قطعات نیاز دارند که این نیاز از طرق مختلفی مثل استفاده از WebService ها، EventBus و یا Messaging بر طرف می شود. این قطعات به طور مستقل Desing ، Develop ، Complie ، Test ،

Deploy و Monitor می شوند. در نتیجه هزینه های این فعالیت ها به دلیل کاهش پیچیدگی کم می شود. سرعت Release شدن نرم افزار نیز افزایش پیدا می کند. چون تست این قطعات کوچک تر اسان تر است، در مجموع امنیت سیستم نرم افزاری نیز افزایش پیدا می کند. همچنین این معماری به خوبی با متودولوژی Agile قابل تطابق است. برای مثال می توان به راحتی Story ها را به دلیل قطعه قطعه بودن نرم افزار تعریف کرد و دلیل سرعت بالای تولید نتایج تست، مطالب مناسبی برای توضیح در جلسه های روزانه در دسترس است.

این معماری مناسب بکارگیری در سیستم های نرم افزاری ای است که اندازه ی آن ها بزرگ است. نکته ی قابل توجه این است که اندازه یک ویژگی نسبی است و مقادیر آن مثل بزرگ، متوسط و کوچک در شرایط مختلف معنای متفاوتی دارند. منظور از سیستم نرم افزاری با اندازه ی بزرگ، سیستمی است که با گذشت زمان از شروع فعالیت نیاز به بزرگ شدن پیدا می کند و با تغییرات در نیازمندی ها مواجه می شود. به گونه ای که به سختی می توان محدوده ای برای پیشرفت و بلوغ آن در نظر گرفت. از سویی دیگر می توان بزرگ بودن را به این صورت تفسیر کرد که در سیستم نرم افزاری انواع مختلف ارتباطات بین موجودیت های مختلف لازم است. مثلا سیستم لازم دارد که از طریق رابط کاربری تحت وب و رابط کاربری موبایل با Client هایش ارتباط بر قرار کند. برای ذخیره سازی داده های متفاوت از پایگاه داده های متفاوت استفاده می کند. می توان احتمال حظور قطعات سخت افزاری تحت کنترل سیستم را نیز در نظر گرفت (IOT) و ... .

از معروف ترین شرکت هایی که این معماری را به کار گرفته اند می توان به NetFlix و Amazon اشاره کرد. استفاده از این معماری باعث می شود هزینه های تولید نرم افزار مدیریت شده تر شوند و سرعت پیشرفت و به طبع درآمد زایی نرم افزار نیز بیشتر شود. با به کار گیری این معماری قطعات نرم افزاری تشکیل دهنده ی نرم افزار اصلی به دلیل کوچک و مستقل بودن، سریع توسعه می یابند، سریع تست می شوند و سریع در دسترس قرار می گیرند. در صورت نیاز به تغییر در نرم افزار فقط قطعه ی نرم افزاری خاصی، تغییر می کند. در نتیجه هزینه ی تغییر پایین است و به سرعت می توان تغییرات لازم را اعمال کرد. برای ایجاد تغییر لازم نیست پیچیدگی کل سیستم را در نظر گرفت. و بازگردانی نرم افزار به حالت در دسترس سریع اتفاق می افتد چون مراحل Complie و Deploy سریع است.

اولین سوالی که به ذهن می رسد این است که منظور از قطعات کوچک تر چیست و این قطعات باید چقدر کوچک باشند. قواعدی وجود دارد که مشخص می کند چه وظایفی را به عنوان یک قطعه در نظر بگیریم. البته این قواعد در شرایط متفاوت ممکن است نتایج متفاوت داشته باشد. این شرایط می تواند نیازمندهای نرم

افزاری، فرهنگ سازمانی و یا حتی تعداد توسعه دهنده های در دسترس باشد. در ادامه توضیح کوتاهی برای هر یک از این قواعد ارایه می کنیم. این قواعد پایه ی طراحی قطعات نرم افزاری تشکیل دهنده ی کل نرم افزار هستند.

یک چیز را خوب انجام بده [۱] هر قطعه ی نرم افزاری باید روی یک وظیفه ی خاص تمرکز کند و آن وظیفه را به خوبی انجام دهد.این یک وظیفه می تواند کل فرایند حسابداری باشد و یا پیدا کردن یک رکورد خاص در پایگاه داده.

جدید بسازید [۲] برای افزودن ویژگی های جدید به نرم افزار بهتر است یک قطعه ی نرم افزاری جدید ایجاد کرد به جای اینکه یک قطعه ی موجود را تغییر داد و ویژگی مورد نظر را به آن اضافه کرد.

خروجی به ورودی [۳] خروجی یک قطعه را باید بتوان به عنوان ورودی یک قطعه ی دیگر استفاده کرد. برای این منظور می توان خروجی های API ها را در قالب های معروفی مثل JSON و HTML که در سطح وب معروف هستند، منتشر کرد.

عدم اصرار روی ورودی تعاملی [۴] در طراحی این قطعات نرم افزاری باید تا حد امکان نیاز به دخالت انسان در تولید داده های ورودی کم شود. یعنی قطعات به گونه ای طراحی شوند که ورودی های خود را مستقیم از خروجی دیگر قطعات بگیرند و انسان در این میان دخالتی نداشته باشد. رعایت این اصل باعث می شود بتوان قطعات را در موقعیت های غیرقابل انتظار بیشتری بکار برد. در واقع باید وابستگی سیستم نرم افزاری به حظور انسان در مراحل مختلف کاهش یابد.

تست زود هنگام [۵] باید در مراحل اولیه پیاده سازی و قبل از پیچیده شدن قطعات نرم افزاری، آن ها را تست کرد. بدین ترتیب توسعه دهنده زودتر feedback دریافت می کند و زود تر اشتباهات رخ داده را متوجه می شود. در نتیجه قطعه ی نرم افزاری زودتر بهبود پیدا می کند.

برای دور انداختن قطعات تامل نکنید [۶] وقتی قطعات نرم افزاری کوچک باشند، برای پیاده سازی آن ها تلاش و وقت زیادی صرف نشده است. در نتیجه در موقعیت هایی که باید یک قطعه ی جدید جایگزین آن ها شود یا به این نتیجه می رسیم که طراحی یا پیاده سازی یک قطعه کاملا غلط بوده، به راحتی می توان آن را کنار گذاشت.

---

[1] Do one thing well
[2] Build afresh
[3] Expect output to become input
[4] Don't insist on interactive input
[5] Try early
[6] Don't hesitate to throw it away

ابزار سازی [1] معمولا توسعه دهنده در فرایند مدیریت، تست، Deploy ، Scale و... قطعات نرم افزاری به پیچیدگی هایی بر می خورند که برای رفع آن ها دست به طراحی و پیاده سازی ابزار های خاصی می زنند. این احتمال وجود دارد که شرکت های مختلف ابزارهای مختص به خودشان را تولید کنند و حتی یک تیم مشخص برای تولید این ابزار ها اختصاص دهند. این ابزار هدف فرایند توسعه نرم افزار نیستند بلکه رسیدن به هدف اصلی را تصحیل می کنند.

۱ـ۵ معایب یا پیچیدگی ها

هیچ طراحی یا معماری ای کامل نیست به طوری که در همه ی مسایل کلید طلایی باشد. در معماری Mi-croservice اگرچه قطعات نرم افزاری تک وظیفه ای با پیچیدگی کم تولید می شوند؛ وقتی تعداد این قطعات بالا رفت طراحی ارتباط بینشان معمولا همراه با پیچیدگی های خاصی است. همچین تست کلی سیستم نرم افزاری نیز فرایند آسانی نیست. این معایب در برابر نقطه ی مقابل معماری Microsevice یعنی معماری Monolithic بیان می شوند.

۱ـ۶ OAuth2

در بسیاری از موقعیت ها یک سیستم نرم افزاری برای انجام وظیفه اش به دسترسی ای نیاز دارد که فقط از طریق حساب کاربری یک شخص مقدور است. برای مثال شخصی در سایت شبکه اجتماعی یک حساب کاربری دارد. شخص علاقه دارد که به کمک یک بورد اینترنت اشیاء اطلاعات سنسور دمایی که در اتاقش نصب کرده را در صفحه اش در شبکه ی اجتماعی منتشر کند(به صورت خودکار). آن بورد اینترنت اشیاء برای دسترسی به صفحه ی اجتماعی شخص به اطلاعات حساب کاربری او نیاز دارد. در واقع شخص باید اطلاعات حساب کاربری اش(نام کاربری و کلمه ی عبور) را در دستگاه ذخیره کند. این عمل از نظر امنیتی مشکل دارد. پس بورد اینترنت اشیاء چگونه اجازه انتشار اطلاعات در سایت شبکه ی اجتماعی را بدست آورد؟ شخص در سایت شبکه ی اجتماعی وارد می شود و در خواست یک Token می کند. سپس Token دریافتی را در بورد اینترنت اشیاء ذخیره می کند. این Token در یک مدت زمان قابل تایین به بورد اینترنت اشیاء اجازه می دهد که درخواست های مشخصی را برای سایت شبکه اجتماعی ارسال کند.

موقعیت های مشابه فراوانی مثل سناریویی که در بالا گفته شد وجود دارد که یک سیستم نرم افزاری ۳rd-party از طریق یک Token از جانب یک کاربر احراز هویت می شود و خدمات مشخصی را دریافت می

[1] Toolmaking

کند. یکی دیگر از کاربرد های این روش در حل مساله ی Single-SignIn است. برای مثال شما در سایت Google وارد می شوید و سپس بدون نیاز به ورود مجدد از سرویس ها مختلف آن مثل ایمیل یا فضای ذخیره سازی استفاده می کنید(در صورتی که این سیستم ها از همدیگر جدا هستند).

۱ـ۷ استاندارد OAuth2

OAuth2 یک پروتکل است که برای مشخص کردن دسترسی یک Client نرم افزاری به Resource هایی از پیش مشخص شده است. اگر چه وطیفه ی این پروتکل Authorization است ولی اگر از دید Client نرم افزاری نگاه کنیم، در واقع عمل Authentication یا احراز هویت را انجام می دهد. در فرایند این پروتکل چند موجودیت تعریف می شوند که به شرح زیر اند:

Resource می تواند یک صفحه در شبکه ی اجتماعی یا یک فرایند از پیش تایین شده برای انتقال پول از حساب End-User باشد.

Client که به App هم معروف است در واقع یک Application موبایل یا وب است که درخواست هایی برای دسترسی به Resource ها را از جانب صاحب Resource ها می دهد.

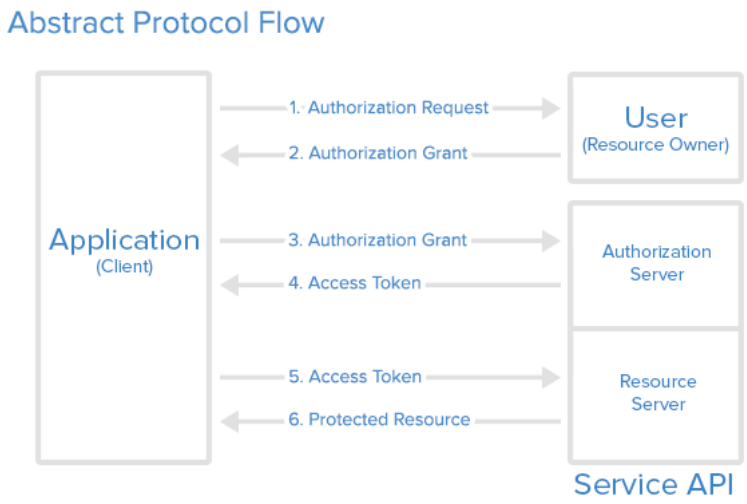
صاحب Resource ها یا End-User یک شخص است که مشخص می کند چه App هایی به چه Re-source هایی دسترسی داشته باشند.

Resource Server یک Server است که ارایه دهنده ی Resource ها است. برای مثال سایت Facebook

Authorization Server که به Server احراز هویت نیز معروف است پیاده سازی نرم افزاری هسته ی پروتکل OAuth2 است. وظیفه این Server تولید Token ها، بررسی اعتبار آن ها، بررسی دسترسی های الصاق شده به آن ها و احراز هویت End-User است.

۱ـ۸ شمای فرایند اجرای پروتکل

روش های مختلفی برای اجرای فرایند این پروتکل وجود دارد که در موقعیت های مختلفی مورد استفاده قرار می گیرند. در واقع همه ی آن ها یک روش واحد هستند که داده ی حاصل از بخش هایی از آن به صورت پیش فرض یا قبل از اجرای فرایند تعیین شده اند، در نتیجه آن بخش ها از فرایند اجرا، حذف می شوند. در زیر به توضیح فرایند کامل می پردازیم.



شکل ۱ـ۶: فرایند اجرای پروتکل OAuth2

۱ Application برای دریافت اجازه ی دسترسی به Resource ها درخواست ارسال می کند.

۲ اگر End-User اجازه را صادر کند، Application اجازه را دریافت می کند.

۳ Application اطلاعات احراز هویت خود و اجازه ای که از End-User دریافت کرده را به Server Authorization ارسال می کند و یک Access Token در خواست می کند.

۴ در صورت صحت اطلاعات، Authorization Server یک Access Token صادر می کند و برای Application ارسال می کند.

۵ Application برای دسترسی به Resource به Resource Server در خواستی همراه با Access Token ارسال می کند.

۶ Resource Server بعد از مطمعن شدن از صحت Access Token ، Resource را در اختیار Appli- cation قرار می دهد.

فصل دوم

مساله ی بهینه سازی مصرف انرژی الکتریکی کاربردهای خانگی

در این فصل به توضیح و بررسی مساله ای که حل آن هدف این پروژه بوده است، می پردازیم. سپس نیازمندی های امنیتی آن را شرح می دهیم و در فصل بعد به ارایه راه حل خواهیم پرداخت.

۲ـ۱ تعریف مساله

انرژی یکی از پایه ای ترین نیازمندی های بشر برای ادامه زندگی است؛ از این رو بشر همیشه درباره ی میزان انرژی در دسترس خود نگران است. با توجه به اینکه درصد قابل توجهی از انرژی الکتریکی که امروزه در جهان تولید می شود، از سوخت های فسیلی حاصل می شود و سوخت های فسیلی تجدید ناپذیر هستند، استفاده ی بهینه از انرژی الکتریکی (برق) یک امر ضروری است. همچنین در فرایند تولید تا تحویل برق به مصرف کننده، تجهیزات و زیرساخت های فراوانی وجود دارد که برای عمل کردن هزینه هایی دارند. مصرف بهینه ی انرژی الکتریکی باعث می شود هزینه های اضافی در این زمینه کاهش یابد. با توجه به اینکه بخشی از این هزینه ها را کابران به صورت مستقیم پرداخت می کنند، مصرف بهینه قطعا در کاهش هزینه های زندگی آن ها تاثیر خواهد داشت.

استفاده ی بهینه از انرژی الکتریکی یک مساله ی چند بعدی است و در شرایط مختلف می تواند شامل پارامترهای متفاوت و جواب های متفاوت باشد. یکی از روش های پیشنهادی برای بهینه سازی مصرف انرژی الکتریکی مدیریت مصرف کاربران خانگی است؛ زیرا درصد قابل توجهی از انرژی الکتریکی در قالب مصارف خانگی مصرف می شود. در هر خانه دستگاه های الکتریکی و الکترونیکی مختلفی وجود دارد که بعضی از آن ها مثل یخچال باید در تمام طول شبانه روز روشن(در حال مصرف انرژی) باشند و بعضی دیگر دارای انعطاف در ساعات مصرف هستند؛ یعنی کاربر یا مطمعن است که باید در یک محدوده ی زمانی خاص از آن دستگاه استفاده کند و یا می تواند طبق یک برنامه ی پیشنهادی در یک زمان پیشنهاد شده از آن دستگاه استفاده کند.

مدیریت مصرف انرژی دستگاه های خانگی با رویکرد مدیریت زمان مصرف، دستگاه هایی را که کاربران در زمان استفاده از آن ها انعطاف دارند را مورد هدف قرار داده است و سعی دارد از طریق روش های مختلف زمان مصرف این دستگاه ها را در بهینه ترین حالت ممکن قرار دهد. یکی از این روش ها کنترل دستگاه های دارای انعطاف در زمان مصرف، به صورت مرکزی است و زیرساخت لازم برای این روش به واسطه ی شبکه ی هوشمند برق و پیشرفت های حوزه ی اینترنت اشیاء فراهم می شود. در این روش شرکت توزیع کننده ی برق با توجه به اطلاعاتی که از میزان مصرف کاربران شبکه در زمان های گذشته و حال دارد، دستگاه های الکتریکی را از راه دور کنترل می کند. در این روش استقلال مصرف کننده های خانگی و حریم خصوصی آن ها رعایت نمی شود.
یک روش دیگر این است که کاربرانی که در یک منطقه یا محله ی جغرافیایی زندگی می کنند، با به اشتراک گذاری اطلاعات مصرف خود، به یک برنامه ی مصرف بهینه برسند و از دستگاه های الکتریکی منعطف خود طبق برنامه مصرف بدست آمده، استفاده کنند.

۲ـ۲ اهمیت زمان

در ساعات مختلف شبانه روز، کاربران شبکه ی توزیع برق، به صورت یکنواخت و متوازن از انرژی الکتریکی استفاده نمی کنند. یعنی مجموع مصرف کابران شبکه در ساعات مختلف شبانه روز یکسان نیست. دلیل این امر را می توان در زمان بندی فعالیت های روزانه ی افراد یافت. برای مثال اکثر افراد جامعه در ساعات اولیه روز در خانه نیستند و مشغول کارند؛ به همین سبب مجموع مصرف کاربران خانگی در ساعات اولیه روز در مقایسه با ساعات اولیه ی شب بسیار کمتر است. این تفاوت و عدم توازن در میزان مصرف در ساعات مختلف، در قالب ساعات اوج مصرف قابل بررسی است. ساعات اوج مصرف ساعاتی از شبانه روز است که در یک محدوده ی جغرافیایی خاص، در مقایسه با دیگر ساعات بیشترین میزان مصرف انجام می گیرد. ساعات اوج مصرف در

فرایند تولید و توزیع برق از اهمیت بالایی برخوردار است. در وضعیت مناسب، میزان برق در دسترس در شبکه باید از میزان درخواست مصرف در ساعات اوج مصرف بیشتر باشد. با توجه به محدودیت هایی که در تولید و توزیع برق در حوزه های جغرافیایی مختلف وجود دارد، یک راه کار برای نگه داری شبکه در وضعیت مناسب، توزیع مصرف کاربران در ساعات مختلف است؛ به طوری که مجموع مصرف در ساعات مختلف به گونه ای دستخوش تغییر شود که در بهترین حالت ساعت اوج مصرف از بین بروند و مصرف در ساعات مختلف کاملا برابر و متوازن باشد. در واقع باید تلاش شود بخش منعطف از مصرف در ساعات اوج مصرف را به ساعت دیگر منتقل کرد تا درخواست برای انرژی الکتریکی در ساعات اوج مصرف کاهش یابد.

یکی از راه کارهایی که شبکه ی توزیع برای حل این مساله در پیش گرفته، شناسایی ساعات اوج مصرف و در نظر گرفتن نرخ های بالاتری برای مصرف در ساعات اوج مصرف در مقایسه با دیگر ساعات است. از این رو اگر کاربران مصرف خود را از ساعات اوج مصرف به ساعات دیگر منتقل کنند، در نهایت هزینه های برق مصرفی آن ها نیز کاهش می یابد.

در برخی مناطق جغرافیایی و به لطف شبکه ی هوشمند توزیع برق، کاربران می توانند انرژی الکتریکی ای را که از طرقی مثل سلول های خورشیدی خانگی تولید می کنند به شبکه ی توزیع تزریق کنند و به نوعی در تولید برق کمک کنند. همچنین ممکن است روش هایی که به طور کلی به صورت پاک انرژی تولید می کنند، در شرایط آب و هوایی خاص میزان قابل توجهی انرژی تولید کنند. از نظر شبکه ی توزیع، این انرژی باید تا حد امکان مصرف شود و در شبکه باقی نماند. به این منظور ممکن است در ساعات خاص و شرایط خاص، هزینه ی مصرف کاهش یابد. در نتیجه به طور کلی به کار گرفتن یک روش که علاوه بر متوازن کردن میزان مصرف در ساعات مختلف، هزینه ی کاربران را نیز کاهش دهد، یک بهینه سازی هزینه و انرژی است که طراحی و پیاده سازی چنین بهینه سازی ای هدف این پروژه است.

۲ـ۳ بهینه سازی انرژی مصرفی و هزینه مصرف

روشی که در این پروژه برای بهینه سازی مصرف انرژی و هزینه با اقتباس از [۱، ۲، ۳] طراحی و پیاده سازی شده است، یک روش بهینه سازی سمت مصرف کننده است. یعنی مصرف کننده به کمک پلتفرم ما به گونه ای از دستگاه های الکتریکی در خانه استفاده می کند که برایند مصرف او با میزان مصرف افراد هم محله اش متوازن (نمودار صاف بجای نمودار زنگوله ای) می شود. متوازن شدن مصرف همانگونه که قبلا توضیح داده شد به کاهش هزینه ها می انجامد.

۲ـ۴ نیازمند های امنیتی

در این بخش به توضیح نیازمندی های این پروژه از دیدگاه امنیتی می پردازیم.

۲ـ۴ـ۱ حریم شخصی و محرمانگی

در فرایند بهینه سازی میزان مصرف و هزینه ی پرداختی، قطعا کاربران باید اطلاعاتی را با یک موجودیت مرکزی یا به صورت Distributed به اشتراک بگذارند. این اطلاعات می تواند تاریخچه ی مصرف انرژی و یا درخواست آن ها برای مصرف انرژی در روزهای آتی باشد. دقدقه ی مهمی که درباره ی اشتراک گذاری این داده ها وجود دارد، حریم خصوصی کاربران است. یعنی کاربران تا حد امکان مجبور نباشد برای انجام فرایند بهینه سازی، اطلاعاتی شامل جزییات مصرف و یا دستگاه های مورد استفاده خود را در اختیار دیگران بگذارند. بدین سبب باید تدابیر امنیتی خاصی در نظر گرفته شود تا در فرایند بهینه سازی در حد امکان از اطلاعاتی که ماهیت جمع دارند استفاده کرد و انتقال این اطلاعات نیز به صورت محرمانه انجام گیرد. همچین در صورت امکان باید از الگوریتم های رمزنگاری Homomorphic استفاده کرد و عملیات های مورد نیاز در فرایند بهینه سازی را روی داده های رمزنگاری شده انجام داد تا حتی در حین فرایند بهینه سازی داده هایی که جزو حریم شخصی کاربر هستند، فاش نشوند.

۲ـ۴ـ۲ احراز هویت

برای جلوگیری از ورود اطلاعات غلط به فرایند بهینه سازی و در راستای جلوگیری از سوء استفاده های احتمالی، لازم است هویت کاربرانی که قصد استفاده از فرایند بهینه سازی را دارند بررسی شود. و فقط کاربرانی که هویتشان مشخص است، اجازه ی تبادل اطلاعات با اجرا کننده ی فرایند بهینه سازی را داشته باشند.

فصل سوم

روش پیشنهادی برای بهینه سازی مصرف انرژی الکتریکی کاربردهای خانگی

در این فصل به توضیح روش پیشنهادی برای بهینه سازی میزان مصرف کاربران و پلتفرمی که برای اجرای این روش طراحی و پیاده سازی شده است، می پردازیم. قبل از توضیح درباره ی روش پیشنهادی، لازم است برخی تعاریف و اصطلاحات که در طراحی و پیاده سازی پروژه دخیل است را توضیح دهیم. تا در قسمت های بعدی گزارش بدون نیاز به توضیح اضافه آن ها را استفاده نماییم.

دستگاه دستگاه به وسایل الکتریکی و الکترونیکی موجود در خانه ها گفته می شود. البته در این گزارش منظور از دستگاه، وسایلی است که در زمان استفاده از آن ها انعطاف وجود دارد (وسیله ای مثل یخچال که باید همیشه روشن باشد، مد نظر نیست). دستگاه ها ممکن است به دو صورت خودکار و غیر خودکار باشند. دستگاه های خودکار بدون نیاز به دخلالت انسان از طریق زیرساخت های اینترنت اشیاء با اجرا کننده ی فرایند بهینه سازی ارتباط برقرار می کنند و داده های لازم را تبادل می کنند و در ساعات مد نظر نتیجه ی فرایند بهینه سازی به صورت خودکار خاموش یا روشن می شوند. دستگاه های غیر خودکار برای تبادل اطلاعات با اجرا کننده ی فرایند بهینه سازی نیاز به دخالت انسان دارند و همینطور برای اعمال نتیجه ی فرایند بهینه سازی کاربر انسانی باید آن ها را خاموش یا روشن نماید.

بردار هزینه یک بردار (آرایه) به طول ۲۴ است که هر خانه ی آن حاوی یک مقدار عددی نشانگر هزینه ی مصرف ۱kWh انرژی برق در ساعت شماره ی آن خانه است. برای مثال هزینه ی مصرف ۱kWh برق در ساعت ۲۰:۰۰ در خانه ی ۲۰ام این بردار قرار دارد.

بردار درخواست مصرف دستگاه یک بردار (آرایه) به طول ۲۴ است که هر خانه ی آن، حاوی یک عدد نمایانگر روشن یا خاموش بودن دستگاه در آن ساعت است. برای مثال یک ماشین لباسشویی را در نظر می گیریم. اگر کاربر در نظر داشته باشد این ماشین لباسشویی را از ساعت ۱۹:۰۰ تا ۲۱:۰۰ روشن (استفاده) کند، در خانه های ۱۹ام و ۲۰ام این بردار مقداری برابر میزان مصرف انرژی الکتریکی ماشین لباسشویی (که معمولا روی جعبه یا کاتالوگ آن درج شده است) قرار می گیرد. این مقدار بر حسب kWh است.

بردار مجموع درخواست مصرف خانه یک بردار (آرایه) به طول ۲۴ است که در هر خانه ی آن یک عدد نمایانگر جمع درخواست مصرف همه ی دستگاه های یک خانه در ساعت شماره ی آن خانه است. برای مثال اگر درخواست هایی مبنی بر استفاده از تلوزیون و لباسشویی و اتو در ساعت ۱۳:۰۰ وجود داشته باشد، در ۱۳امین خانه ی بردار مجموع درخواست مصرف خانه، عددی برابر با مجموع میزان مصرف برق این ۳ دستگاه در یک ساعت، قرار می گیرد.

جمع کل درخواست های مصرف خانه یک عدد است که از جمع المان های بردار مجموع درخواست مصرف خانه بدست می آید.

بردار مجموع درخواست مصرف همه ی خانه ها یک بردار(آرایه) به طول ۲۴ است که در هر خانه ی آن یک عدد نماینگر جمع درخواست مصرف دستگاه های موجود در همه ی خانه ها در ساعت شماره ی آن خانه است. برای مثال اگر خانه های دخیل در فرایند بهینه سازی برای ساعت ۱۳:۰۰ هر کدام به ترتیب برای مصرف ۴، ۵ و ۳ kWh درخواست داده باشند، در خانه ی ۱۳ام این بردار عدد ۱۲ قرار خواهد گرفت.

جمع کل درخواست های مصرف خانه ها یک عدد است که از جمع المان های بردار مجموع درخواست مصرف همه ی خانه ها بدست می آید.

بردار نتیجه ی بهینه ی سازی یک بردار (آرایه) به طول ۲۴ است که بعد از اتمام فرایند بهینه سازی به ازای هر دستگاه برای یک تاریخ خاص تولید می شود. اگر در هر خانه از این بردار عدد یک قرار گیرد، یعنی دستگاه مورد نظر باید در ساعت شماره ی آن خانه استفاده شود و اگر در خانه ای از بردار عدد صفر قرار گیرد، بدین معنی است که آن دستگاه نباید در ساعت شماره ی آن خانه استفاده شود.

۳ـ۱ راه حل پیشنهادی

هدف مشخص فرایند بهینه سازی که اقتباس مستقیم از روش عنوان شده در [۱، ۲، ۳] است، متوازن کردن میزان مصرف کاربران شبکه در طول شبانه روز است به طوری که تا حد امکان ساعتی به عنوان ساعت اوج مصرف وجود نداشته باشد و مجموع مصرف کاربران در ساعت های مختلف تقریبا برابر باشد. اولین قدم در این فرایند جمع آوری اطلاعات مورد نیاز برای فرایند بهینه سازی است. دومین قدم اجرا فرایند بهینه سازی و سومین قدم ارسال نتایج فرایند بهینه سازی برای کاربران(یا دستگاه ها) است.

فرایند بهینه سازی در هسته ی خود یک الگوریتم دارد که روی داده های ورودی اعمال می شود. هر خانه که در کل فرایند بهینه سازی شرکت می کند، به صورت مستقل این الگوریتم را اجرا می کند و در صورتی که هنوز به نتیجه ی مورد نظرش نرسیده باشد، طی فرایندی که بعدا توضیح داده می شود، باعث می شود در بقیه ی خانه ها الگوریتم بهینه سازی یک دور دیگر اجرا شود. این چرخه تا جایی ادامه پیدا می کند که همه ی خانه ها به نتیجه مورد نظر خود رسیده باشند. که طبق [۱] این وضعیت به صورت یک تعادل نش قابل دستیابی است.

با توجه به اینکه نتیجه این فرایند بهینه سازی یک بردار زمان بندی برای هر دستگاه است، از این به بعد برای سهولت بجای فرایند(یا الگوریتم) بهینه سازی از لفظ فرایند(یا الگوریتم) زمابندی استفاده خواهیم کرد.

۳ـ۱ـ۱ جمع آوری اطلاعات

برای اجرای هر دور از فرایند زمابندی به داده هایی به عنوان ورودی نیاز است. این داده ها شامل موارد زیر می شوند.

۱. بردار درخواست مصرف دستگاه (به ازای هر دستگاه یک بردار)

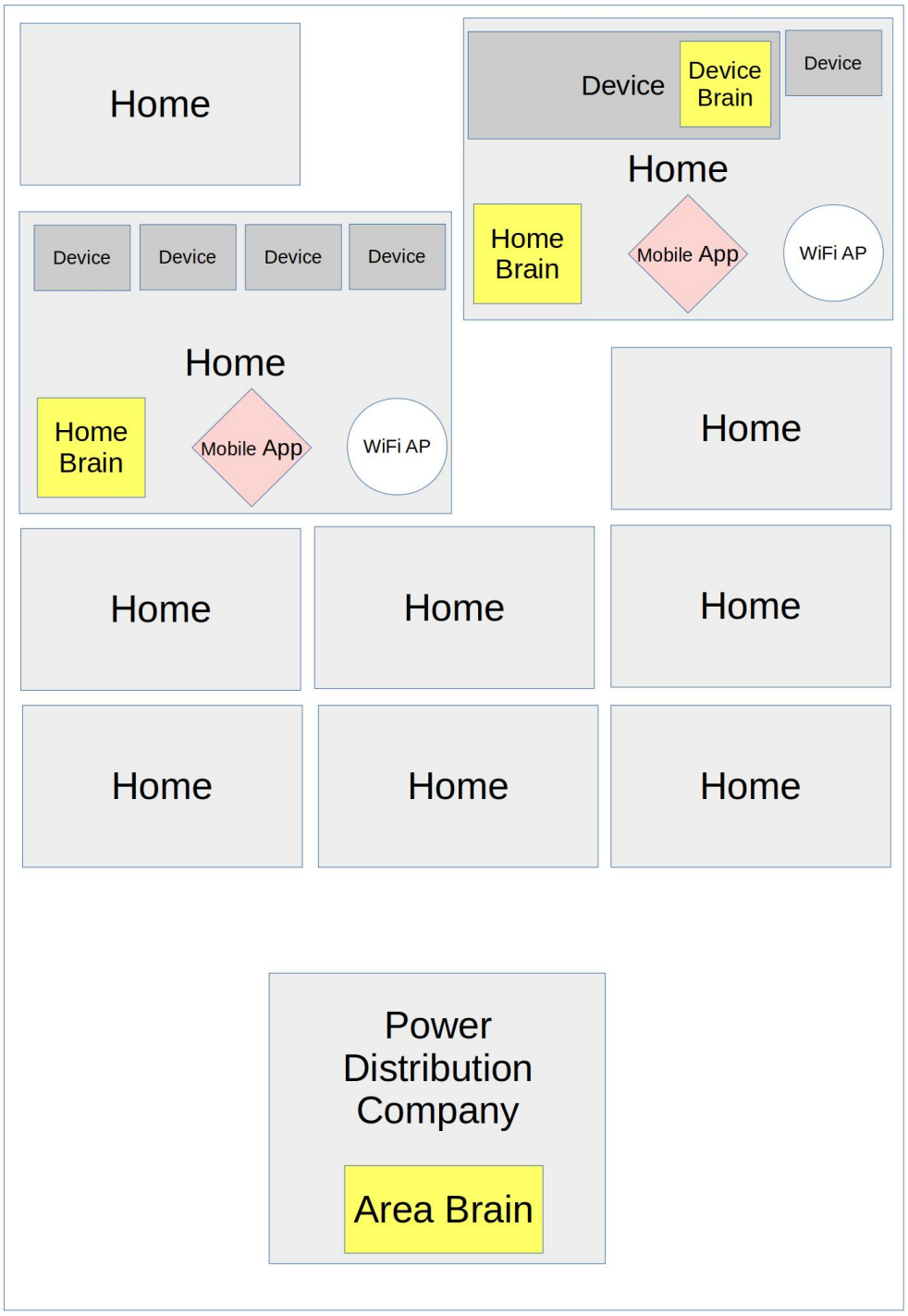
۲. بردار هزینه

۳. بردار مجموع درخواست مصرف همه ی خانه ها

۴. جمع کل درخواست های مصرف خانه ها

۳ـ۱ـ۲ موجودیت ها

در این بخش به توضیح درباره ی موجودیت های مختلفی که در طراحی فرایند زمابندی در نظر گرفته شده (و وظایف آن ها) می پردازیم.



شکل ۳ـ۱: موجودیت های دخیل در مساله با نگاه سطح بالا

در تصویر ۳ـ۱ موجودیت هایی که در مساله ی زمانبندی مطرح شده، دخلیل هستند(به صورت سطح بالا) را نمایش می دهد. در هر خانه چندین Device یا دستگاه وجود دارد که هر دستگاه مجهز به Device Brain

است. یک نرم افزار قابل اجرا روی تلفن همراه و یک واحد پردازشی برای اجرای عملیات های زمابنندی نیز در هر خانه وجود دارد. یک Area از چندین خانه تشکیل شده که واحدهای پردازشی فعال در خانه ها برای دست یابی به اطلاعات مورد نیازشان با Area Brain ارتباط برقرار می کنند. در دسترس قرار دادن Area Brain به عهده ی شرکت توزیع برق منطقه ای است.

۱.Device Brain:

- ارایه بردار درخواست مصرف دستگاه
- پیش بینی میزان انرژی مورد نیاز برای روز آینده(دستگاه های خودکار)

۲.Home Brain:

- ارایه ی بردار مجموع درخواست مصرف خانه
- اجرای الگوریتم زمان بندی مصرف
- ارایه ی بردار نتیجه ی بهینه ی سازی

۳.Mobile Application:

- رابط کاربری(افزودن/حذف دستگاه، اعلام میزان مصرف و ...)

۴.Area Brain:

- ارایه ی بردار مجموع درخواست مصرف همه ی خانه ها
- ارایه ی جمع کل درخواست های مصرف خانه ها

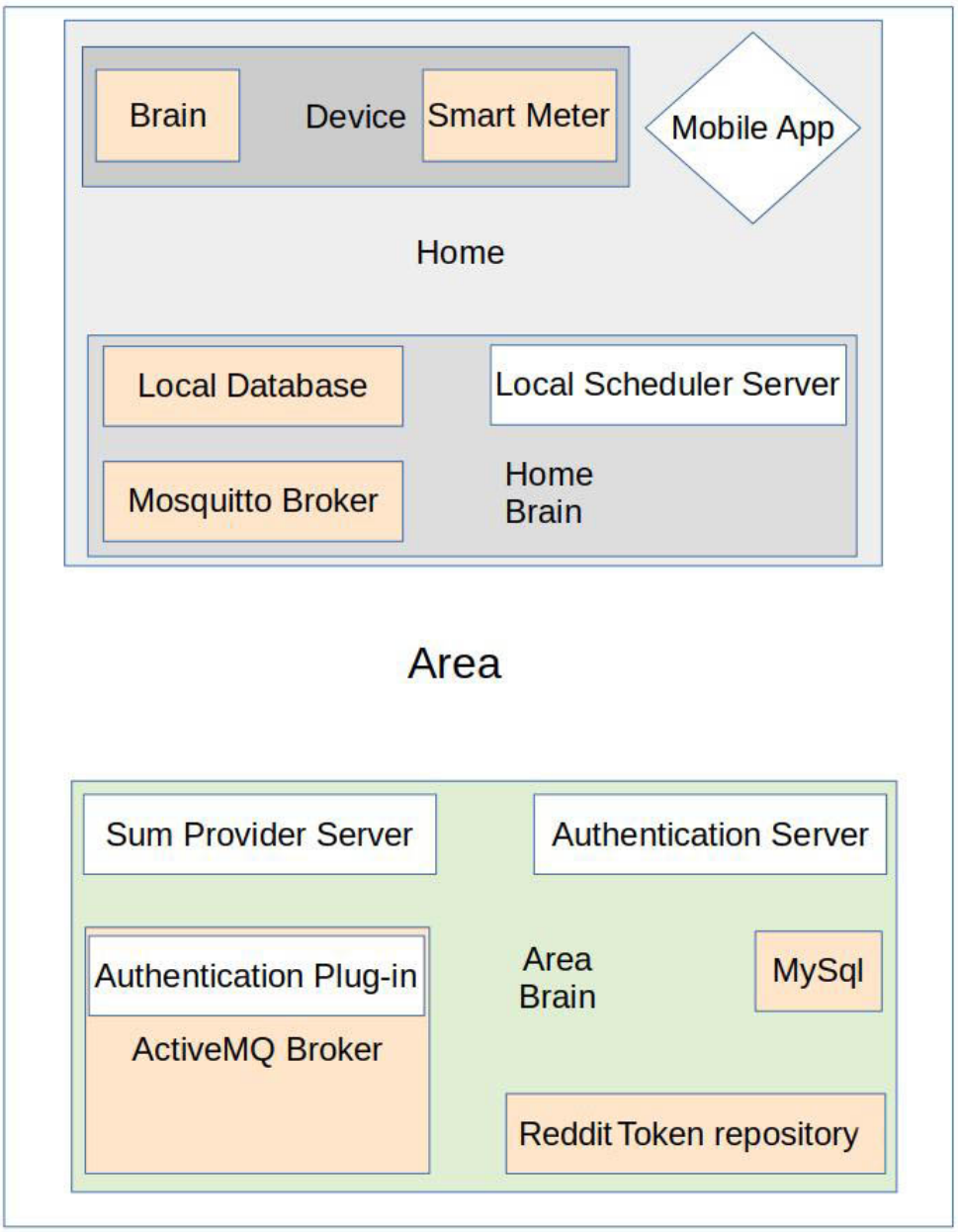
۵.User:

- استفاده از رابط کاربری و ثبت اطلاعات درخواست مصرف دستگاه های غیر خودکار

۳ـ۲ پلتفرم پیاده سازی شده

در این پروژه برای اجرای فرایند زمان بندی مدنظرمان، پلتفرمی طراحی و پیاده سازی کرده ایم. این پلفترم هر سه قدم جمع آوری اطلاعات، اجرای الگوریتم زمان بندی و ارسال نتایج در فرایند زمان بندی را پوشش می دهد. در طراحی این پلتفرم از معماری Microsevices استفاده شده و سعی شده تا حد امکان با شرایط و نیازمندی های حوزه ی اینترنت اشیاء مطابق باشد. در این پتفلرم چندین قطعه ی نرم افزاری با هم ارتباط بر قرار می کنند

تا بتوانند پس از جمع آوری اطلاعات مورد نیاز یک زمان بندی مناسب برای دستگاه های موجود در خانه های کاربران شبکه فراهم کنند. در ادامه به توضیح جنبه های مختلف این پلتفرم می پردازیم.



شکل ۳ـ۲: قطعات نرم افزاری پلتفرم

قسمت هایی از تصویر ۳ـ۲ که پس زمینه ی سفید دارند، در قالب این پروژه پیاده سازی شده اند و قسمت های با پس زمینه ی غیر سفید، یا نرم افزار آماده هستند و یا این پروژه شامل پیاده سازی آن ها نمی شده است.

۳ـ۲ـ۱ موجودیت های مفهومی پلتفرم

در تصویر ۳ موجودیت اصلی به نام Device ، Home و Area وجود دارد. Area منظور از منطقه ای است که کاربران در آن زندگی می کنند و خانه هایشان(Home) در آن قرار دارد. در هر خانه تعدادی دستگاه نیز وجود دارد. در هر دستگاه یک واحد پردازش به نام Device Brain وجود دارد که وظیفه اش در بخش قبلی ذکر شد. Device Brain به شبکه ی داخلی موجود در خانه (مثلا یک مودم اینترنت همراه با AccessPoint) دسترسی دارد و با Home Brain تبادل اطلاعات می کند. Device Brain باید مطابق با پلتفرم پیاده سازی شود و دستگاه های موجود در خانه به آن مجهز شوند.

و در هر خانه یک کامپیوتر (مثلا یک RaspberryPi) وجود دارد که نقش Home Brain را بازی می کند و به شبکه ی داخلی موجود در خانه دسترسی دارد. عملیات های پردازشی مربوط به فرایند بهینه سازی در Home Brain انجام می شود.

همچنین در هر خانه حداقل یک تلفن همراه وجود دارد که به Application توسعه داده شده به عنوان بخشی از این پروژه مجهز است. این تلفن همراه باید به شبکه ی داخلی موجود در خانه دسترسی داشته باشد.

۳ـ۲ـ۲ قطعات نرم افزاری معماری و وظایف آن ها

Mobile App تبادل داده های مربوط به دستگاه های غیرخودکار با Local Scheduler و رابط کاربری پلتفرم

Local Scheduler یک Server که با زبان جاوا پیاده سازی شده و بعد از دریافت اطلاعات از دستگاه ها و Application موبایل(از طریق Rest-Endpint) به اجرای الگوریتم زمابندی مشغول می شود و برای کامل شدن داده های مورد نیازش با Sum Provider ارتباط بر قرار می کند.

Mosquitto Broker یک Broker مناسب فضای اینترنت اشیاء و کامپیوتر های با منابع محدود است که برای تبادل پیام های MQTT استفاده می شود. از این Broker در ارسال نتایج عملیات زمانبدی برای دستگاه ها استفاده می شود.

Local Database یک پایگاه داده برای ذخیره داده های مورد نیاز Local Scheduler

Sum Provider یک Server که با زبان جاوا پیاده سازی شده و وظیفه اش انجام عملیات جمع رمزنگارانه ی Homomorphic روی داده های دریافتی از Local Scheduler ها (از طریق Rest-Endpoint)

است. همچنین وظیفه ی دیگر این قطعه ی نرم افزاری ارسال حاصل جمع ها برای Local Scheduler ها از طریق پیام MQTT است.

Authentication Server یک Server پیاده سازی شده با زبان جاوا است که عملیات های مربوط به پروتکل OAuth2 را انجام میدهد. کاربران برای استفاده از پلتفرم ، باید در این Server ثبت نام کنند. Local Scheduler ها برای تبادل داده با ActiveMQ Broker و Sum Provider باید توسط Authentication Server و از جانب کاربر صاحب خانه احراز هویت شوند. فرایند احراز هویت و صدور اجازه ی ارتباط برای قطعات نرم افزاری یاد شده توسط Authentication Server انجام می شود.

Reddit یک پایگاه داده ی مناسب ذخیره ی Token است که در فرایند احراز مورد استفاده قرار میگیرد

MySql Database پایگاه داده ای است که برای ذخیره ی داده های مورد نیاز Sum Provider و Authentication Server استفاده می شود.

ActiveMq Broker یک Broker پیام است که در این پلتفرم برای ارسال حاصل جمع ها از Sum Provider به Local Scheduler ها استفاده می شود.

۳ـ۳ سناریوی کامل استفاده از پلتفرم

در این قسمت با این فرض که کلیه ی نرم افزارها و سخت افزارهای مورد نیاز نصب شده و آماده ی ارایه خدمات هستند، فرایند استفاده و فعالیت پلتفرم را توضیح می دهیم. یک شخص بعد از ثبت نام برای استفاده از پلتفرم، نام کاربری و رمز عبور خود را در بخش ورود Application موبایل وارد می کند. در صورت صحت اطلاعات، Authentication Server در پاسخ یک Refresh-Token ارسال می کند. Application موبایل این Refresh-Token را به Local Scheduler اطلاع می دهد تا Local Scheduler امکان استفاده از خدمات Sum Provider را داشته باشد. سپس کاربر به کمک Application موبایل دستگاه های غیرخودکار موجود در خانه اش را به Local Scheduler معرفی می کند و اطلاعاتی مثل میزان مصرف انرژی الکتریکی در یک ساعت توسط دستگاه را وارد می کند.

در ابتدای هفته کاربر به کمک Application موبایل، ساعاتی که قصد دارد از دستگاه ها (به تفکیک تاریخ) استفاده کند را مشخص می کند و Application موبایل اطلاعات این درخواست ها را برای Local Scheduler ارسال می کند. در هر روز هفته کاربر فقط اجازه دارد اطلاعات درخواست مصرفش برای روزهای آینده را وارد کند. همچنین کاربر اجازه دارد در هر روز، اطلاعات مربوط به درخواست مصرف فردا را تا ساعت ۱۳:۰۰ آن

روز وارد کند.

دستگاه های خودکار هم درخواست های مصرف خود را بر اساس قواعدی مثل قواعدی که در بالا برای دستگاه های غیر اتوماتیک ذکر شده، برای Local Scheduler ارسال می کنند. خودکار بودنشان به این معنا است که خود دستگاه ها بدون دخالت انسان، زمان های مناسب برای روشن یا خاموش بودنشان را تشخیص می دهند(پیش بینی می کنند).

راس ساعت ۱۳:۰۰ هر روز، Local Scheduler عملیات زمان بندی درخواست های مصرف مربوط به فردا را آغاز می کند. برای اجرای الگوریتم زمان بندی علاوه بر بردارهای درخواست مصرف دستگاه ها به بردار مجموع درخواست مصرف همه ی خانه ها نیاز است. در اولین دور از اجرای الگوریتم Local Scheduler از یک بردار با مقادیر Random به عنوان بردار مجموع درخواست مصرف همه ی خانه ها استفاده می شود. همچین برای اجرای الگوریتم زمابندی به بردار هزینه نیاز است که فعلا فرض می کنیم Local Scheduler به آن دسترسی دارد. اگر بعد از اجرای یک دور الگوریتم زمابندی، به نتیجه ی دلخواه (بهینه) رسید که داده های مربوط به نتیجه ی بهینه و زمان بندی مصرف ذخیره می شود تا بعدا به اطلاع دستگاه ها برسد. اگر نتیجه ی دلخواه بعد از یک دور حاصل نشده بود، Local Scheduler بردار مجموع درخواست مصرف خانه و عدد جمع کل درخواست های مصرف خانه را برای Sum Provider به صورت رمز شده با کلید عمومی Sum Provider ارسال می کند(اجازه ی ارسال این داده را به واسطه ی Refresh-Token بدست می آورد).

Sum Provider هر بار که داده های ذکر شده را از هر خانه دریافت کند، بدون رمزگشایی اطلاعاتی که دریافت کرده، بردار مجموع درخواست مصرف همه ی خانه و عدد جمع کل درخواست های مصرف خانه ها را محاسبه می کند و نتیجه را به صورت یک پیام MQTT به اطلاع همه ی Local Scheduler های فعال در خانه ها میرساند.

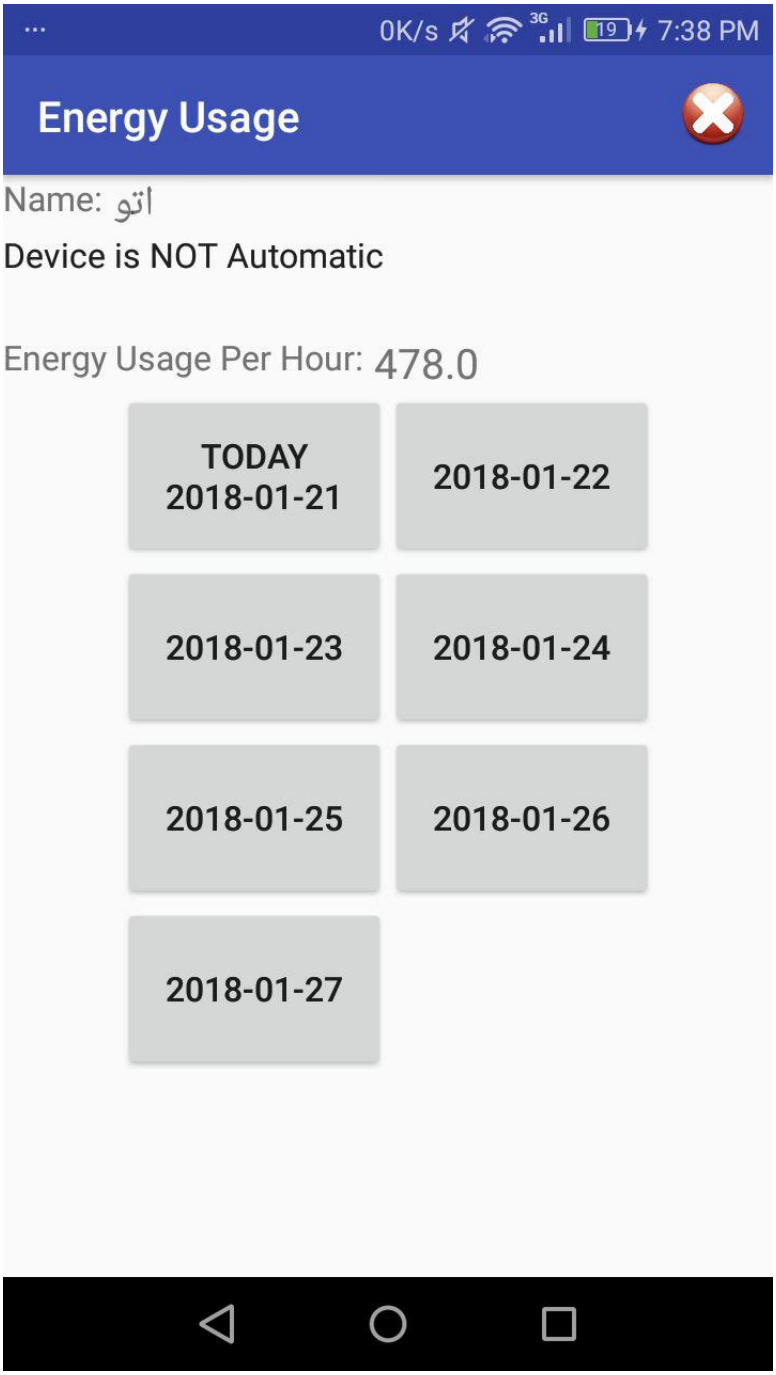
Local Scheduler ها دریافت این اطلاعات را نشانه ی تغییر اطلاعات ورودی الگوریتم زمابندی می دانند. در نتیجه با اطلاعات دریافتی جدید یک دور دیگر الگوریتم زمان بندی را اجرا می کنند.

و چرخه ی بالا تا جایی ادامه پیدا می کند که همه ی Local Scheduler ها به زمابندی بهینه ی مورد نظر خود رسیده باشند. این نتیجه در واقع بردار نتیجه ی بهینه سازی است.

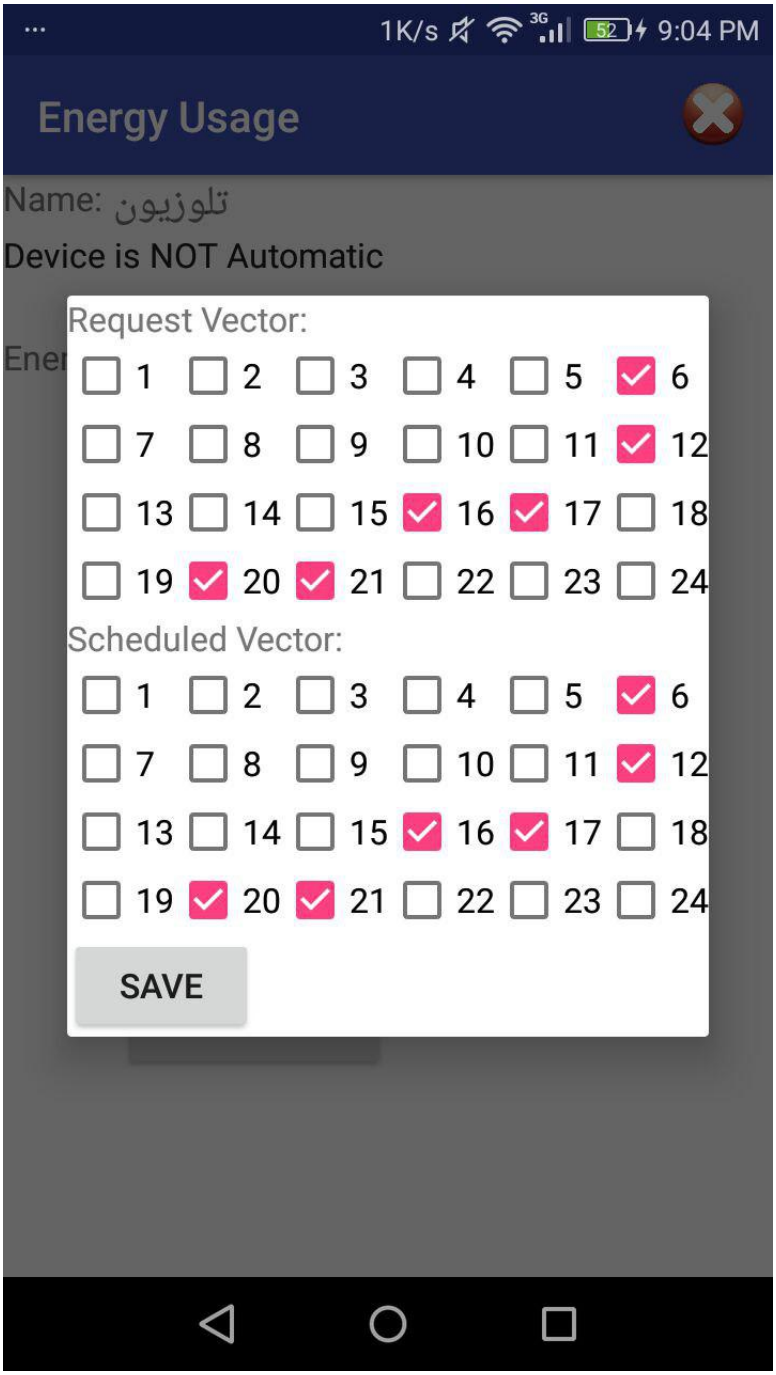
در ساعت همان روز ۲۲:۰۰، Local Scheduler بردار نتیجه ی بهینه سازی مربوط به هر دستگاه را برایش ارسال می کند. بردار نتیجه ی بهینه سازی مربوط به دستگاه های خودکار برای خودشان و بردار مربوط به دستگاه های غیر خودکار برای Application موبایل (هر دو دسته داده به صورت پیام MQTT) ارسال می شود.

هم اکنون کاربر با مراجعه به Application موبایل زمابندی بهینه برای استفاده از دستگاه های غیر خودکار را مشاهده می کند. و دستگاه های خودکار نیز زمان بندی را دریافت کرده اند و طبق آن روشن یا خاموش خواهند شد.

شکل ۳ـ۳: لیست دستگاه ها در Application موبایل



شکل ۳ـ۴: صفحه ی نمایش اطلاعات دستگاه اتو



شکل ۳ـ۵: صفحه ی وارد کردن درخواست مصرف

۳ـ۴ الگوریتم بهینه سازی

در قطعه ی نرم افزاری Local Scheduler پلتفرم ، فضایی برای پیاده سازی یک الگوریتم وجود دارد که وظیفه ی اصلی پلفترم به عهده ی آن است. این الگوریتم باید با استفاده از تکنیک هایی مثل بهینه سازی بردار به یک بردار زمان بندی مناسب برای دستگاه های موجود در هر خانه برسد. اطلاعات ورودی این الگوریتم در بخش های قبلی گفته شد.

پلتفرم این قابلیت را دارد که از الگوریتم های مختلفی پیشتیبانی کند. یعنی برای تغییر الگوریتم زمان بندی نیازی به تغییر در کل پلتفرم نیست. فقط قسمتی از پلتفرم که مربوط به پیاده سازی الگوریتم است تغییر می کند. با توجه به اینکه تمرکز این پروژه روی طراحی و پیاده سازی پلتفرم بوده است، از توضیح درباره ی جزءیات طراحی و پیاده سازی الگوریتم استفاده شده عبور می کنیم. برای کسب اطلاعات بیشتر درباره ی الگوریتم استفاده شده به [۴، ۵] مراجعه کنید.

۳ـ۵ نوآوری ها

در این بخش به توضیح اختصاصی نوآوری هایی که در خلل طراحی و پیاده سازی این پروژه انجام شده و تفاوت آن ها با روش های قبلی می پردازیم.

۳ـ۵ـ۱ استفاده از الگوریتم رمزنگاری Paillier برای حفظ حریم شخصی

روش پیشنهادی در [۲] که در [۱] کامل تر و مشکلات حریم خصوصی آن بر طرف شده است، بر منبای عدم حظور یک Server مرکزی در چرخه ی تبادل اطلاعات بنا شده است. در واقع تمامی داده های مورد نیاز برای حصول بردار نتیجه ی زمابندی، توسط نودهای پردازشی ای که در خانه ها نصب شده، تولید شده و این داده ها به صورت محرمانه بین این نود ها به صورت Pear to Pear مبادله می شود. اما در پلتفرم بین واحد های پردازنده موجود در خانه ها(بین Local Scheduler ها) داده ای تبادل نمی شود و داده هایی که لازم است به اشتراک گذاشته شود ابتدا برای یک Server مرکزی (Sum Provider) ارسال می شود و بعد از انجام عملیات های Homomorphic روی آن ها، نتایج برای همه ی نودهای پردازشی موجود در خانه های ارسال می شود.

دلیل این تغییر در طراحی، پیچیده بودن پیاده سازی یک سیستم Pear to Pear در واقعیت است. همچنین حجم پردازش هایی که نودهای پردازشی باید در حالت Pear to Pear انجام دهند (برای مدیریت ارتباط ها) بسیار بالاتر حالتی که است که یک Server مرکزی عملیات های Homomorphic را انجام دهد.

وقتی صحبت از استفاده از Server مرکزی به میان می آید، اولین نگرانی کاربران حریم شخصی آن هاست که ممکن است به ۲ صورت در خطر باشد.

۱ـ در زمان انتقال داده ها به Server مرکزی ۲ـ Server مرکزی به عنوان یک شخص سوم ممکن است مورد اعتماد همه ی کاربران نباشد

برای حل این مساله در پلتفرم از الگوریتم رمزنگاری Paillier استفاده شده است که امکان انجام عملیات های مختلف روی داده های رمزنگاری شده بدون نیاز به رمزگشایی آن ها را می دهد. در این پلتفرم داده هایی که

از سمت Local Scheduler برای Sum Provider ارسال می شوند، توسط کلید عمومی Server با الگوریتم Paillier رمز می شوند. در نتیجه در مسیر انتقال، حریم شخصی کاربران به خطر نمی افتد.
Sum Provider فقط اخرین داده های دریافتی از Local Scheduler ها را به همان صورت که دریافت کرده(رمزنگاری شده) ذخیره می کند. و عملیات مد نظرش را به صورت Homomorphic روی داده های رمز شده انجام می دهد. نتیجه ی این عملیات شامل حریم خصوصی کاربران نمی شود پس رمزگشایی شده و به اطلاع Local Scheduler ها می رسد.
به صورت کلی اگر به پیاده سازی پلتفرم اعتماد کنیم، حریم خصوصی کاربران به خطر نمی افتد.

۳-۵-۲ طراحی و پیاده سازی یک روش احراز هویت بر پایه ی Token برای پروتکل MQTT

به صورت پیش فرض احراز هویت در MQTT فقط به دو روش مبتنی بر گواهینامه (SSL) و مبتنی بر User-name/Password ممکن است. در روش اول Client های MQTT باید گواهینامه داشته باشند که با شرایط اینترنت اشیاء مطابقت ندارد.
در روش دوم اکثر Broker ها از یک ساختار فایلی برای ذخیره ی اطلاعات Client ها استفاده می کنند که نه از دیدگاه امنیت قابل قبول است نه از نظر قابلیت گسترش پذیری. همچنین در سناریوهای اینترنت اشیاء بسیار رایج است که یک برنامه از جانب یک کاربر تقاضا ارسال اطلاعات برای یک Server را داشته باشد(مثلا یک برنامه ی درحال اجرا روی RaspberryPi دمای اتاق را هر ۵ ساعت یکبار برای یک Server ارسال کند). در چنین مواقعی ذخیره کردن Username/Password کاربر روی دستگاهی که آن برنامه را اجرا می کند، از نظر امنیتی مشکلاتی را ایجاد می کند. همچنین اگر ارتباط Client با Broker به کمک SSL امن نشده باشد، اطلاعات کاربر به صورت Plaintext در پکت های بسته های MQTT مشخص است. در نتیجه بهتر است برای احراز هویت Client های MQTT از روش مبتنی بر Username/Passwrod به صورت مستقیم استفاده نشود.

روش پیشنهادی: کاربر باید ابتدا در یک Server که به Server احراز هویت معروف است وارد شود. سپس یک Token دریافت کند. حال کاربر می تواند با ارسال این Token به Server های ارایه کننده ی خدمات (مثل Broker ها) خدمات دریافت کند. یا Token دریافتی را به برنامه هایی بدهد که قرار است از جانب او خدماتی را دریافت کنند. Server های ارایه کننده ی خدمات برای احراز هویت، Token دریافتی را برای Server احراز هویت ارسال می کنند و معتبر بودن آن را بررسی می کنند. برای بکارگیری این Token در MQTT باید Token را در پیام های ایجاد ارتباط با Broker به عنوان ClientID ارسال کرد. Broker با

دریافت پیام درخواست ایجاد ارتباط، به افزونه ای که در این پروژه طراحی شده است، مراجعه می کند و معتبر بودن Token را(از طریق Server احراز هویت) بررسی می کند. در صورت معبتر بودن، افزونه اجازه ی ایجاد Connection را می دهد و برنامه می تواند از جانب کاربری که هویتش در Server احراز هویت تایید شده، برای Broker پیام های MQTT ارسال کند.

در این روش برای پیاده سازی Server احراز هویت و افزونه ی احراز هویت از استاندار OAuth2 استفاده کردیم.

۳ـ۶ وضعیت آزمایشی

در حال حاظر پلتفرم در مرحله ی آزمایشی است . در زیر لیست فعالیت هایی قرار داده شده که برای خروج پلتفرم از وضعیت آزمایشی و اجرایی شدن در محیط واقعی لازم است.

- اضافه کردن مکانیزم جلوگیری از انجام عملیات اضافه در Sum Provider
- اضافه کردن امکان ثبت کردن کاربران در پلتفرم با Application موبایل
- اضافه کردن امکان ارسال بردار هزینه برای Local Scheduler ها توسط یک قطعه ی نرم افزاری جدید
- ایجاد زیرساخت نرم افزاری برای تبدیل قطعات نرم افزاری موجود در پلتفرم به Container های آماده ی استفاده ی Docker
- رفع مشکل عدم نمایش روز اول هفته ی آینده در Application موبایل
- افزودن شرط های محدودیت ساعت ارسال درخواست مصرف
- فارسی کردن تقویم Application موبایل
- بهبود رابط کاربری Application موبایل

فصل چهارم

نتیجه گیری

در این پروژه به تحلیل یکی از مسایل اثرگذار در حوزه توزیع انرژی برق یعنی متوازن کردن میزان مصرف انرژی در ساعات مختلف شبانه روز برای کاربران (عموما خانگی) پرداختیم. و سعی شد با استفاده از دانش مهندسی نرم افزار و امنیت اطلاعات راه حل هایی که در [۲، ۳، ۱] ارایه شده بود را در قالب یک سیستم نرم افزاری طراحی و پیاده سازی کنیم. نتیجه یک پلتفرم نرم افزاری دارای بخش ها و قطعات نرم افزاری مختلف بود که این قطعات از طرق مختلفی با هم ارتباط داشتند و داده های مختلفی تبادل می کردند تا در نهایت یک زمانبدی مناسب برای استفاده از دستگاه های الکتریکی موجود در خانه ها ارایه شود.

این پلتفرم پتانسیل و زیرساخت مناسب برای تبادل داده های مختلف در راستای حل مسایل دیگری را نیز دارد. شرکت های توزیع برق منطقه ای می توانند با به کارگیری این پتلفرم و ترغیب کاربران به استفاده از آن، به بهینه سازی مصرف انرژی کمک کنند. و همچنین به سمت مکانیزه تر کردن خدماتشان حرکت کنند. برای مثال می توان از Application موبایل این پلتفرم به عنوان درگاه خدمات الکترونیک شبکه ی توزیع برق برای کاربران استفاده کرد. و خدماتی مثل نمودار های میزان مصرف ، پرداخت هزینه ی مصرف و اطلاع رسانی را بر بستر اینترنت به کاربران ارایه داد.

امید است که با به کار گیری علم و تکنولوژی زمینی سبز تر و هوایی پاک تر داشته باشیم.

فصل پنجم

ضمیمه

۵-۱ کد منبع

برای دسترسی به کد منبع پلتفرم و بررسی مراحل پیاده سازی آن به آدرس

`https://gitlab.com/bmd007/bachelors_project`

مراجعه کنید. در این صفحه می توانید علاوه بر کد منبع پلتفرم ، گزارش هایی درباره ی پروتکل های مورد استفاده در حوزه ی اینترنت اشیاء، مشاهده فرمایید. این گزارش ها در قالب تحقیقات اولیه برای طراحی پلتفرم تهیه شده اند و پروتکل های CoAP ، XMPP ، UPNP و MQTT را پوشش می دهند.

۵-۲ توضیح پروتکل CoAP

Constrained Application Protocol (CoAP) یک پروتکل Network-Orineted برای انتقال Documen-t است که شباهت های بسیار به HTTP دارد. هدف از طراحی این پروتکل استفاده ی آن در دستگاه هایی با توان پردازشی پایین و منبع تغذیه ی ضعیف و کم عمر بوده است. کاربردهای این پروتکل اساسا به اینترنت اشیاء مربوط می شود که از: smart energy, smart grid, building control, intelligent lighting control, industrial control systems, asset tracking, environment monitoring می توان به عنوان مثال

یاد کرد.

۵ـ۲ـ۱ مقایسه با HTTP

به دلیل شباهت بسیار زیاد CoAP با HTTP ، مقایسه ی این دو باعث مشخص شدن ویژگی های کلیدی CoAP خواهد شد.

لایه ی انتقال

بر خلاف HTTP که در لایه ی انتقال از TCP استفاده می کند، فقط آدرس های Unicast را در حالت Synchrounos پشتیبانی و فقط با معماری Client/Server کار میکند؛ Coap از پروتکل UDP استفاده می کند. در نتیجه سربار کنترل Congestion و نگه داری Connection حذف می شود، امکان ارسال پیام MultiCast فراهم می شود و همچنین تبادل پیام به صورت Asynchrounos ممکن می شود. در صورتی یک کاربرد خاص نیاز به Reliability در انتقال پیام داشته باشد، در لایه ی Application مکانیزهای مناسب در نظر گرفته شده است.

مدل ارتباط Observe

این پروتکل با اقتباس از مدل Request در پروتکل HTTP امکان Observe کردن یک Resource را فراهم کرده. در واقع یک Flag به اسم Observe در بسته ی Request وجود دارد که اگر مقدار دهی شود، Server بعد از ارسال پاسخ اولیه، به ارسال پاسخ ادامه می دهد(در واقع Server ارتباط را قطع نمی کند). این امکان باعث ایجاد مکانیزمی شبیهPush-Notification می شود که Server می تواند تغییر یک وضعیت مثل تغییر مقدار یک سنسور را به صورت آنی به ارسال کننده ی Request اطلاع دهد.(در پروتکل CoAP نود های سنسور به صورت Server شناخته می شوند و با ارسال Request با متدی مثل Get مقدار سنسور را برای درخواست کننده در پاسخ ارسال می کنند. این مساله در فضای کاری HTTP با توجه به سنگین و پیچیده بودن پروتکل و ضعیف بودن نودهای کنترل کننده ی سنسور، تقریبا غیر ممکن و در واقع غیر قابل Scale است.).

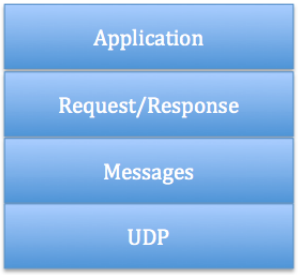
کشف سرویس

در پروتکل CoAP این قابلیت وجود دارد که دستگاه ها و نودهای فعال به عنوان موجودیت اینترنت اشیاء، قابلیت ها و سرویس هایی که ارایه می دهند را به اطلاع دیگر موجودیت ها برسانند. به این منظور هر نود باید یک لیست از منابع و سرویس هایی را که در دسترس قرار داده به همراه توضیحات لازم (Meta-Data) تحت یک لینک (URL) استاندارد منتشر کند. دیگر نود ها با ارسال درخواست Get به این لینک، لیست را دریافت کرده و در صورت لزوم با استفاده از Meta-Data ی موجود در لیست سرویس مورد نظر را شناسایی

و فراخوانی می کنند. این قابلیت به صورت پیش فرض در HTTP وجود ندارد و به برای این منظور از پروتکل های Service Discovery مستقل استفاده می شود.

۵ـ۲ـ۲ پیام ها

در لایه ی بالای لایه ی انتقال ، CoAP از ۳ زیر لایه انتزاعی تشکیل شده که به ترتیب از پایین به بالا Message, Request/Response, Application هستند.

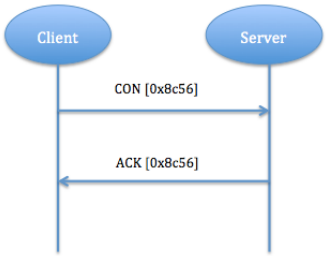


شکل ۵ـ۱ : لایه های Coap به صورت انتزاعی

لایه ی Message

لایه ی Message وظیفه ی مواجه شدن با UDP و تبادل داده به صورت Asynchronous را دارد. پیام های CoAP به واسطه ی این لایه به چهار نوع تقسیم می شوند. Con (Confirmable), NON (Non-Comfimable), ACK (Acknowlegdment), RST (Reset)

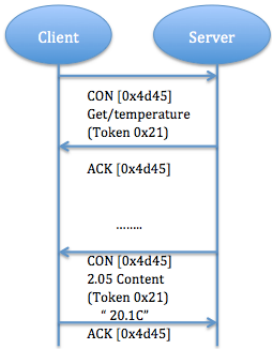
پیام هایی که به صورت Con ارسال می شوند باید توسط گیرنده با یک پیام ACK پاسخ داده شوند. اگر گیرنده نتوانست پیام را دریافت کند، یک پیام از نوع RST ارسال می کند. و به این ترتیب می توان Datagram های UDP را به صورت Reliable ارسال کرد(شکل ۵ـ۲) .



شکل ۵ـ۲ : ارسال پیام به صورت Reliable

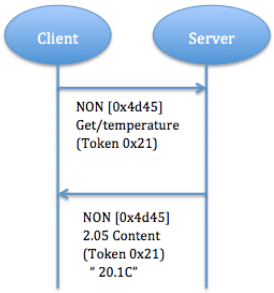
اگر پیام توسط فرستنده به صورت NON ارسال شوند هیچ پاسخی مبنی بر Acknowledge از گیرنده دریافت نمی کند. اما در صورتی که گیرنده در دریافت پیام با مشکلی مواجه شد یک پیام RST به عنوان پاسخ ارسال می کند.

از ساختاری که ذکر شد برای پیاده سازی مدل های Request/Response استفاده شده به طوری که Client می تواند Request خود را به صورت CON ارسال کند و پاسخ آن را در ACK دریافت کند. همچنین Server می تواند پاسخ را به طور مجزا در قالب یک پیام CON ارسال کند(در این حالت ۲ پیام CON و ۲ پیام ACK تبادل خواهد شد(شکل ۵ـ۳)) .



شکل ۵ـ۳: مدل Separate-Request/Respsone

حتی امکان ارسال request و respone در قالب NON وجود دارد که نه Server نه Client منتظر ACK نخواهند بود (شکل ۵ـ۴) .



شکل ۵ـ۴: مدل Non-Comfirmable-Request/Respsone

۵ـ۲ـ۳ ساختار پیام ها

پیام های CoAP در واقع داده های Binary هستند که بخش Data را در Datagram های UDP به خود اختصاص می دهند. طول بخش ثابت Header پیام های CoAP ۴ بایت است. این Header شامل یک بخش Token و یک بخش Options و یک بخش Payload همگی با طول های متغیر است.

Table 3 Message Format

0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1				
Ver	T	OC	Code	MessageID
Token (if any, TKL bytes)…				
Options (if any)…				
Payload (if any)…				

شکل ۵ـ۵: ساختار پیام های CoAP

قسمت Code نماینگر نوع پیام(CON یا NON یا …) است. قسمت MessageID نیز در فرایندهایی مثل ارسال ACK و RST کاربرد دارد.

۵ـ۲ـ۴ امنیت

با توجه به اینکه اینترنت اشیاء در جنبه های حساسی از زندگی تاثیر گذار خواهد بود، نیاز به امن بودن فضای تبادل اطلاعات در این حوزه کاملا جدی است. به این منظور CoAP امنیت را از طریق بکار گیری پروتکل DTLS فراهم کرده است. DTLS به نوعی نمونه ی همزاد SSL/TLS است که در لایه ی انتقال از پروتکل UDP بهره می برد. به این ترتیب مناسب CoAP و محدودیت های فضای اینترنت اشیاء است.DTLS تمامی ۳ نیاز اصلی در حوزه ی امنیت یعنی محرمانگی انتقال داده ها، احراز هویت و اطمینان از صحت داده های دریافتی را، به صورت End-to-End و بدون اعمال سربار هایHeader رمزنگارانه در لایه های پایین تر شبکه، ارایه می کند.

۵ـ۲ـ۵ اینترنت اشیاء

پروتکل CoAP اساسا برای استفاده در حوزه ی اینترنت اشیاء طراحی شده. در طول گزارش به جنبه های مختلف طراحی این پروتکل پرداختیم. در اینجا به اختصار لیستی از ویژگی هایی که این پروتکل را برای اینترنت اشیاء مناسب کرده ذکر می شود:

* **۱:** معماری این پروتکل بر اساس معماری RSETful بنا شده در نتیجه مقادیر سنسورها و سرویس های عملگرها میتوانند در قالب یک آدرس URL با متد های Post, Get و … در دسترس قرار گیرند.
* **۲:** امنیت End-to-End به واسطه ی استفاده از پروتکل DTLS فراهم شده که این پروتکل در لایه ی Application فعالیت می کند و در لایه ی انتقال از UDP بهره می برد. در نتیجه با کمترین سربار محاسباتی و کمترین سربار شبکه امنیت مورد نیاز فراهم می شود.
* **۳:** تبادل پیام به صورت Asynchrounos (غیر همزمان) امکان پذیر است در نتیجه CPU های دستگاهایی که در طرفین ارتباط قرار دارند Cycle ها خود و انرژی منبع تغذیه را صرف انتظار کشیدن برای دریافت

پاسخ نمی کنند. هر زمان که پاسخ به بدست نود رسید، CPU به واسطه ی یک CallBack نرم افزاری از در دسترس قرار گرفتن آن آگاه شده و پردازش های لازم را روی آن انجام می دهد.

* **۴:** اندازه کم Header و کم بودن پیچیدگی Pars کردن آن باعث مصرف کمتر Cycle های Cpu و مصرف کمتر Ram خواهد شد.

* **۵:** پشتبانی از URI و Content-type و Accept که باعث می شود نوع داده های مختلف به راحتی به واسطه ی این پروتکل قابل انتقال باشند.

* **۶:** توانایی های کش کردن و پروکسی کردن باعث می شود دستگاه های مبتنی بر CoAP بتوانند به راحتی با سیستم های مبتنی بر HTTP صحبت کنند.

* **۷:** امکان به کاری گیری Reliability در لایه ی Application در صورت نیاز.

* **۸:** پشتیبانی از آدرس های Unicast و .Multicast

* **۹:** کم بودن حجم پیام ها و تعداد آن ها پروتکل را برای محیط هایی که هزینه ی پهنای باند بالایی دارند،به گزینه ی مناسبی تبدیل کرد است.

کتاب‌نامه

[1] Mohammad Ashiqur Rahman, Mohammad Hossein Manshaei, Ehab Al-Shaer and Shehab, Mohamed. Secure and private data aggregation for energy consumption scheduling in smart grids. *IEEE Transactions on Dependable and Secure Computing*, 2015.

[2] A.-H. Mohsenian-Rad, V.W.S. Wong, J. Jatskevich-R. Schober and Leon-Garcia, A. utonomous demand-side management based on game-theoretic energy consumption scheduling for the future smart grid. *IEEE Transactions on Smart Grid, 1(3):320–331*, 2010.

[3] Z. Baharlouei, M. Hashemi, H. Narimani and Mohsenian-Rad, H. Achieving optimality and fairness in autonomous demand response: Benchmarks and billing mechanisms. *IEEE Transactions on Smart Grid, 4(2):968–975*, 2013.

[4] Boyd, S. and Vandenberghe, L. Convex optimization. *Cambridge University Press*, 2004.

[5] Bertsimas, D. and Tsitsiklis, J. N. Introduction to linear optimiza-tion. *Athena Scientific*, 2009.

[6] Irakli Nadareishvili, Ronnie Mitra, Matt Mclarty Mike Amundsen. *Microservice Architecture ALIGNING PRINCIPLES, PRACTICES, AND CULTURE*. O'REILLY, 2016.

[7] Badola, Vineet. www.cloudacademy.com/blog/microservices-architecture-challenge-advantage-drawback, 2015.

[8] Anicas, Mitchell. www.digitalocean.com/community/tutorials/an-introduction-to-oauth-2, 2014.

[9] www.docs.apigee.com/api-services/content/oauth introduction. Introduction to oauth 2.0.

[10] official website of OAuth2. www.oauth.net.

[11] Washer, Peter. *Learning Internet of Things*. Packt, 2015.

[12] Apache. www.activemq.apache.org.

[13] Mqtt. www.mqtt.org.

[14] implementation of Paillier cryptosystem in Java. www.github.com/kunerd/jpaillier.

[15] HIVEMQ, MQTT Essentials: Part 1 – Introducing MQTT. www.hivemq.com/blog/mqtt-essentials-part-1-introducing-mqtt.

[16] HIVEMQ, MQTT Essentials: MQTT Essentials Part 2: Publish and Subscribe. www.hivemq.com/blog/mqtt-essentials-part2-publish-subscribe.

[17] Janakiram MSV, Get to Know MQTT: The Messaging Protocol for the Internet of Things. www.thenewstack.io/mqtt-protocol-iot/, 2016.

[18] Margaret Rouse, MQTT (MQ Telemetry Transport). www.internetofthingsagenda.techtarget.com/definition/mqtt-mq-telemetry-transport, 2016.

[19] solace.com, MQTT Control Packet format. www.docs.solace.com/mqtt-311-prtl-conformance-spec/mqtt2016.

[20] eclipse.org, MQTT and CoAP, IoT Protocols. www.eclipse.org.

[21] CoAP. www.coap.technology.

[22] Xi Chen, Constrained Application Protocol for Internet of Things. www.www.cs.wustl.edu/ jain/cse574-14/ftp/coap/index.html, 2014.

[23] Shovic, John C. *Raspberry Pi IoT Projects*. apress, 2016.

[24] Monk, Simon. *Raspberry Pi Cookbook*. OREILLY, 2016.

[25] Poole, Matthew. *Building a Home Security System with Raspberry Pi*. PACKT, 2015.