Group 13: Sneakernet
Project Report*

Leonhard Badenberg, Patrik Bütler, Luka Obser

July 19th 2020

Abstract

Through this project we provide an interface to create and use sneakernets in the BACnet as a mode of data distribution. As a fundamentally different approach to data distribution sneakernets offer a way of transferring massive amounts of data in a highly secure way at the cost of latency, when compared to classical computer networks such as the Internet. While the potentially high throughput of sneakernets is an argument for them, we see the main advantage of our service in the type of security it offers the BACnet and its participants.

*This document will focus on reporting the development process of our service and how it functions. The results themselves can be found here: https://github.com/cn-uofbasel/BACnet/tree/master/groups/13

1 Introduction

As a decentralized network the BACnet (Basel Citizens Net) aims to offer telecommunication channels that are not dependant on the global Internet. The vision of this project was to create a software that would allow a layperson to take part in the BACnet through a sneakernet. Sneakernets are a mode of data distribution in which data exchange between nodes is achieved by loading content on external storage media, such as USB drives, and delivering those to the recipient.
This means that our software must give the user the ability to extract and integrate packets from and to their local representation of the BACnet. Furthermore, to be layperson friendly, we would need an automatic way of determining which packets should be extracted rather than letting the user choose feeds and events individually. We chose to adapt a design philosophy that restricted user interaction with the extraction process. This led to a rather tedious initiation handshake when adding new users to the sneakernet as we do not offer an explicit way to prepare the storage medium, which is the heart of a sneakernet, for this - but more on that later.

2 Background

Sneakernets have both advantages and disadvantages in comparison to better known computer networks such as the Internet. The main differences are:

- Latency:
  Sneakernets rely on old-fashioned methods of transportation such as a person, drone or trained animal moving the, quite literal, packet between nodess. This means that sneakernets, when compared to mainstream transmission media in computer networks as seen in the lecture (coax, optical fiber, radio, etc.), take significantly longer to transmit a packet.

- Bandwidth:
  Rather than relying on preexisting cables or the air as transmission medium we use data storage devices such as USB drives, external hard drives or CDs which allow us to send data in bulk rather than only a few bits at a time. Formally this means that the size of the packet transported can be potentially much larger allowing higher bandwidth and, consequently, potentially higher throughput in corner cases despite high latency.

- Security:
  Sneakernets are usually a decentralized channel operated only by a small group of users. This has the advantage that some network security issues such as authentication can be outsourced to face-to-face human interactions. A sneakernet, in which the storage medium is handled with care, deliveries of the medium are made personally and, most importantly, the users trust each other, does not leave much space for man-in-the-middle or sniffing attacks.

Sneakernets have many contemporary practical applications. Amazon offers a service in which a truck loaded with 100 petabytes of storage picks up the clients data to upload it into their Amazon S3 cloud storage service.[1] The Cuban black market runs a sneakernet called "El Paquete" as a substitute for the global Internet to which Cuba has little access.[2]
But not only the potentially higher throughput is why sneakernets are chosen over computer networks. Security concerned operations such as military ones also rely on sneakernets. An example for this would be the sneakernet used by al-Qaida leader Osama bin Laden to transport emails to Internet cafes before sending them to stay off the grid yet connected.[3]

3 Tasks and Challenges

3.1 Tasks

The development of our service consisted of three main tasks. Firstly, to build something that can be conveniently used by BACnet users that do not have a computer science background. The design process that led to our final product is described below. The second, and most fundamental, task was to figure out a way to extract and integrate packets from and to the local database representations of the BACnet. For this we were able to use group 4's API[4] which exported to and imported from *.pcap* files. The last task in this project, and focus of our computer scientific work, was the development of an extraction algorithm that could determine which packets from a user's local BACnet database were relevant to the other users of the sneakernet instance. A goal we set ourselves was to keep the number of packets stored on the storage medium as low as possible at any time. A conceptual description of said extraction algorithm will be provided below.
A large question was whether we should offer onboarding to the BACnet. After consideration we decided that we, as a transport layer service, should not be directly involved in this. Nonetheless, we still had to handle onboarding in the scope of our project as the set of users using a sneakernet is dynamic.

3.2 Challenges

Due to the hierarchical structure of the BACnet[5] we were dependant on interfaces provided by groups responsible for merging (group 4) and storing (group 7) logs. Group 14s feed control was also identified as a potential source of conflict when combined with our extraction algorithm. Ultimately we decided to limit

[1]https://aws.amazon.com/de/snowmobile/
[2]https://en.wikipedia.org/wiki/El_Paquete_Semanal
[3]https://www.washingtontimes.com/news/2011/may/13/how-bin-laden-emailed-without-being-detected-us/
[4]https://github.com/cn-uofbasel/BACnet/tree/master/groups/04-logMerge
[5]https://github.com/cn-uofbasel/BACnet/blob/master/doc/arch.md

interactions with databases to the usage of group 4s API which eased the workload for us and higher up groups such as 7 and 14. Collaboration with group 4 allowed us to specify crucial aspects of their provided interface. Together we decided upon meaningful method parameters and how to handle basic issues such as specifying a maximum amount of events to be kept on the storage media.
Another challenge from this hierarchical structure was that development had to take place without dependencies being fully implemented yet. This led to a long collaborative debugging process before connected groups could demonstrate their results.
The last major challenge we faced was when, due to hierarchical limitations we failed to forsee, we could not run our software from the storage medium. We planned on compiling our program into an executable that would be located on the storage medium for ease of use but we failed to adapt to this challenge fast enough. There could have been a solution but due to the nature of such a project we had to accept that the environment around us is equally evolving up until the due date.

4 Division of Labor

We assigned three main responsibilities to each other:

- Leonhard was responsible for the front end and thus heavily involved in the aforementioned first task
- Patrik was responsible for the back end and the implementation of the extraction algorithm
- Luka was responsible for organizational matters such as documentation of progress and communication with other groups.

These roles aimed to provide us, as a group, with an overview over the state of any part of the project to allow us to plan ahead, set priorities and communicate about open tasks. Naturally everyone also helped with what needed most work during the project. Report notes were mostly brainstormed in collaboration and the writing was split up based on sections.

5 Design choices

We wanted to keep the design of the GUI simple to allow uncomplicated use by laypeople. We started with buttons for import and export, and a way to set the path to your local BACnet representation. When we changed our software to run from the directory containing the local BACnet representation, this was changed to ask for a path to the storage medium instead.
To distinguish between different devices with different states of their feed, we needed unique ids. The initial thought of automatically assigning unique IDs to users was quickly discarded. Instead, we decided on using usernames. This

also helped with creating multiple users on the same device which was helpful for testing due to the current pandemic.
As the implementation evolved over time a few of the initial GUI elements became obsolete. We moved selecting the path and username to the beginning of the GUI and added a quick check for our LogMerge dependency[6]. We also merged import and export button into just one update button as our resource management was not refined enough to justify a separation.

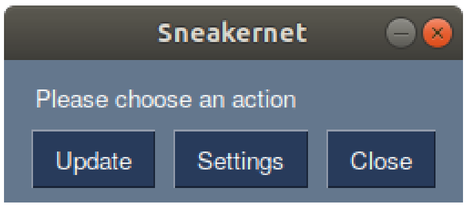

Figure 1: Final mainframe of the GUI

6 Implementation

6.1 Extraction algorithm

We worked with an abstract representation of feeds and events in the form of dictionaries as agreed on with group 4. The result was a sneakernet dictionary which saved a user's name and a second "user" dictionary containing the respective user's database state. The user dictionary contains a list of all feeds a user has saved and their respective most recent sequence number. This is being updated whenever we import or export events and is saved on the storage medium.

The main part of back end work was to work with those dictionaries and make sure they work properly, as well as updating them after exporting and importing. Before we dive into specifics lets examine what has to happen when a user updates their BACnet database using our sneakernet service.

First off, we request group 4 to import all events that are currently present on the connected storage medium. Since group 14's service filters out duplicates and only really imports relevant and valid events into the database we don't have to look into what we are trying to import. In the next step we now delete all stored pcap files present on our storage medium and request an export of all events not owned by at least one user of the sneakernet. Here group 14 filters again what is dumped on the storage medium so that we don't have to worry about allowing exporting private packets that have no authorized recipient.

[6]https://github.com/cn-uofbasel/BACnet/blob/master/groups/04-logMerge/logMerge/README.md

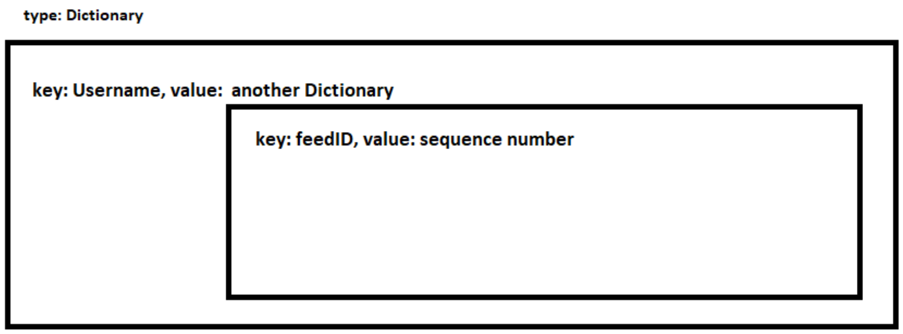
type: Dictionary
key: Username, value: another Dictionary
key: feedID, value: sequence number

Figure 2: Abstract information about every user

To save information in between sessions we dump the sneakernet dictionary into a text file with the following template

```
user_1;feedID_1:seqNum_1,feedID_2:seqNum_2, ... +user_2;feedID_1:seqNum_1, ...
```

where every user of the same device is encoded alongside all the feeds and the latest sequence number that this person had access to the last time a synchronization with the sneakernet took place. If one user didn't get an update for a longer period every event that wasn't received yet will stay on the storage device until every requesting user has received it.

6.2 Onboarding

New participants automatically request all feeds of other users on the same device and vice versa, all existing users are requesting the new person's feeds. The naive assumption here is that every user of the sneakernet is requesting all of each others feeds.

Security is given by group 14's integration in group 4's API. Feeds that have no recipient will never be exported. Feeds that a user has not subscribed to will never be imported. This still leaves events on the storage medium that certain users of the sneakernet should not have access to but this is unproblematic due to our aforementioned assumption of trust.

7 Results

While we could expand our tool to have a lot more functionality by offering the users more freedom, we think that our final product fulfills the promise of

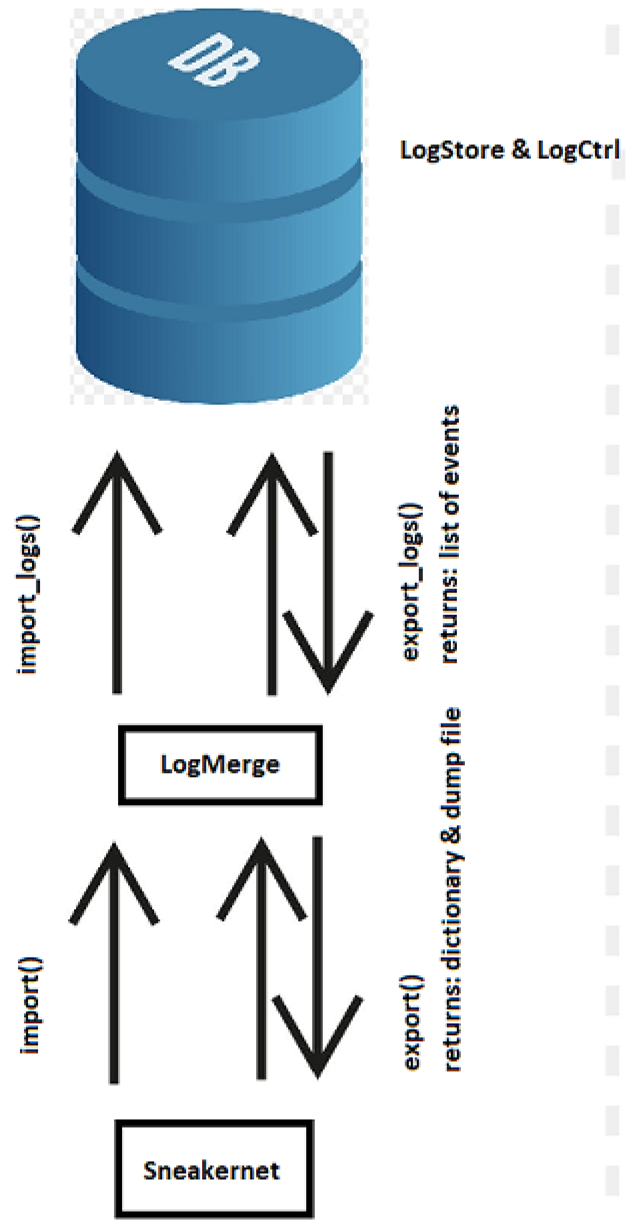

Figure 3: How we are using LogMerge to access the database

data distribution and synchronization through a sneakernet. As for resource management, improvements could be made as the update process described

above naively deletes all stored pcap files before replacing them with partially identical ones. The way the meta dictionary is encoded also allows for refinement to decrease its size.
It is important to note that we only provide the ability to prepare and evaluate storage devices as sneakernet transmission media. This means that if you join with the goal of getting an up-to-date representation of the entire BACnet it could, in theory, be impossible to achieve that since the decentralized nature of sneakernets requires collaboration with co-users. That way, our service can not be considered "on demand" since requests for wanted events need to be sent out before the user can expect to get them.
A recorded demo showcasing the work of group 3, 4, 7, 14 and ourselves can be found on the BACnet's repository.[7]

8 Final words

This project showed us how effective separation of concerns can be when it comes to dealing with hierarchy in larger projects. Limiting our interactions with the BACnet representations through group 4 allowed us to stay flexible enough so that we could adapt to changes that happened on their layer without causing many new problems for us. We saved us and other groups time by not communicating with groups that don't affect us directly. Staying flexible also allowed us to quickly adapt our plans, when we had to change the place our software was executed from.

We would like to take this chance to thank everyone who contributed alongside us to the BACnet and anyone who might do so in the future. We hope you could learn as much from this experience as we did and wish you good luck.

[7]https://github.com/cn-uofbasel/BACnet/raw/master/groups/Demo_D/Demo_D_Procedure.mp4