BACnet Core - Report

Ein Projekt der Vorlesung «Introduction to Internet and Security»

Raphael Kreft, Tim Bachmann, Nico Aebischer

17. Juli 2021

Inhaltsverzeichnis

Im Dokument «BACnet Core - Documentation» wird zusätzlich zum Core detailliert auf das BACnet selbst eingegangen. Für noch genauere Details verweisen wir auf die Dokumentation im Code selbst. Hier soll aber vor allem der BACnet Core und seine Architektur beleuchtet werden: Inwiefern handelt es sich dabei um den "Kern"des BACnets? Wie ist die modulare Struktur aufgebaut? Wie kann bei der Implementierung anderer Komponenten des BACnets den Core nutzen und von ihm profitieren? Schlussendlich gehen wir noch auf den Arbeitsprozess des Projekts ein, reflektieren diesen und geben einen Ausblick darüber, was in Zukunft mit dem Core gemacht werden kann.

1 Das «Basel Citizen Net» - Kurz und knapp

Das Internet wird mittlerweile von Milliarden von Menschen gebraucht - jeden Tag. Viele vergessen dabei, dass die Benutzung des Internets alles andere als eine Selbstverständlichkeit ist. Dies erfahren vor allem unterdrückte Minderheiten: In den letzten Jahren kam es immer wieder zu Restriktionen, wobei das Internet ganz gekappt wurde oder gezielt politische Opposition targetiert wurde.
Dies motiviert viele Menschen, sich Gedanken über Alternativen zum Internet zu machen. Wie lässt sich Kommunikation über digitale Kanäle sicherstellen, ohne dabei dem Goodwill von Regierungen und anderen grossen Akteuren ausgesetzt zu sein?
Im Rahmen der Vorlesung «Introduction to Internet and Security» wurde im Frühlingssemester 2021 die Arbeit des Vorjahres am BACnet, einem dezentralen Netzwerk, wieder aufgenommen. Ziel war es, die Funktionalität des BACnets zu erweitern. Das BACnet basiert auf Konzepten, die in «Secure Scuttlebut» implementiert wurden. Nachfolgend wird näher darauf eingegangen.

1.1 Das Secure Scuttlebutt Prinzip

Das Projekt «Secure Scuttlebutt» ist ein Versuch, Social Media als Teil eines dezentralen Netzwerks zu implementieren, bei dem Applikationen serverlos mittels abgeglichenen Append-Only-Logs funktionieren.
Die Feeds werden von Netzwerkteilnehmern lokal erstellt und bestehen aus einer Aneinanderknüpfung von Events. Jeder Event enthält Metadaten, Nutzdaten und einen Verweis auf den jeweils vorherigen Event.
Nutzer können Feeds anderer Nutzer abonnieren. Bei einem Datentransfer erhalten sie dann diejenigen Feeds lokal auf ihren Rechner, die sie abonniert haben und die vom Gegenüber zur Verfügung gestellt werden können. Somit entsteht ein dezentrales Netzwerk.
Dabei spielt es keine Rolle, auf welchem Weg die Daten zwischen den Teilnehmern übertragen werden. Es kann neben einer UDP/TCP Verbindung auch USB-Stick-Austausch (Sneakernet) oder Radiowellenübertragung genutzt werden. Durch die Freieheiten bei der Implementation der Replikation kann also sichergestellt werden, dass man die Kontrolle über das Netzwerk behält und man sich nicht von Drittparteien abhängig macht.

1.2 BACnet - aktueller Stand

Das Basel Citizen Net hat die Grundstruktur von Secure Scuttlebutt übernommen, arbeitet also auch mit Feeds und Events, welche Applikationsdaten verschiedener Netzwerkteilnehmer enthalten. Im Frühlingssemester 2020 wurde der Grundstein für die Entwicklung des BACnets gelegt.
Die Projekte waren auf die verschiedenen Komponenten des BACnets verteilt. Einige davon lassen sich mit Ergänzungen zu unserem Core kombinieren und andere implementierten Chats oder verschiedene Techniken zum Datentransfer.
Im Frühlingssemester 2021 wurde die im Vorjahr gelegte Basis weiterentwickelt. Beispiele dafür sind die Weiterentwicklung einer Chat Applikation des Vorjahrs mit einer Sketch-Funktion, verbessertes LoraLink, Verschlüsselungsmechanismen für den Chat oder das Verpacken von News Artikeln (z.B. von SRF) in Events.

2 BACnet Core

2.1 Die Problemstellung

Das BACnet lässt sich in verschiedene Module aufteilen. Dies war natürlich auch schon im Frühjahrssemester 2020 bekannt und die Gruppen wurden dementsprechend gebildet. Die Konsequenz der vielen kleineren Gruppen war dann allerdings, dass die Entwicklung des gesamten BACnets durch die Gruppierung dehomogenisiert wurde und daraus Ineffizienzen entstanden. Als Student im Frühjahrssemester 2021 war es dann zu Beginn sehr schwer, sich einen Überblick zu verschaffen. Was ist das BACnet überhaupt? Wie sind die Module strukturiert? Welche der Vorjahresgruppen liefern für mein eigenes Projekt verwertbare Vorarbeit?
Uns ist das von Beginn weg aufgefallen und wir haben uns deshalb entschieden, eine API anzubieten, welche

die zentralen Komponenten der letztjährigen Projekte auf sinnvolle Weise zusammenfasst und optimiert. Es soll einfacher werden, sich als Neuling im BACnet zurechtzufinden und mit dem Entwickeln von neuen Features zu starten.

2.2 Die Lösung

Auf der Suche nach dem «Kern» des BACnets kommt man unweigerlich zuerst auf das Storage-Modul. Es ist unumgänglich, eine Speicherarchitektur zu schaffen, welche die Datenstrukturen des BACnets effizient abspeichern kann. Daher besteht der neue «Core» zu einem grossen Teil aus speicherbezogenen Klassen.
Der Core muss dazu imstande sein, Events zu erstellen und effizient in Feeds zu organisieren. Diese müssen so verarbeitet werden können, dass man sie im permanenten Speicher (Datenbank) ablegen sowie aus Datenbankeinträgen wiederherstellen kann.
Der Kern muss auch den geregelten Anschluss seinerselbst an das Netzwerk sicherstellen. Die Replikationsschicht des BACnets kann über verschiedene Channels via einen Communication-Link mit dem BACnet Core in Verbindung treten und Datenaustausch vornehmen.
Schlussendlich braucht es auch noch Security-Aspekte. Es muss möglich sein, die Events und Feeds zu verifizieren und deren Authentizität zu prüfen.
All diese Funktionalitäten waren bislang auf verschiedene Projekte verteilt und der Überblick hat gefehlt. Mit dem Core ist es unser Ziel, das Erstellen, Synchronisieren und Verwalten von Nodes im Netzwerk einfach zu machen. Die Schwierigkeit für nachfolgende Gruppen soll nicht darin liegen, sich um das Erstellen von Events und um die Synchronisation zu kümmern, sondern sie sollen sich vollends auf die Implementation ihres eigenen Projekts fokussieren können.

3 Aufbau des Cores

In diesem Abschnitt geht es um den modularisierten Aufbau des Cores. Mit der Modularisierung zielen wir darauf ab, dass nachfolgende Gruppen auf einfache Weise ihre Projekte mit dem Core verbinden können und dass allfällige Änderungen innerhalb eines Moduls gemacht werden können und nicht den ganzen Core betreffen. Im folgenden betrachten wir die Module des BACnet Cores.

3.1 Core Interface

Das Modul «Core Interface» dient der Organisation von Daten als Teil des BACnet Frameworks, strukturiert sie also in Events und Feeds. Das Modul ist gemäss Abbildung 1 aufgebaut.

3.1.1 Events

Nutzdaten von Applikationen werden als Events auf der Datenbankinstanz der Node abgespeichert. Um Events zu versenden werden sie in ein Binärformat (CBOR) gebracht, dass sich zur Serialisierung und damit zum Datentransfer zwischen den Nodes eignet. Events bestehen aus Metadaten, einer Signatur und den Nutzdaten.
Für den genauen inneren Aufbau von Events verweisen wir auf den Code oder die detailliertere Dokumentation des Projekts.
Die wichtigsten Funktionen der Events sind jeweils getter-Methoden um die Meta- oder Nutzdaten zu bekommen sowie zwei Methoden, um Events in das CBOR-Format zu konvertieren bzw. von einem CBOR-Objekt einen Event zu erstellen.

3.1.2 Feeds

Feeds sind eine Aneinanderkettung von Events. Mithilfe eines Hashwertes wird jeweils auf den vorangehenden Event referenziert. Es sind Append-Only-Logs und jeder Feed kann durch eine eindeutige Feed ID identifiziert werden. Wie auch Events enthält jeder Feed Metadaten. Wir unterteilen Feeds in die Klassen Owned und Subscribed. Innerhalb der beiden Klassen wird jeweils noch in Master- und Content Feed unterschieden.

Owned Feeds einer Node sind diejenigen, die von der Node selbst erstellt wurden. Subscribed Feeds sind dementsprechend Feeds, die von anderen Nodes stammen, aber trotzdem in der Datenbank einer anderen Node gespeichert sind.
Zu jeder Node gehört immer auch ein Masterfeed, der bei der Initialisierung der Datenbank erstellt wird. Für jeden neuen Content Feed wird dann auf dem Masterfeed ein Event angehängt. Damit wird die Zugehörigkeit des

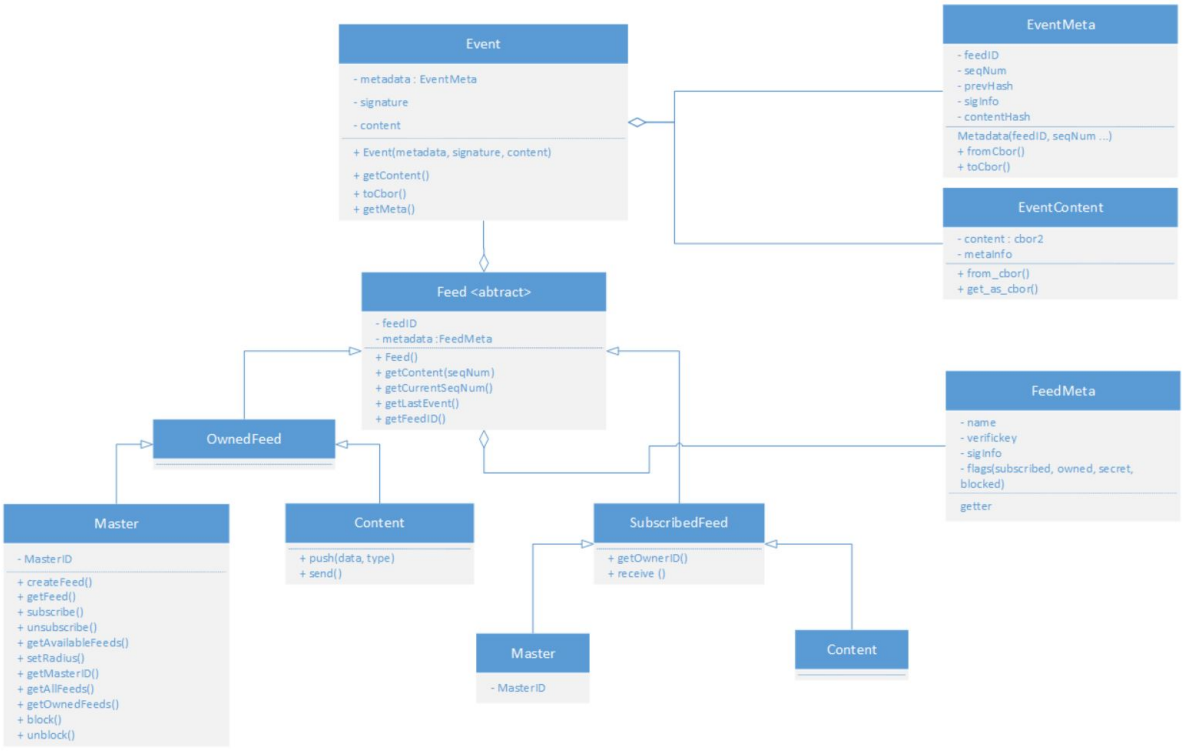

Abbildung 1: Aufbau des Core Interface Moduls.

Feeds zur Node angegeben. Der jeweils erste Event eines Content Feeds verweist auf den Masterfeed, auf dem er erstellt wurde. Damit wird vermieden, dass ein Masterfeed einen Feed zu unrecht als den eigenen beansprucht.
Der zentrale Unterschied zwischen Master-und Contentfeeds ist, dass die Masterfeeds standardmässig immmer synchronisiert werden. Aus der Sammlung der Masterfeeds auf einer Node kann dann eine Liste von Trusted und Untrusted Feeds gemacht werden.
Die Content Feeds (Owned und Subscribed) dienen zur Eingliederung von Applikationsdaten in die Datenstruktur des BACnets. Meist werden applikationsspezifisch Feeds angelegt, denn bei der Auswertung der Daten muss ja erkannt werden, wie die Nutzdaten interpretiert werden sollen.

3.2 Storage

Das Modul «Storage» implementiert die Logik für das geregelte Abspeichern der Daten auf einer Node. Jede Node beinhaltet eine Datenbankinstanz. Da auch hier ein modularisiertes System erwünscht ist, soll der Zugriff der Node auf die Datenbankinstanz unabhängig von der gewählten Datenbanksprache sein. Es soll also möglich sein, die Sprache zu wechseln und dabei möglichst wenig im bestehenden Code anzupassen, sondern nur neuen Code passend auf das Interface, auf das die Node zugreift, zu schreiben.

3.2.1 Storage-Control

Die Storage-Control ist als abstrakte Klasse präsent und dient als Grundlage für die Implementierung verschiedener Möglichkeiten zur Datenbankanbindung. Die Aufgabe der Storage-Control ist es, die in allen Speichervarianten benötigten Funktionalitäten vorzugeben. Dazu gehören zum Beispiel Methoden um neue Feeds zu erstellen, Feeds zu blockieren, Änderungen zu pushen oder Events zu importieren. Diese High-Level-Funktionen müssen dann im Detail vom Database-Handler und vom SQLite Controller ermöglicht werden.

3.2.2 Database-Handler

Der Database-Handler wird von der Storage-Control aus «gesteuert» und nimmt sich der Low-Level Implementation der datenbankspezifischen Funktionen an. Der Code ist allerdings immer noch nicht auf eine Datenbankanbindung spezialisiert, dies wird im SQLite Controller gemacht.

3.2.3 SQLite Controller

Der SQLite Controller ist die vom BACnet Core spezifizierte Speichervariante. Es wird mit einer SQLite Datenbank gearbeitet.
Da die Storage-Control allgemein gehalten ist, wird es auch für zukünftige User des BACnet Cores möglich sein, auf einfache Art und Weise Storage-Anbindungen zu machen. Die Idee ist es, dass man nicht auf eine SQLite Anbindung limitiert ist, sondern je nach Bedarf z.B. auch noch eine MySQL Anbindung oder weitere Varianten hinzufügen kann.

3.2.4 Event Factory

Die Event Factory ist wie der Name schon sagt dazu da, Events (aber auch Feeds) zu erstellen. Sie wurde zu einem grossen Teil von der Vorjahresgruppe «LogMerge» übernommen. Beim Erstellen von Events und Feeds müssen zusätzlich zum Content noch einige weitere Informationen mitgegeben werden, wie zum Beispiel die Feed ID und die letzte Sequence Number, um den Event im Feed einzugliedern, die Signature Info oder einen Content Identifier, der für die Unterscheidung des Contents in die verschiedenen Apps zuständig ist.

3.3 Security

Das Modul «Security» behandelt die Sicherheit im BACnet. Es muss eine Kontrollstruktur vorhanden sein, die einerseits überprüft, ob Feeds überhaupt in die Datenbank aufgenommen werden sollen und andererseits die Integrität von Feeds garantieren kann.
Dafür gibt es verschieden Kriterien für den Im- bzw. den Export. Dieses Set von Regeln stellt sicher, dass keine Feeds ausserhalb des gewünschten Radius auf der eigenen Node zu liegen kommen und dass blockierte Feeds nicht importiert werden. Zusätzlich dazu gibt es verschiedene Funktionen, mit denen man die Hashwerte des Contents und Signaturen überprüfen kann.

3.4 Replication

Das Modul «Replication» nimmt sich der Verbreitung von Feeds und Events über verschiedene Netzwerkteilnehmer an. Jede Node ist Teil des dezentralisierten BACnets und muss über mindestens einen Kommunikationslink mit anderen Teilnehmern kommunizieren können, damit die Applikationsdaten über das Netz verteilt werden.
Ein Synchronisationsprotokoll legt fest, welche Methoden (z.B im Storage Modul) aufgerufen werden sollen. Die Protokollinstruktion wird zusammen mit den Daten in einem Messagecontainer abgelegt und in dieser Form in das CBOR-Format transformiert respektive von einer CBOR Datei in diese Form gebracht.

3.4.1 COM-Link

Der Hauptbestandteil ist das Zusammenspiel des Communication-Links mit verschiedenen Channels. Der Communication-Link ist als Instanz auf der Node enthalten und agiert als Mediator, der Funktionen zum Datenaustausch gegen aussen anbietet. Beispiele dafür sind *importMasters()*, *exportAll()* oder *autoSync(timeInterval)*.
Der COM-Link wird von der Node aus angesteuert und kommuniziert dann mit der Storage-Control (SQLite Controller), wo die logisch «höheren» Anfragen zu einzelnen Datenbankabfragen verarbeitet werden.

3.4.2 Channels

Die Channels sind das Verbindungsstück des BACnet Cores nach aussen. Die abstrakte Klasse «Channel» legt fest, was bei einer spezifischen Implementierung sicher vorhanden sein muss. Hier ist ebenfalls der modulare Charakter der Architektur von grosser Bedeutung, damit weitere Channels einfach hinzugefügt werden können, ohne am Core selbst was ändern zu müssen.
Unser Core bietet schon eine spezifische Implementation eines UDP-Channels an, wobei der Datentransfer über eine UDP-Verbindung funktioniert. Wer den Core für ein anderes Projekt benutzt, kann dann auf einfache Weise und anhand des Beispiels mit UDP einen eigenen Channel implementieren, der ein anderes Transportmedium unterstützt.

3.5 Samples

Das Modul «Samples» enthält Beispielcode, anhand dessen man sehen kann, wie der Core gebraucht werden kann. Die Beispiele waren erst primär für uns selbst gedacht, um unseren Code zu testen. Allerdings entschlossen wir uns dazu, die Samples im Projekt zu lassen, damit mögliche Nutzer des Cores einen einfacheren Einstieg haben.

4 Arbeitsprozess

Die Arbeit am BACnet klang von Beginn weg verlockend. Die Idee, ein von Studenten entwickeltes dezentrales Netzwerk in Basel aufzuziehen fand grossen Anklang bei uns. Nach einem ersten Blick auf die Projekte vom letzten Jahr war uns dann aber schnell klar, dass es sehr anspruchsvoll werden würde.
Die erste grosse Hürde bestand darin, sich einen Überblick zu schaffen. Es gab keine zentrale Anlaufsstelle, bei der man sich informieren konnte, sondern ein gutes Duzend Gruppen, die jeweils einen kleinen Teil des BACnets abdeckten.
Nach anfänglichen Schwierigkeiten entschlossen wir uns bald dazu, unsere Probleme mit dem Verständnis des Ganzen auszunutzen, indem wir genau das zu unserem Projekt machen. Unser grosses Ziel war es, einen verständlichen Einstieg ins BACnet zu ermöglichen und gleichzeitig die zentralsten Core-Funktionalitäten anzubieten, sodass es zukünftigen Gruppen einfacher fällt, mit ihrer Arbeit zu beginnen.

Zunächst hatten wir vor, die Arbeit am Projekt mit einem Bottom-Up Approach anzugehen. Wir wollten uns ausgehend von den anderen Projekten überlegen, was für Funktionalitäten ein BACnet Core anbieten muss. So wäre es möglich, sehr gezielt für die anderen einen Kern zu programmieren, den sie für ihre eigenen Arbeiten nutzen könnten. Wir haben diese Angehensweise dann aber doch verworfen und uns an einen Top-Down Approach gemacht. Unser oberstes Ziel war es, Ordnung in die etwas chaotische Struktur der Projekte zu bringen. Daher wäre es wohl nicht ratsam gewesen, auf Stufe der Projekte zu beginnen, die schon sehr spezifisch sind und allenfalls abweichende Anforderungen stellen könnten.
Wir hatten eine relativ lange Planungsphase, in der wir ausschliesslich durch Diskussionen im Plenum und dem Erstellen und Verwerfen von Diagrammen gearbeitet haben. Obwohl wir grosse Teile vom Code des letzten Jahres übernehmen konnten, war es uns sehr wichtig, diesen stärker zu modularisieren und in einer wohldefinierten Struktur zu organisieren.

Bei der Implementation haben wir zunächst einmal wichtige Bestandteile wie Klassen für Events, Feeds etc. kopiert. Danach haben wir die restlichen Teile zuerst als «Skeleton» geschrieben. Bei dem Prozess ergaben sich dann noch einige Änderungen. Wir planten ursprünglich, mit einem Eventbus Nachrichten zwischen den Instanzen hin- und herzuschicken. So sollte die Kommunikation zwischen den Feeds, der Datenbankinstanz und dem Kommunikationslink einfach funktionieren. Wie wir allerdings bemerkten, brachte die Verwendung des von uns gewählten Eventbusses mehr Aufwand als Nutzen. Für diverse Nachrichtentypen hätten wir entsprechende Klassen definieren müssen und es hätte auch die Anzahl der Methoden im Projekt unnötig gross gemacht. Deshalb arbeiten wir jetzt einfach klassisch mit dem Übergeben von Instanzen (z.B. der Datenbank) an andere Objekte, worüber wir die Methoden dann einfach direkt aufrufen können.
Dank der genauen Planung war es uns dann möglich, die Implementation zielgerichtet und zügig durchzuführen.

5 Reflexion und Ausblick

Rückblickend war die Arbeit am BACnet eine lehrreiche und unterhaltsame Aufgabe. Es war durchaus anspruchsvoll, sich in die Projekte des Vorjahres einzulesen und daraus Schlüsse für das eigene Projekt zu ziehen. Allerdings war das auch sicherlich eine gute Übung, da es sehr wahrscheinlich ist, dass wir wieder einmal auf ein Projekt stossen werden, bei dem erst einmal viel Recherche und Codebasis-Analyse ansteht.
Wir haben das Gefühl, dass wir mit diesem Projekt sehr zielgerichtet gearbeitet haben, so wie wir es in «Software Engineering» gelernt haben. Zuerst kam eine lange und detailreiche Planungsphase. Danach haben wir das geplante vorerst als Prototyp mit Skeleton-Methoden implementiert und die Struktur nochmals angepasst, bevor wir dann die genaue Implementation begonnen haben.

Unser grosses Ziel mit dem BACnet Core war es, das Verständnis für das BACnet und die zentralen Aspekte davon zu verstärken. Es soll in Zukunft nicht mehr so schwierig sein, sich in die Funktionsweise des Ganzen einzulesen und sich die nötige Architektur für Erweiterungen zusammenzusuchen. Der Core sollte die nötigsten Funktionalitäten zum Start bieten und der modulare Charakter macht es möglich, in sinnvoller Frist Anpassungen für das eigene Projekt zu machen.
Wir hoffen, mit der Arbeit am BACnet Core den Weg für zukünftige Gruppen geebnet zu haben und damit unseren Beitrag für das Ergänzen des BACnets mit weiteren Funktionalitäten leisten konnten.

6 Verwendete Bibliotheken

Im Projekt wurden die folgenden externen Bibliotheken verwendet:

- cbor2 `https://pypi.org/project/cbor2/`
- hashlib `https://docs.python.org/3/library/hashlib.html`
- secrets `https://docs.python.org/3/library/secrets.html`
- nacl `https://pypi.org/project/PyNaCl/`
- contextlib `https://docs.python.org/3/library/contextlib.html`
- sqlalchemy `https://www.sqlalchemy.org/`
- enum `https://docs.python.org/3/library/enum.html`
- queue `https://docs.python.org/3/library/queue.html`
- abc `https://docs.python.org/3/library/abc.html`