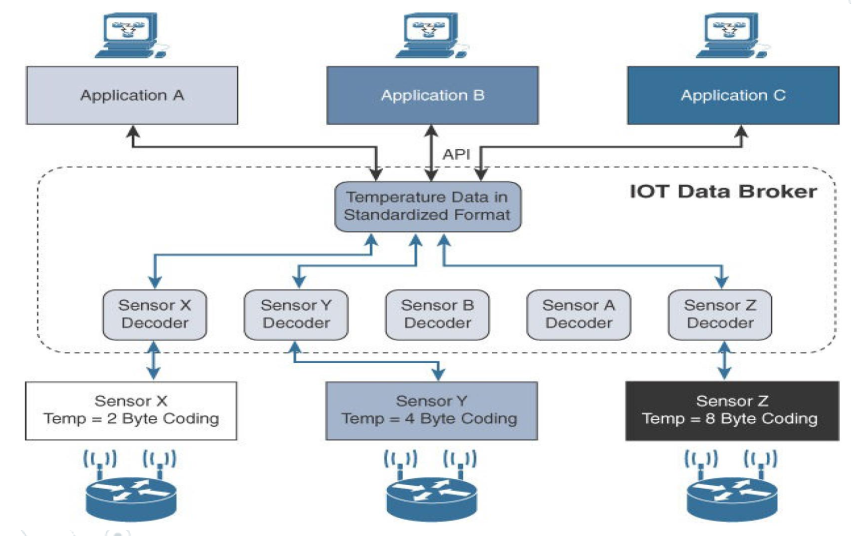
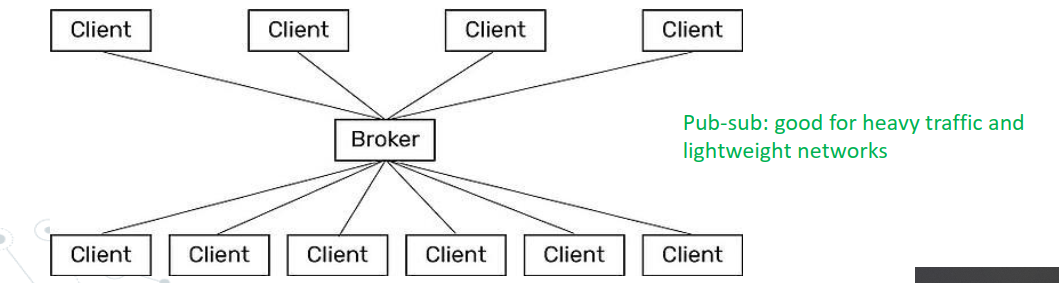
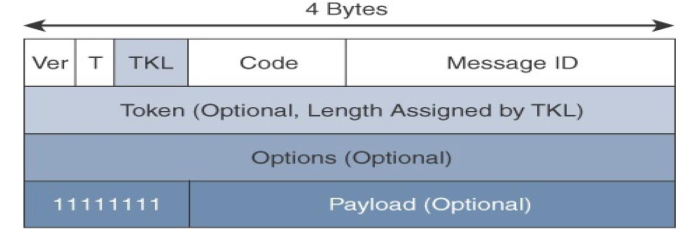
فصل 5

* چرا HTTP مناسب آیوتی نیست؟ سربار parsing زیادی داره (در ادامه توضیحات بیشتر)
* بیشتر سرویس‌های شبکه روی UDP هستن: DNS, NTP, SNMP, DHCP, VoIP
* ممکنه کار با TCP برای دیوایس‌های آیوتی سخت باشه
  + می‌خوان چند بایت کوچیک per transaction بفرستن و TCP سربار ACK و... بالایی داره
  + 20 بایت سربار هدر داره (هدر UDP فقط 8 بایته)
* مثال: فرض کنید یه سیستم دارید که با پروتکل DLMS/COSEM کار می‌کنه
  + اگه شبکه سلولاره: از TCP استفاده کنیم چون شبکه می‌تونه سربار TCP رو هندل کنه
  + اگه شبکه LLNه: UDP بهتره
* Long Association
  + در مثال قبل، می‌تونیم هزینه کانکشن رو با استفاده از روش long association، یعنی باز نگه داشتن session ها برای مدت طولانی‌تر کمتر کنیم چون سربار باز نگه داشتن session کمتر از سربار قطع و وصل کردن مکرر کانکشنه.
* سه حالت application layer protocol در کاربردهای آیوتی
  + پروتکل لایه اپلیکیشن نداشته باشیم => داده مستقیما میره به لایه ترنسپورت
    - مثال: دیوایس‌های کلاس 0، قدرت اجرای IP, UDP, TCP یا حتی پروتکل لایه اپلیکیشن ندارن، ساخت یه Protocol Stack خوب براشون غیرممکنه
    - مثلا دستگاه‌هایی که دما و رطوبت رو با پروتکل‌های LPWA LoRaWAN می‌فرستن. دو بایت دما، دو بایت رطوبت، این چهار بایت رو با LoRaWAN MAC Layer می‌فرستن و از TCP/IP استفاده نمی‌کنن
    - عیب عدم وجود پروتکل لایه اپلیکیشن: استانداردسازی و Interoperability سخت میشه. مثلا سنسورهای دمای برندهای مختلف به شیوه‌های متفاوتی دما رو گزارش می‌کنن. اگر هزاران سنسور داشته باشیم، کار با اونا سخت میشه چون اپلیکیشن‌های مختلف باید از سنسورهای مختلف داده بگیرن و بتونن بفهمنش. => راه حل: IoT Data Broker
    - IoT Data Broker: یه middleware (میان‌افزار) که خروجی سنسورها رو به یه فرمت استاندارد و قابل فهم و یکنواخت برای همه اپلیکیشن‌ها در میاره.
    - 
  + Generic web based protocols
    - روی دیوایس‌های آیوتی که توی شبکه‌های غیر Constrained کار می‌کنن زیاد دیده میشه. مثل HTTP
    - چرا HTTP مناسب آیوتی نیست؟
      * هدرهای سنگینی داره. وقتی یه سنسور می‌خواد کلا 2 بایت داده بفرسته، چرا باید 100 ها بایت هم هدر ارسال کنه؟
      * سنکرونه. کلاینت منتظر جواب سرور می‌مونه. ولی مدل سنکرون مناسب آیوتی نیست (به علت محدودیت‌های شبکه و تاخیر بالا)
      * یک‌طرفه‌ست. کلاینت باید کانکشن رو شروع کنه. توی دنیای آیوتی معمولا سنسورها خودشون کلاینتن.
      * HTTP is 1-1. Broadcast کردن سخت و پرهزینه‌ست در حالی که توی آیوتی خیلی وقتا لازم میشه.
  + IoT application layer protocols
    - برای کار روی شبکه‌های Constrained طراحی شدن و با محدودیت‌های پروتکل‌های پایینی مثل 6LoWPAN خوب کنار میان
    - MQTT: Message Queuing Telemetry Transport
    - CoAP: Constrained Application Protocol
* دو مدل تبادل داده
  + Request and Response
    - کلاینت ریکوئست یه داده خاص رو می‌فرسته، سرور با داده جواب میده
  + Publish and Subscribe
    - یه سورس اصلی به اسم Broker (یا سرور) داریم
    - کلاینت‌ها می‌تونن به بروکر داده بفرستن، یا ازش داده بگیرن، یا هر دو
    - کلاینت‌ها تنها زمانی داده می‌فرستن که داده تغییر کرده باشه => Report by Exception: RBE
    - بروکر داده برای خودش نگه‌ نمی‌داره.
  + مقایسه مدل‌ها
    - در مدل اول هر کلاینت یه کانکشن جدا به سرور باز می‌کنه و مکررا درمورد داده‌ای که لازم داره می‌پرسه
    - اگه شبکه خوب و قوی باشه، و تعداد سرورها کم باشن، مدل اول خوبه (مادامی که سرورها توانایی جواب داده به تعداد زیاد ریکوئست رو داشته باشن)
    - اما اگه تعداد سرورها و کلاینت‌ها زیاد بشه، مدل اول با مشکل ترافیک بالا مواجه میشه
    - در مدل دوم نیازی به کانکشن مستقیم، و ریکوئست‌ دادن مکرر برای گرفتن داده نیست
    - کانکشن میان کلاینت‌ها و بروکر باز می‌مونه و خیلی سبک و کم‌هزینه‌ست.
    - 
    - فقط دو نوع داده توی مدل دوم داریم:
      * دیتای تغییر یافته و آپدیت شده
      * یه Heartbeat که بروکر بفهمه کلاینت هنوز هست و وصله
    - به خاطر استفاده از RBE، ترافیک شبکه خیلی کم میشه چون داده رو هروقت تغییر میکنه میفرستیم، نه اینکه به صورت پریودیک بدون توجه به تغییر کردنش ارسالش کنیم
* پروتکل‌ها
  + CoAP
    - UDP
    - Client-Server + Publish-and-Subscribe
    - 
      * Ver: ورژن پروتکل
      * T (Type): نوع پیام
        + CON: Confirmable

ارسال reliable داده با UDP

Congestion control و time-out داره

مکانیزم Stop and wait داره (تا اک یه پکت نیاد بعدی رو ارسال نمیکنه)

پیامای تکراری رو تشخیص میده

* + - * + NON: Non-confirmable

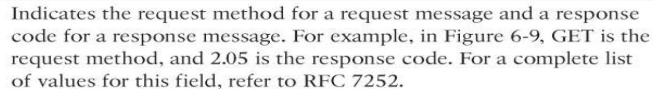
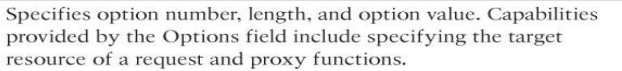
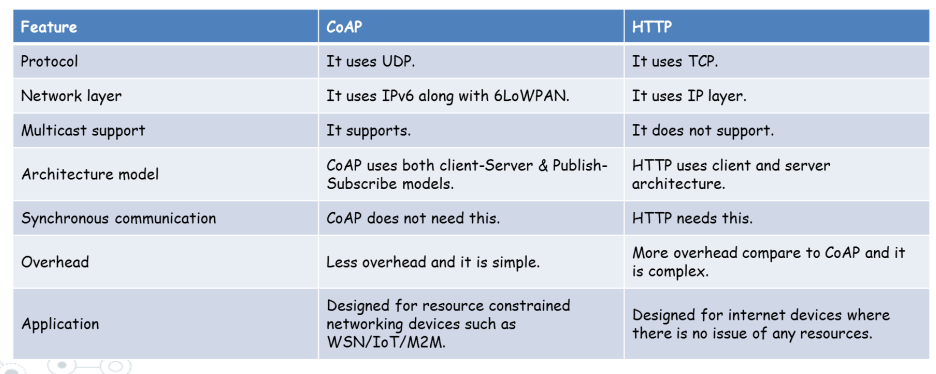
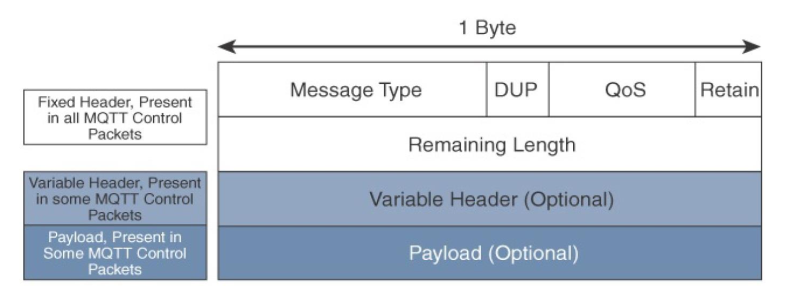
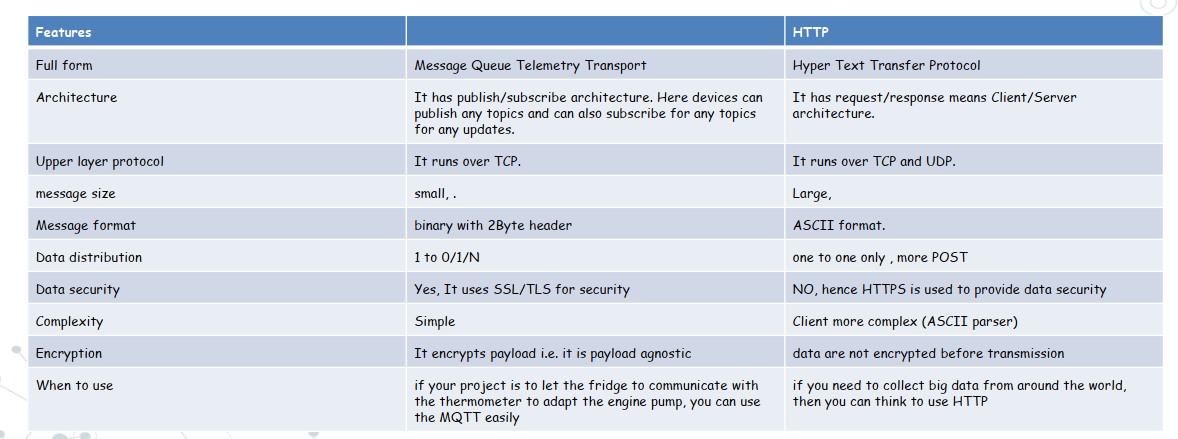
Reliable نیس

* + - * + ACK

با ارسال message ID

* + - * + RST: Reset

گیرنده وقتی یه پیام con یا non رو نمی‌تونه پردازش کنه این پیامو میفرسته

* + - * TKL: مشخص می‌کنه توکن چند بایته (0 الی 8 بایت)
      * Code: 
      * Message ID: پیام‌های تکراری رو تشخیص میده و ACK و RST رو مچ می‌کنه.
      * Token: ریکوئست‌ها و ریسپانس‌ها رو به هم مرتبط می‌کنه.
      * Options: 
      * Payload: اگه باشه، یه بایت کاملا 1 قبلش میاد که پایان سکشن Options رو مشخص کنه.
    - مقایسه CoAP با HTTP:
      * روی UDP کار میکنه
      * Latency و توان مصرفی کمتر
      * 
  + MQTT
    - TCP
    - Publish-and-Subscribe only
    - خیلی سبکه و فضایی رو اشغال نمی‌کنه. حتی دستگاه‌های خیلی ضعیف می‌تونن ازش استفاده کنن
    - شیوه بسته‌بندی داده رو مشخص نمی‌کنه
    - فرمت پیام: 
    - مقایسه MQTT با HTTP:
      * MQTT is data-centric whereas HTTP is document-centric
      * MQTT is pub-sub whereas HTTP is client-server
      * MQTT کوتاه و کوچیکه
        + فقط پیام‌های CONNECT, PUBLISH, SUBSCRIBE, UNSUBSCRIBE برای دولوپرها مهمن
      * گذردهی MQTT 93 برابره
      * MQTT سه سطح Quality of Service داره که دلیور شدن پیاما رو تضمین می‌کنن
        + At most once: همون best-effort.
        + At least once: پیام حداقل یه بار دلیور میشه.
        + Exactly once
      * 
    - مقایسه MQTT و CoAP
      * 