Budapest University of Technology and Economics
Faculty of Electrical Engineering and Informatics
Department of Telecommunications and Media Informatics

Zoltán Csiszár

PROTOCOL TRANSLATION SUPPORTED BY MACHINE LEARNING TOOLS

SUPERVISOR

Dr. Pál Varga

BUDAPEST, 2023

Table of Contents

Abstract

In the rapidly changing ICT environment, the number of protocol standards and their versions keep increasing. The interoperability of communicating devices can pose a challenging obstacle due to the diversity of protocols and their different versions. The unclear mapping of protocols and the multitude of implementation variants make their translation a largely unresolved issue to date. Although there are translators and proposed solutions for easier translation between certain protocols, they either are unique, non-scalable solutions or generalized theoretical concepts with practical challenges.

This paper starts with a summary of the current artificial intelligence-based translation solutions. The core of my study revolves around the methodological description of machine learning-based protocol translation, supported by a tool chain, and demonstrated through practical case verifications. In a heterogeneous environment, translating between various types of devices poses additional challenges. I also illustrate how various protocol translators can be integrated into modern, service-based architectures.

The feasibility study results of machine learning-based translation are primarily presented through the translation between HTTP and CoAP protocols. Fine-tuning and application of machine learning-based natural language models, training existing neural models, and creating my own neural models are also part of the methods I studied. I also deal with the preprocessing and postprocessing of translatable protocol structures to enhance the translation efficiency of different models.

Since the data structures and protocols often contain confidential information, it may not be possible to use live protocol content for machine learning. To overcome this, in this paper, I also address how a large training dataset can be generated using generative AI without having to use confidential information.

1 Introduction

Due to the ever-increasing number of communicating systems with different protocols and data structures in use, there is a great demand for new translation methods that can be used to translate between them.

This is especially true in industry 4.0 and 5.0 system-of-systems, where the number of vendors and their legacy protocols and data structures are overwhelmingly high. Standardization is a desirable solution, although not practical in a domain, where vendor-specific data structures and protocols are still there in systems operating over 20 years – and new systems have to communicate with them. The solution in these cases – as well as in the interoperability with newly created data structure and protocol version possibilities – is using translators.

This is such a valid need and great challenge in industry 4.0 that the EU have initiated multinational project of Key Digital Technologies on this, called Arrowhead flexible Production Value Networks (fPVN) [1].

Since the number of direct translators needed is proportional to the square of the number of used languages, coding translators for every protocol-pairs used is simply unscalable, and unaffordable. Therefore, the demand to overcome this problem has resulted in many studies researching possible solutions, although most of them are purely theoretical and still require many experimenting, and implementation to see if they are feasible for live industrial use. Some studies suggest a common middle language for each protocol – such as English is used as a common meta-level translation base for natural languages [2]. This would result in the number of translators needed being proportional to the number of protocols thus significantly reducing the work that has to be done. One problem with this proposed solution is that, due to the many various types of protocols, it is questionable if there is a common language that can accurately represent the exact translation of each and every one of them, possibly resulting in inaccurate translations.

The solution that I suggest and analyze through this paper is the use of machine learning in protocol translation. This is actually one of the approaches under analysis by the above-mentioned EU project, Arrowhead fPVN. My work is a contribution to this project's efforts as well.

Due to the rapidly improving machine learning tools, neural networks could provide a feasible solution – even if translation precision is not 100%. This is indeed a slippery slope: how accurate should a Machine-to-Machine (M2M) translation be? As human language-to-language translations are often far from 100% precision (due to semantic and contextual reasons), maybe we cannot expect 100% translation precision from M2M translation? In these cases, we should build fault tolerant systems that have the robustness against such imperfections. On the other hand, in mission critical systems, 100% translation accuracy is a must. The current paper does not try to define the precision levels for the use-cases at all – but based on the Arrowhead fPVN project proposition, it suggests that in future, high-diversity system-of-systems the requirement for translation accuracy can be below 100% in order to enable functional interoperability in the overall system-of-systems.

Another disclaimer for the paper is the chosen data formats. XML-JSON translation is highly feasible with algorithmic data format mapping procedures – so we can say it is useless to apply machine learning for this task as if won't be effective. This is true – even for XML-SenML, or CoAP-HTTP translations. But these translation use-cases are chosen for methodological point of view, to validate the feasibility of the translation method, and see how precise the current tools could be with well-known (algorithmically comparable) results. We will not be able to have such a comparison between all the data models, so we could see these use-cases as benchmarks for the toolset – and this toolset will get improved over time; after which we can apply it to much less obviously known data format and protocol translation tasks.

2 Protocol translation approaches

The overall task here is not only to translate from one protocol to another, but potentially to translate between data formats and even between data models. This is illustrated by Figure 1. In the Figure the notations are used based on the Service Oriented Architectures (SOA) concept, where the communicating parties are information (service) providers and consumers. A well-known variant of the SOA approach is the microservice-based design, used overall in modern distributed systems. Recently, SOA is used in industrial IoT systems of system designs, one of the practical examples being the Eclipse Arrowhead framework [3]. Figure 1 is an illustration in the current deliverable of the Arrowhead fPVN project [4], discussing recent advances of the framework, including data and protocol translation.

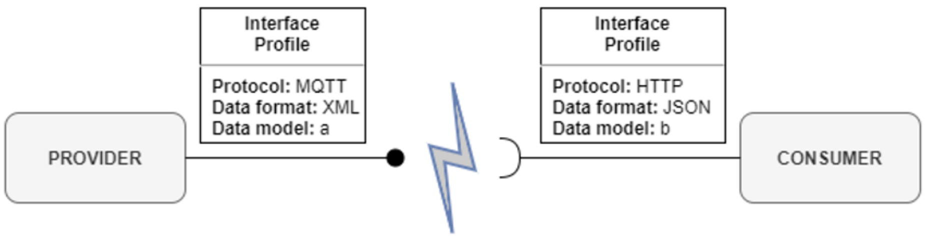


Figure 1. Translation needs between communicating parties - on the level of protocols, data formats and data models [4]

2.1 Translation architectures

The main idea is that the translator is situated between the two parties that communicate, and it executes the translation task in the data exchange path as seamlessly as possible. A simple architectural example is that of NAT, Network Address Translators – first standardized in 1994 in IETF RFC 1631, obsoleted by IETF RFC 3022 in 2001, which is still in active existence [5].

A similar a setup is shown by Figure 2, where the “Translator hub” is situated between the CoAP and MQTT infrastructures [6].

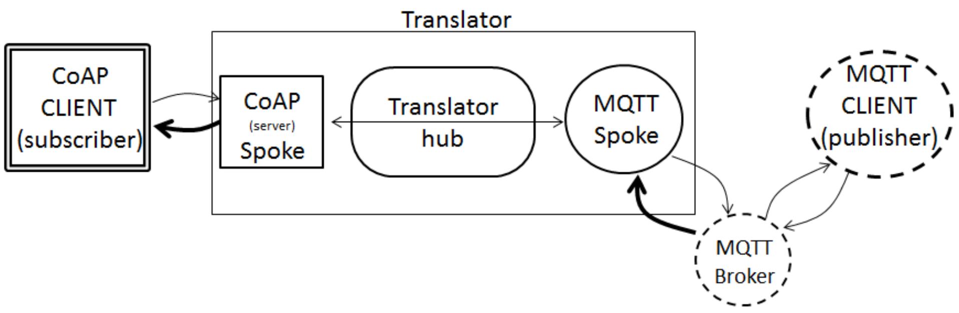
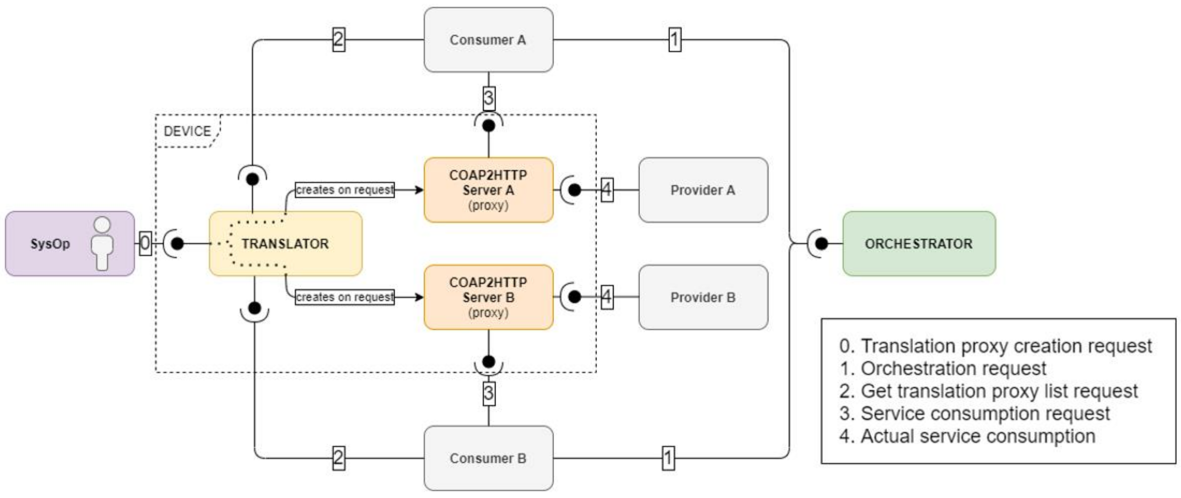


Figure 2. Translator in the data path, between conceptually different infrastructures: CoAP is Request-Response, MQTT is Publish-Subscribe type [6]

If the translator is not statically deployed between the parties, but its resources are dynamically associated with sessions in the multi-system architecture (a.k.a. System of Systems), the control gets somewhat more complex. This is shown in Figure 3, where the translator is used in a SOA setup, and the Orchestrator helps in the matchmaking process between provider and consumer systems [3]. As Figure 3 suggests, the Translator system (as part of the translation device) can instantiate dedicated protocol proxies between the given systems (provider and consumer), hence overcoming potential translation resource issues. This exact approach complies with the current reference implementation (version 4.6) of the Eclipse Arrowhead concept [7].



Consumer A
DEVICE
SysOp
TRANSLATOR
creates on request
COAP2HTTP Server A (proxy)
COAP2HTTP Server B (proxy)
Provider A
Provider B
ORCHESTRATOR
Consumer B
0
1
2
3
4
0. Translation proxy creation request
1. Orchestration request
2. Get translation proxy list request
3. Service consumption request
4. Actual service consumption

Figure 3. Translation architecture in Eclipse Arrowhead v4.6 (current) [4]

This concept is being further improved in version 5 of the Eclipse Arrowhead framework, as shown by Figure 4. This approach takes into account data model translation as a potentially independently configurable element of protocol translation [4].

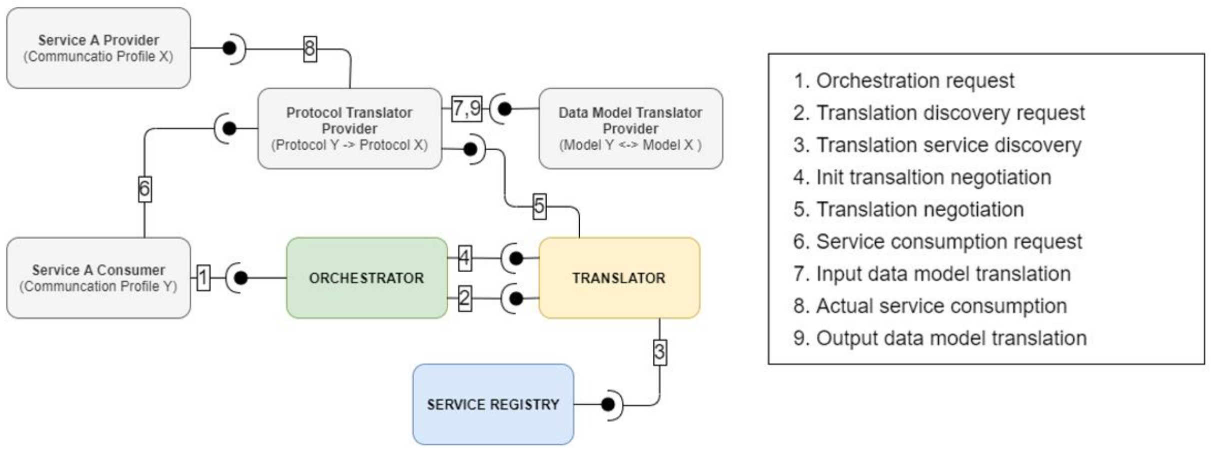

Service A Provider
(Communcatio Profile X)
8
Protocol Translator Provider
(Protocol Y -> Protocol X)
7,9
Data Model Translator Provider
(Model Y <-> Model X)
6
5
Service A Consumer
(Communcation Profile Y)
1
ORCHESTRATOR
4
2
TRANSLATOR
3
SERVICE REGISTRY
1. Orchestration request
2. Translation discovery request
3. Translation service discovery
4. Init transaltion negotiation
5. Translation negotiation
6. Service consumption request
7. Input data model translation
8. Actual service consumption
9. Output data model translation

Figure 4. Translation architecture in Eclipse Arrowhead v5 (planned) [4]

The AI-based translation described later in this document fits both in the version 4.6 and in the version 5 architectural approach of Eclipse Arrowhead, and it is planned to be used in several translation use-cases of the EU KDT Arrowhead fPVN project [4].

2.2 Rule based approach

Traditionally protocol translation has been done in a rule-based manner, also called mapping. In this case, each parameter in one protocol is directly mapped to other parameters in another protocol. This is feasible to do with a limited number of parameters – and between those protocols that do not change often (or ever), such as IPv4 and IPv6, as Figure 5. suggests [8]. Otherwise, if the protocols change and they have different versions, it can get very cumbersome to follow these with rule-based translation. Related issues have been raised by [9] in 2009, and not yet resolved since.

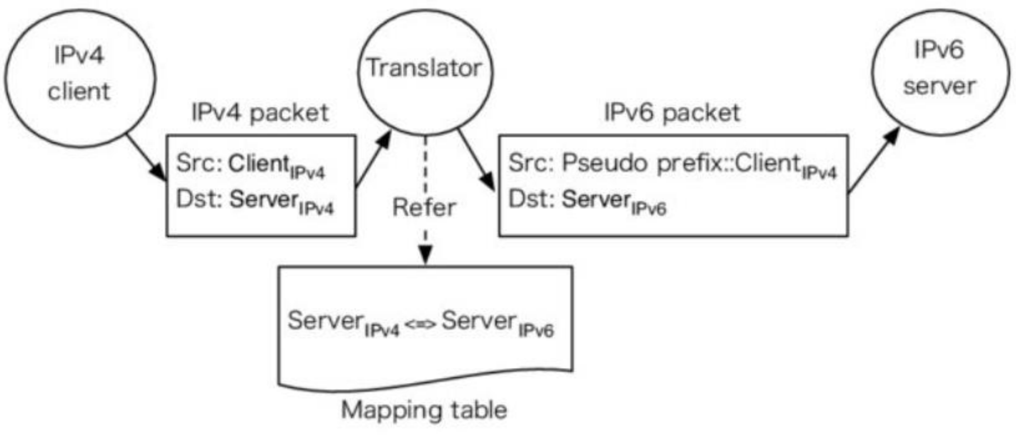


Figure 5. Process of IPv4-IPV6 rule based translation [8]

2.3 Semantic translation

In a recent paper I co-authored, the related work discussed semantic translation as well. The following findings of this subsection have also appeared in [10].

Ontology is a collection of techniques, algorithms, and systems to express information at a high abstraction level and the relationship between data. Ontology provides a conceptual view of an existing data set and forms a higher layer for the operation between providers and consumers.

There are several good examples of ontology-based translation. A. Poggi et al. [11] presented a new ontology language to effectively design systems for Ontology Based Data Access (OBDA). O. Corcho et al. [12] further investigated the OBDA approaches and discussed that the new generation of OBDA is the” mapping translation” concept. Mapping translation is a technique to translate schemas represented in different languages. C. Cruz et al. [13] presented an ontology-based approach for XML data integration and interoperability problems; their solution finds the relation between several XML schemas by defining and investigating XML grammars.

Translating between XML, or JSON – XML schemas is not always straightforward. There are no RFCs or standards to map the data values. This is where ontology could help by adding semantic annotations and supporting metadata translation. F. Moutinho et al. [14] presented a method to solve the interoperability problems between systems which use different XML schemas. The proposed solution is integrated with the Arrowhead frameworks, and it automatically instantiates XML translators based on semantic annotations [15]. G. Amaro [16] et al. further elaborated on this method and extended the work to support JSON- and XML-based systems.

Semantic translation plays a pivotal role in enhancing interoperability between different systems that use XML data. The goal of semantic translation is to ensure that data retains its meaning and significance when transferred between different systems. Recently a comprehensive framework has been presented to tackle the challenges involved in translating XML data across devices with varying levels of semantic understanding. The cornerstone of this approach is the use of semantic annotations within XML schemas.

These annotations enable devices to understand data within their contextual framework, providing a mechanism to map concrete syntax to individualized concepts.

2.4 Translation based on machine learning

Considering the vast number of protocols and protocol versions in use, the interoperability with different systems, or with earlier versions of the same system is problematic. Rule based protocol translators can, and to some extent have overcome this issue, however writing a translator for each pair of protocols is simply unfeasible, considering that this way the number of translators is proportional to the square of existing protocols. More so, considering that in real-world applications manufacturers are not always strictly following the protocol standards. Some protocols cannot even be exactly mapped to others based on rules.

With all these difficulties, the need arises for a translator, that on one hand is not specific to two protocols, that is less rule specific, and can more easily adapt to protocols that are not following strictly the rules, but most of the time are intuitively recognizable and translatable to one that follows on of the standards.

With the recent developments in the field more and more examples can be seen where machine learning based models show signs of intuitivity and can solve a wide range of problems.

In this paper I examine if machine learning based protocol translation can meet these criteria and can provide a viable solution for protocol translation in the future.

The following findings of this subsection have also appeared in [10], as part of the related works study. During our literature research, we found that the models are principally used for data analysis, feature extraction, and real-time prediction, and furthermore, for security reasons (e.g., threat detection, anomaly detection, information protection), or for classification and regression tasks [17][19].

In [18], Kushal Rashmikant Dalal collected several algorithms and ML techniques for IoT security and compared the supervised and unsupervised models. Dalal found that the limitation of the ML techniques is mainly due to data heterogeneity, namely the differences in semantics, formats, and types.

Martin Bauer [20] proposed the usage of a virtualized IoT platform (VirIoT) to share information between producers and consumers by applying a standardized, neutral information model (NGSI-LD). This paper recommends Machine Learning methods during the ontology matching steps to support the developers in their work. The NGSI-LD model is a graph model with the possibility of labeling the information with metadata and also representing the relationship properties. The data format of NGSI-LD is JSON. The basic idea behind the solution is that the ML algorithms could be effectively used to automatically extract information from sensor data and translate it into a common, neutral format. The author applied knowledge infusion [21] to the matching steps.

Neural Machine Translation (NMT) [22] is a state-of-the-art, Neural Network-based technique to translate between languages. The training phase operates with dictionaries, which are formed into word pairs after a preprocessing phase. DNMT (Deep Neural Machine Translation) is a type of NMT which operates with multiple Neural Network layers, not just one.

OpenNMT [23] is an open-source python framework for learning and translating with NMT.

Tennage [24] et al. presented techniques to improve the performance of Neural Machine Translation methods in language translation. The authors applied the OpenNMT [23] framework for the evaluation work. The results show that translating between grammatically similar languages could be improved by using a middle language (e.g., translation to English as a middle step).

In the practical evaluation part of this paper, we also used OpenNMT as a platform to gain initial results with.

2.4.1 OpenNMT

OpenNMT is a versatile neural machine translation framework that offers TensorFlow and PyTorch implementations. While originally developed for language translation, its flexible architecture and customization options have made it useful for a variety of sequence-to-sequence modeling tasks. OpenNMT has gained significant attention for its superior performance and reliability in language translation, producing high-quality translations across multiple language pairs. Additionally, it can be used for protocol translation, where the objective is to convert between different communication protocols. OpenNMT's capability to map between different standards makes it an ideal solution. It manages sequence transformations and the inherent sequential nature of protocols, making it a suitable choice for such tasks. It provides a data-driven approach to achieving protocol interoperability.

2.4.3 ChatGPT

ChatGPT, developed by OpenAI, is a state-of-the-art language model that uses the GPT architecture. It is well-known for its ability to produce coherent and contextually relevant text over long passages. It has been primarily used in natural language processing tasks such as conversation, content generation, and question-answering. However, it can be adapted to various scenarios.

In the context of protocol translation, ChatGPT's sequence understanding, and generation capabilities could be useful. It is pre-trained on vast corpora, which gives it a rich understanding of structures and sequences. With fine-tuning for protocol translation, ChatGPT could potentially map between different communication standards by recognizing and generating the appropriate protocol sequences. Its ability to maintain context over extended sequences can ensure that translations between protocols are accurate and consistent at a macro-level.

Training and evaluation are essential for any model, but the foundational capabilities of ChatGPT make it a promising option for protocol translation applications.

2.5 Methodology

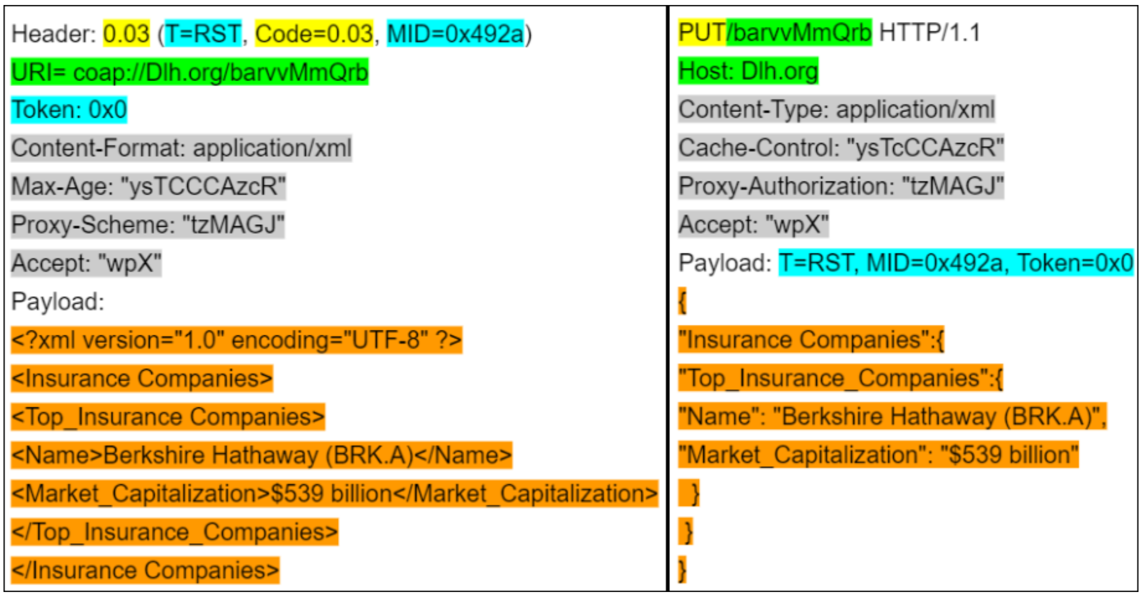
To assess the potential of machine learning in protocol translation, we decided to focus mostly on translating an HTTP message containing XML to a CoAP message having a JSON payload, as these protocols and data structures can be matched with each other, more easily than other protocols.

This necessitated a translator between both the HTTP and CoAP protocols and the XML and JSON formats. While potential solutions for XML to JSON translations already existed, the translation between HTTP and CoAP posed a challenge, primarily due to CoAP's binary protocol format. Existing proxies did not meet the requirements, as a textual representation of CoAP messages was essential for potential NLP techniques.

To address this, I developed a CoAP-HTTP translator, after a thorough research of both the CoAP and HTTP protocols’ structure. Based on the CoAP reference I chose a textual representation of the binary message. The developed translator supports translations from CoAP to HTTP. It identifies and translates distinct sections in CoAP, such as the method, URI, options, and payload, aligning them within the structure of an HTTP message.

As certain CoAP elements do not have direct HTTP counterparts, we had to find a way to incorporate them into the HTTP. Elements like Token, Code, and MID were put into the first line of the HTTP message body. For CoAP options without a clear HTTP header match, I chose the header with the closest meaning possible. To enhance modifiability and clarity, option-header pairs are stored in a configuration file used for the translator, so they can be easily modified if needed. For the translation of the payload, I used the xmltodict python library that can translate between XML and JSON data structures.

Given the need for extensive test data, a program was developed to generate XML-based CoAP messages. Considering the magnitude of data generated, the messages were organized across multiple files of a similar size for better management. The translation program was then refined to process all relevant files within a directory, effectively translating and saving them with distinguishable naming conventions.



```
Header: 0.03 (T=RST, Code=0.03, MID=0x492a)
URI= coap://Dlh.org/barvvMmQrb
Token: 0x0
Content-Format: application/xml
Max-Age: "ysTCCCAzcR"
Proxy-Scheme: "tzMAGJ"
Accept: "wpX"
Payload:
<?xml version="1.0" encoding="UTF-8" ?>
<Insurance Companies>
<Top_Insurance Companies>
<Name>Berkshire Hathaway (BRK.A)</Name>
<Market_Capitalization>$539 billion</Market_Capitalization>
</Top_Insurance_Companies>
</Insurance Companies>
```

```
PUT/barvvMmQrb HTTP/1.1
Host: Dlh.org
Content-Type: application/xml
Cache-Control: "ysTcCCAzcR"
Proxy-Authorization: "tzMAGJ"
Accept: "wpX"
Payload: T=RST, MID=0x492a, Token=0x0
{
"Insurance Companies":{
"Top_Insurance_Companies":{
"Name": "Berkshire Hathaway (BRK.A)",
"Market_Capitalization": "$539 billion"
 }
}
}
```

Figure 6. CoAP request with XML payload and its HTTP translation with JSON payload

In Figure 6., we can observe the structure of both the HTTP and CoAP messages. In the HTTP message, the command marked in yellow indicates to the server the task to be executed. In the case of CoAP, each command has a code. For instance, the code associated with the PUT command is 0.03. We marked the URI in green, which in HTTP appears in two parts. These two parts are presented as a single URI value in CoAP. Together with the HTTP version number, these elements constitute the header of the HTTP message. We omitted the version number in the translation because in the context of a CoAP message, the HTTP version number used is irrelevant.

In the CoAP message, the options marked in gray provide opportunities for transmitting additional information or parameters. In HTTP, the counterparts to these options are the headers, serving a similar purpose. The text marked in orange in the CoAP message represents the payload, in this case, an XML-formatted text. Its equivalent in HTTP is the body, which fundamentally corresponds to the payload. However, we translated the XML into JSON and included the CoAP message's type (T), Message ID (MID), and Token in the first line, all of which are components of the CoAP protocol but can not be found in HTTP.

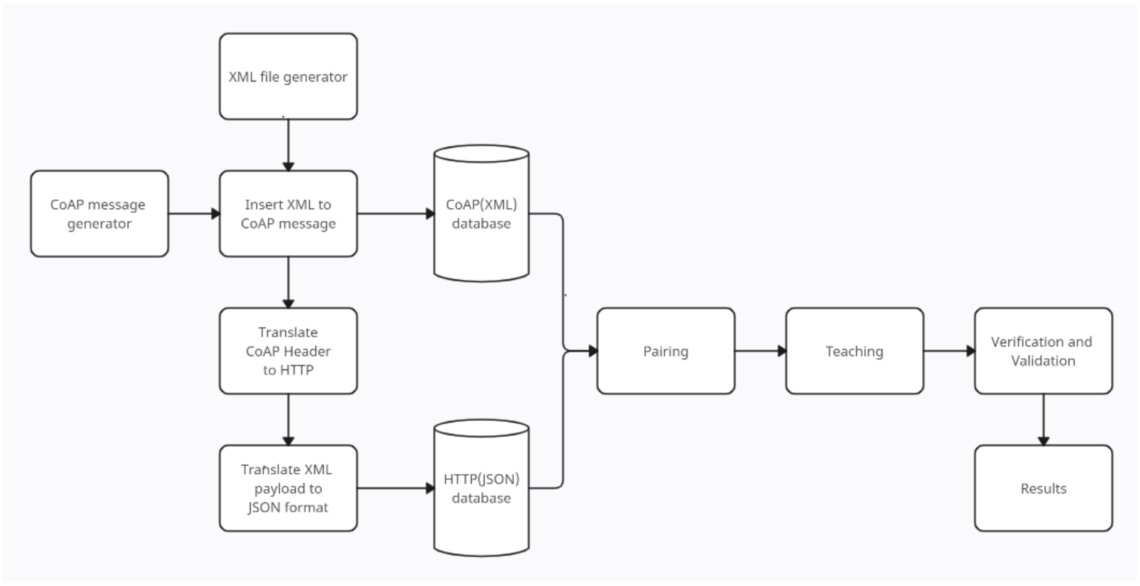

XML file generator
CoAP message generator
Insert XML to CoAP message
CoAP(XML) database
Translate CoAP Header to HTTP
Translate XML payload to JSON format
HTTP(JSON) database
Pairing
Teaching
Verification and Validation
Results

Figure 7. Methodology of the initial testing project

Figure 7. summarizes our initial plan for the testing of creating a machine learning based translator. First, we created a CoAP message and XML data generator. The XML messages were then inserted into the CoAP messages. These CoAP messages were then translated to HTTP messages by a self-made rule-based CoAP-HTTP translator, and their XML payload was translated to JSON by the xmltodict python library. After we generated large numbers of CoAP and HTTP samples which was a training dataset for teaching a machine learning model.

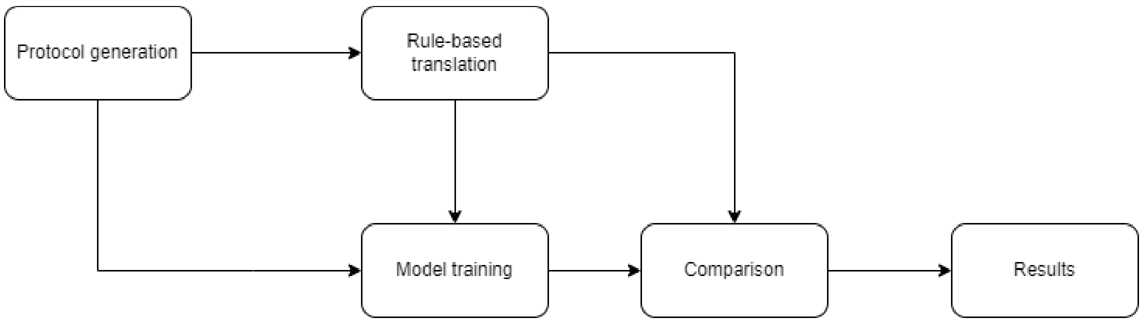


Figure 8. General model training method for testing

Figure 8. shows a more general representation of the teaching of the model, where first a protocol is generated and translated using a rule-based translator. After that, the translated protocol pairs are used for the training of the model, and translations made by the model are compared with rule-based translations to evaluate the accuracy of the model.

It is important to note that this system for model training is only used as a proof of concept for machine learning-based translation, and to be able to experiment with machine learning techniques to reach the maximal translation accuracy, since if a rule-based translator exists there is no real need for a machine learning based translator. Still, as mentioned in the introduction, this work with well-known translatable data structures is inevitable to understand and benchmark the ML tools' capabilities now and later as they improve.

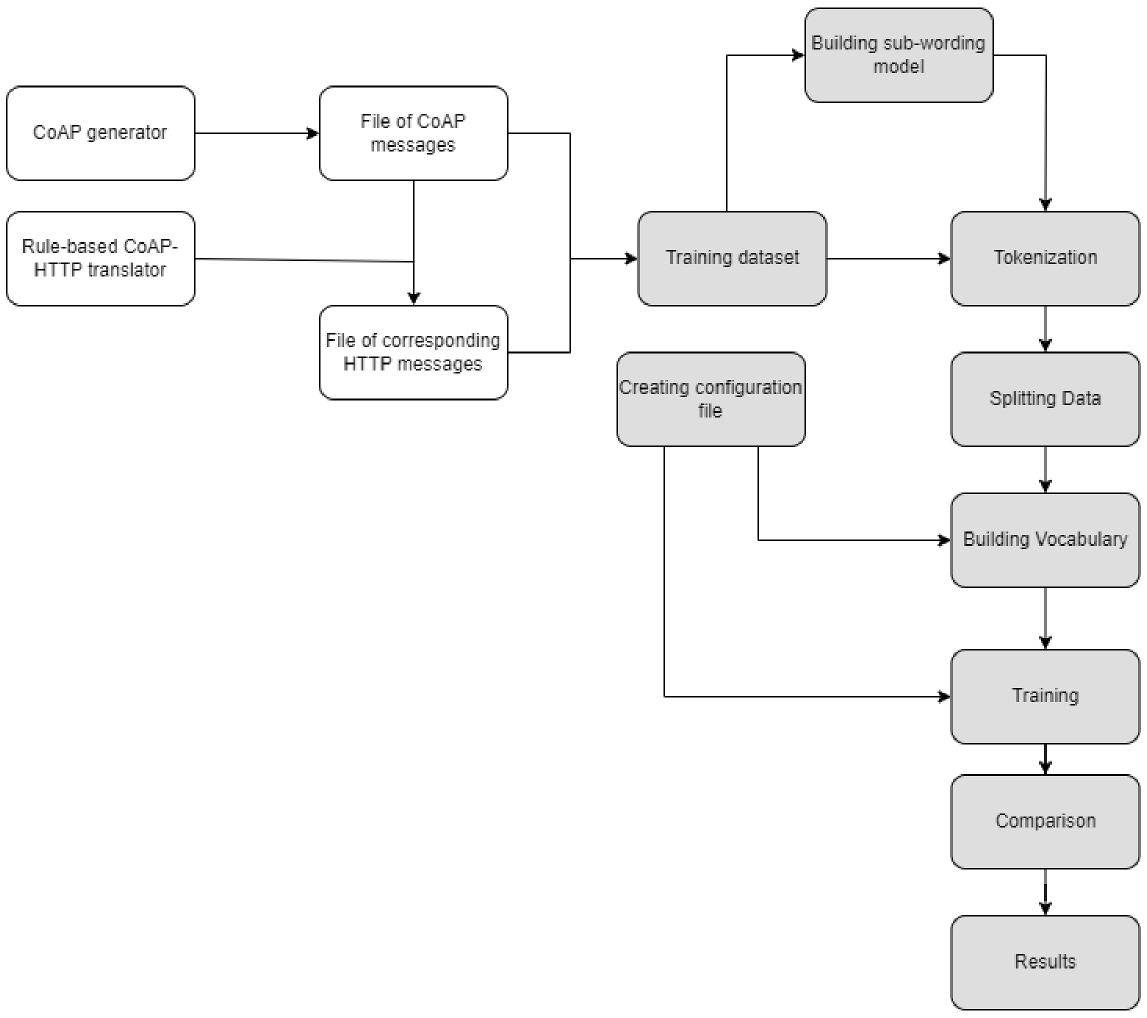


Figure 9. OpenNMT training for CoAP-HTTP translation

In Figure 9. we can observe the individual steps of the training of a CoAP-HTTP translator with the use of OpenNMT:

- CoAP Generator: The CoAP Generator is responsible for creating randomized CoAP messages, which are used as the foundational data for our entire process.
- File of CoAP Messages: Once generated, these CoAP messages are stored in a specific file for easy accessibility during subsequent transformation processes.
- Rule-based CoAP-HTTP Translator: After the generation process, a rule-based CoAP-HTTP translator is employed to derive training pairs essential for machine learning. This translator converts the XML contents of the CoAP message into JSON using the xmltodict online library. As a result, we obtain both XML payloads embedded within CoAP messages and their counterpart JSON payloads within HTTP messages.

- File of Corresponding HTTP Messages: The converted HTTP messages are then stored in another distinct file for easy access during the next steps.
- Training Dataset: Both the CoAP messages and their corresponding HTTP translations are combined to create the training dataset, which is critical for efficient model training.
- Building Sub-wording Model: Before model training, the SentencePiece sub-wording model is used to tokenize the dataset, creating a condensed vocabulary that's optimized for training.
- Tokenization: The dataset is then tokenized to break down the data into more manageable fragments, refining our training inputs.
- Creating Configuration File: A configuration file is created, containing various parameters such as the number of training, validation, and warm-up steps, and an early stopping parameter that dictates the duration of training.
- Splitting Data: The dataset is categorized into three distinct subsets: training, validation, and test data sets to optimize training efficacy.
- Building Vocabulary: A specialized vocabulary is crafted from the tokenized dataset, which becomes the foundation of the training process.
- Training: The model goes through a learning, adapting, and refining phase in this stage.
- Comparison: The model's outputs are compared with the expected results in this critical evaluation phase, which determines the model's accuracy and efficiency.
- Results: The results provide insights into the model's strengths, areas of improvement, and overall performance metrics, concluding the entire process.

An additional step applied at OpenNMT training was masking out the values of the data and only using this masked data for training and evaluation. This was done in order to increase accuracy of the model, as this way the completely random values — whose translation wasn’t nearly as accurate as translating other parts of the message — didn’t have to be translated. after translation masked out values are substituted back to the message.

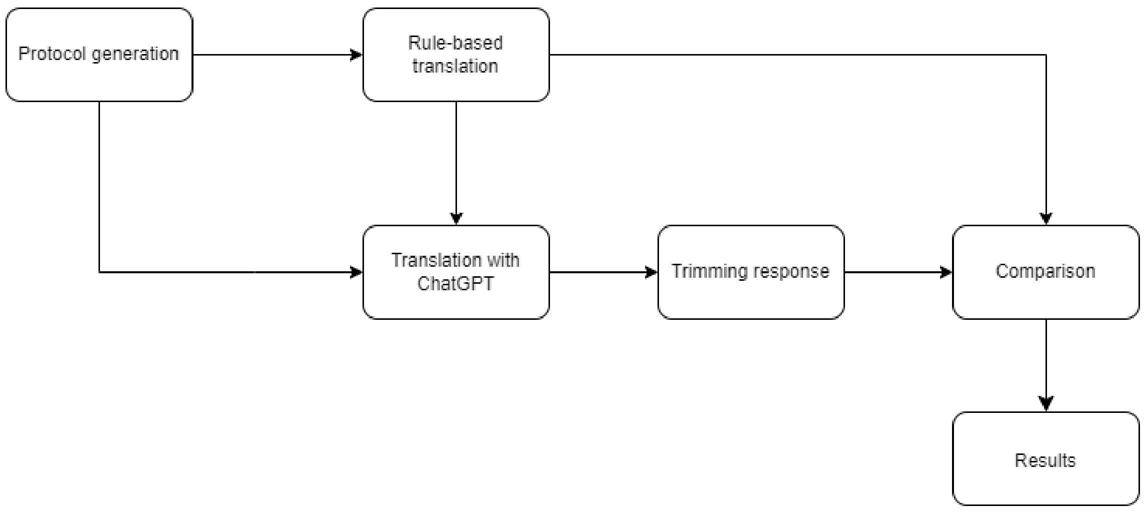


Figure 10. ChatGPT testing method

In Figure 10. our method for the automatized testing of ChatGPT translation can be seen. For assessing the usability of ChatGPT in protocol and data format translation, tests were first conducted on its online website by providing the protocols or datasets to translate by hand. The results were then compared with the translations made by rule-based translators. After getting promising results on testing smaller numbers of translations, the translation was automated by using the OpenAI API and integrating it with the XML and CoAP generator and the rule-based translators.

A script was also created to cut out only the protocol or data structure from the ChatGPT answer, as it often surrounded the answer with comments. Additionally, a comparison script was created to automatically compare the rule-based translation and the ChatGPT-created translation.

2.6 ML-based test data generation

Machine learning has huge potential in the test data generation aspect of protocols and data structures as well. Often, there is a shortage of testing data due to the confidential nature of certain datasets. One of the suggested solutions is to use generative AI for creating test data. By using a small amount of real data to train the generative AI, it can produce vast amounts of test data that closely resemble the original, without revealing any confidential information. This approach not only facilitates thorough testing but also helps in training neural networks. Moreover, it can speed up research by reducing the need for real data, making the whole process smoother and more efficient. Obviously, this proposal requires further research to find the right tools, and to test if generated data is as versatile and can be as effective for testing and training as real-life data would be.

3 Results

All of the test results have tables of the same structure with columns for Train, Valid, and Test representing the sizes of the training, validation, and testing data sets. Val:Train indicates the number of training iterations after which a validation step is performed, with each iteration comprising 50 records. The Train Acc. and Valid Acc. columns display the training and validation accuracy scores, respectively.

3.1 XML-JSON

3.1.1 OpenNMT

During the initial round, the models were trained to translate XML messages containing one layer of embeddedness and one key-value pair. All tag names and value pairs were limited to a discrete set of strings and numbers. This approach reduces the number of possible messages that can be translated. However, since industrial protocol standards have strict rules for data representation, it does not sacrifice the method's usability for the sake of simplification. Table I. showcases the results of the single key-value pair tests,

Train	Valid	Test	Val:Train	Train Acc.	Valid Acc.
1000	500	200	1:10	64.63%	82.55%
2000	500	200	1:10	71.19%	95.01%
5000	1000	200	1:10	88.82%	99.99%
10000	1000	200	1:10	88.37%	100.00%

Table I. Training OpenNMT with XML-JSON with single key-value pairs

Based on the test cases, it was found that the NMT method can effectively learn the structure of data with a relatively small training set. However, to reduce the error rate below 0.01%, a set 100-1000 times larger was necessary. The error rate of OpenNMT training is determined by the number of correctly translated tokens in relation to the total number of tokens. Additionally, the second round of models were trained to translate XML messages with one layer of embeddedness and multiple key-value pairs. The results of this use case are summarized in Table II.:

Train	Valid	Test	Val:Train	Train Acc.	Valid Acc.
2000	2000	200	1:10	50.40%	53.09%
4000	2000	200	1:10	58.38%	54.55%
6000	2000	200	1:10	74.30%	68.08%
8000	2000	200	1:10	88.29%	74.97%
10000	2000	200	1:10	92.10%	77.22%
12000	2000	200	1:10	96.00%	79.07%
14000	2000	200	1:10	98.90%	80.65%
16000	2000	200	1:10	99.00%	81.15%
18000	2000	200	1:10	99.30%	81.77%
20000	2000	200	1:10	99.50%	82.41%

Table II. Training OpenNMT with XML-JSON with multiple key-value pairs

We can observe that both the training and validation accuracy has significantly dropped, and despite these values growing relatively rapidly with increasing the number of samples both the training and validation accuracy has hit an upper boundary, where increasing the number of samples further could not increase the training or validation accuracy to 100%. We can further see how validation accuracy slowly approaches an upper limit on Figure 11.

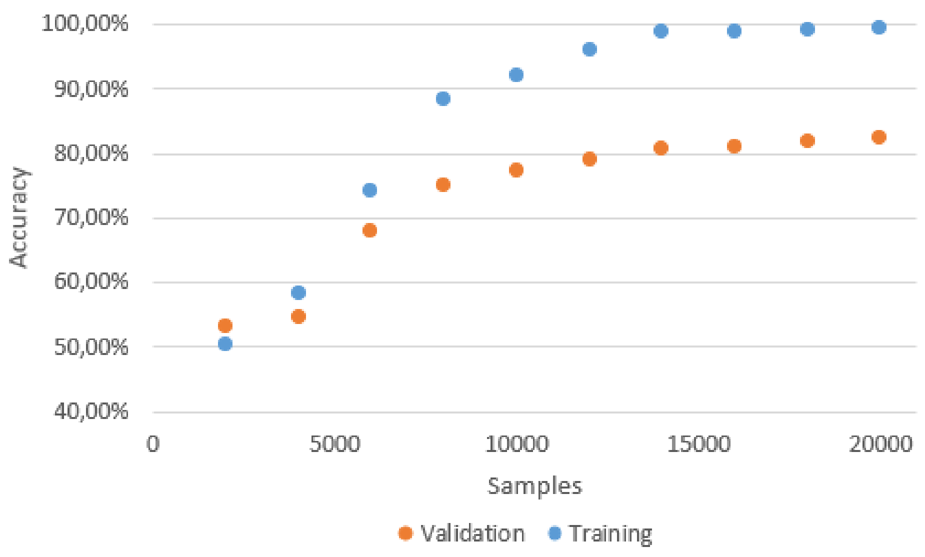


Figure 11. Translation accuracy of XML-JSON with multiple key-value pairs

Due to most values within the XMLs being random values their translation wasn't efficient; therefore, the idea of masking was proposed. By masking we are deliberately replacing numbers as strings within the document and later substitute them back algorithmically. The results reached with this technic are summarized in Table III:

Train	Valid	Test	Val:Train	Train Acc.	Valid Acc.
2000	2000	200	1:10	98.73%	98.58%
4000	2000	200	1:10	98.78%	98.58%
6000	2000	200	1:10	98.77%	98.59%
8000	2000	200	1:10	98.80%	98.57%
10000	2000	200	1:10	98.77%	98.60%
12000	2000	200	1:10	98.79%	98.55%
14000	2000	200	1:10	98.77%	98.56%
16000	2000	200	1:10	98.77%	98.58%
18000	2000	200	1:10	98.77%	98.60%
20000	2000	200	1:10	98.77%	98.57%

Table III. Training OpenNMT with XML-JSON with multiple key-value pairs and masked-out values

In Table III. we can observe much higher training and validation accuracy even at lower number of samples. Another important result in these training data is the significantly higher validation accuracy that was reached. Compared to the unmasked translation the validation accuracy of never went above 83%, here we could reach an accuracy of 98.6%.

These results suggest that masking out values can be a useful tool for improving translation accuracy, even though other methodologies must be implemented complementing this one to reach higher accuracy values.

3.1.2 ChatGPT

Initially, I gave 100 XML formatted texts for the GPT3.5 language model for translation with the prompt of:

” Translate the following XML formatted text to JSON. Only give the translation as an answer, nothing else.” and added the generated XML to the prompt.

The XMLs were randomly generated using completely random strings and a maximum depth of 3. After comparing the results with the same XMLs translated by the xmltodict python library used before I found that disregarding whitespaces 89 out of 100 translations were identical to the translations made by the rule-based algorithm.

After using the automatized method described in the methodology 90% of the messages were perfectly translated by the GPT 3.5 turbo model. After changing to GPT 4, the percentage of perfect translations went up to 97%.

Despite these promising results, there were many unreliable aspects of translation with ChatGPT: when the XMLs were unformatted, without tabulators and newline characters, the translation accuracy dropped significantly. Also, often there was no response for the requests made, causing the function to timeout. In some cases, additional syntaxes were added by the translator that are used in some cases, but not always, so the translation need additional trimming to be able to get it into a uniform form. Despite emphasizing in the prompt that only the translation should be given and nothing else, often there was other text surrounding the data, so it was necessary to find and trim the part of the response, which contained the translation itself.

Considering these results ChatGPT can be viewed as a promising tool, that could be used in non-critical cases. Its accuracy could possibly be improved by further experimenting with prompting, and better trimming of the response.

3.2 XML-SenML

As a second use case, we trained and tested JSON-SenML translation models. SenML[25] is a predefined format for representing sensor measurement values. It could be a common language for providers who apply different schemas and provide different JSON payload formats. The JSON messages were algorithmically generated to fit the provider message schemas. Table IV represents the JSON-SenML accuracy results of the trained OpenNMT models.

Train	Valid	Test	Val:Train	Train Acc.	Valid Acc.
2000	1000	100	1:10	81.95%	87.57%
4000	1000	100	1:10	84.75%	87.56%
6000	1000	100	1:10	85.68%	87.49%

Table IV. Training OpenNMT with XML-SenML

Here we can see that validation accuracy doesn’t increase as the number of samples gets bigger.

3.3 CoAP-HTTP

3.3.1 OpenNMT

With CoAP HTTP translation after the initially unpromising unmasked tests, where a maximum accuracy of less than 80% was reached even with 20.000 training samples, I got the results presented in Table V from the tests, where values were masked out:

Train	Valid	Test	Val:Train	Train Acc.	Valid Acc.
2000	2000	200	1:10	98.38	98.57
6000	2000	200	1:10	98.44	98.57
10000	2000	200	1:10	98.48	98.56
20000	2000	200	1:10	98.22	98.56

Table V. Training OpenNMT with CoAP-HTTP with masked-out values

These results show a validation accuracy of more than 98.56% for each number of training samples, showing that an even smaller amount of training samples could be necessary, and could result in higher accuracy than what was reachable by unmasked testing.

3.3.2 ChatGPT

For CoAP-HTTP translations we got 0 perfect translations. The CoAP representation used is arbitrary and ChatGPT has likely never encountered it during its training, which could partially explain why it was unable to accurately translate CoAP messages to HTTP. However, the translations provided by ChatGPT showed similarities to the correct translation. This suggests that providing examples or fine-tuning could lead to more accurate translations using these protocols.

Examples of the original CoAP message, the rule-based HTTP translation and the HTTP translation made by ChatGPT:

CoAP message:

Header: 5.00 (T=NON, Code=5.00, MID=0xed98)

Token: 0x1

Content-Format: application/xml

Location-Query: "frost"

Proxy-Uri: "zone"

Proxy-Scheme: "steam"

Payload: ""

HTTP message translated by Rule-based translator:

HTTP/1.1 500 Internal Server Error

Content-Type: application/xml

URL Query String: "frost"

Proxy-Authorization: "zone"

Proxy-Authorization: "steam"

Payload: T=NON, MID=0xed98, Token=0x1

""

HTTP message translated by ChatGPT:

HTTP/1.1 500 Internal Server Error

Content-Type: application/xml

Location: ?frost

Uri: zone

Scheme: steam

Payload:

We can observe that the first line is properly translated, and the Content-Format header was translated to the Content-Type option, however the rest of the headers could not be translated. The payload also doesn’t include the T, MID and Token values since these are attributes that do not exist in HTTP and it was our decision to put it in the payload.

After giving it an example in the prompt, and instructing to put the T, MID and Token values into the payload, we get the same kind of results:

CoAP message:

Header: 4.03 (T=CON, Code=4.03, MID=0x27fd)

Token: 0x7

Content-Format: application/xml

Location-Path: "weather"

If-Match: "radar"

Payload: ""

HTTP message translated by Rule-based translator:

HTTP/1.1 403 Forbidden

Content-Type: application/xml

URL Path: "weather"

If-Match: "radar"

Payload: T=CON, MID=0x27fd, Token=0x7

""

HTTP message translated by ChatGPT:

HTTP/1.1 403 Forbidden

Content-Type: application/xml

If-Match: "radar"

Location: "weather"

Payload:

Here we can see that although some options are correctly translated, the T, MID and Token values are still not put in the payload, which suggests that good prompting might not be enough for CoAP-HTTP translation, and further improvement of the model or fine-tuning might be needed.

4 Summary

The main focus of my study was to research the viability of using machine learning for data format and protocol translation. Although many traditional methods show promising results for data structure and protocol translation, there are countless areas where applying rule-based or semantic translation is simply not feasible.

Through experimentation, I found two promising methods of using machine learning tools for translation. One of these methods was the use of OpenNMT, an open-source neural machine translation framework. Initial test results with OpenNMT showed low accuracy for translation even when training was done on a large number of samples. After investigating the reasons for mistranslated messages, the idea of masking out the values was proposed. From the results it is clear that masking out the values significantly improved the accuracy of the translation. This suggests that it could be a great tool to increase accuracy in further developments of AI translation. Considering the fact that OpenNMT was designed for natural language translation and not for protocols and data-structures, a tool developed specifically for these translations could function as a usable protocol-translator in the future.

The other promising method tested for translation is ChatGPT, a state-of-the-art large language model. Like OpenNMT, ChatGPT was not designed for protocol translation, but its general knowledge from the terabytes of internet data makes it capable of translation. With approximately 90% of the messages being correctly translated, resulting in an approximately 98.7% key-value translation, ChatGPT outperforms OpenNMT in accuracy. However, I found that it is much less accurate in translations where data structures or protocols are less common and less commonly translated to each other. Another big drawback is the ChatGPT API having bugs, such as not getting answers for requests until it times out. These unreliabilities make ChatGPT currently unviable for protocol translation. However, there have been many improvements, such as the GPT4 model, which has shown better performance than the GPT3.5 tested in this study. A new feature also allows the fine-tuning of ChatGPT, which could be helpful in achieving better translation results even with less frequent protocol pairs. With new advancements constantly happening in the realm of AI and ChatGPT in particular, I believe ChatGPT is a very promising tool and needs further research to be able to fully realize its potential in protocol and data-structure translation.

In summary, my study provides a great starting point for further research in ML-supported protocol translation while showing early stages of success in this promising field. With further improvement in these areas and tools made more specifically for protocol and data-structure translation, machine learning-assisted translation could be a groundbreaking advancement in system interoperability.

5 References

[1] Arrowhead flexible Production Value Networks (fPVN), EU KDT project, Online: https://fpvn.arrowhead.eu/about/ (Accessed: 2023.11.01.)

[2] H. Wang, H. Wu, Z. He, L. Huang, K. W. Church, "Progress in Machine Translation", Engineering, Volume 18, pp. 143-153. Elsevier, 2022.

[3] J. Delsing, (Ed.), "*IoT Automation: Arrowhead framework",* CRC Press, 2017.

[4] P. Varga, T. Bordi, J. Delsing, "D2.1 - First version of the Arrowhead fPVN microservice architecture", EU KDT Arrowhead fPVN, 2023.

[5] P. Srisuresh and K. Egevang, "*Traditional IP network address translator (Traditional NAT)"* IETF RFC3022, January 2001.

[6] H. Derhamy,J. Eliasson, J. Delsing, P. P. Pereira, and P. Varga, "Translation error handling for multi-protocol soa systems", in 2015 IEEE 20th Conference on Emerging Technologies Factory Automation, (ETFA), 2015.

[7] Eclipse Arrowhead framework – Design documents and Java Reference implementation, Online: https://github.com/eclipse-arrowhead/core-java-spring/wiki (Accessed: 2023.11.01.)

[8] K. Shima, W. Ishida, and Y. Sekiya, Y., "Design, Implementation, and Operation of IPv6-only IaaS System with IPv4-iPv6 Translator for Transition toward the Future Internet Datacenter", in *CLOSER* 2012, (pp. 306-314).

[9] Y.D. Bromberg, L. Réveillère, J. L. Lawall, and G. Muller, "Automatic generation of network protocol gateways", in Middleware 2009: ACM/IFIP/USENIX, 10th International Middleware Conference, Urbana, IL, USA, November 30–December 4, 2009. Proceedings 10 (pp. 21-41). Springer Berlin Heidelberg.

[10] T. Tóthfalusi, E. Varga, Z. Csiszár, and P. Varga, "ML-based translation methods for protocols and data formats", in IEEE Conference on Network and Service Management, CNSM, AnServApp Workshop, Niagara Falls, Canada, 2023.

[11] A. Poggi, D. Lembo, D. Calvanese, G. De Giacomo, M. Lenzerini, and R. Rosati, "Linking data to ontologies," J. Data Semantics, vol. 10, pp. 133–173, 01 2008.

[12] O. Corcho, F. Priyatna, and D. Chaves-Fraga, "Towards a new generation of ontology based data access," Semantic Web, vol. 11, pp. 153–160, 01 2020.

[13] C. Cruz and C. Nicolle, "Ontology-based heterogeneous xml data integration." JDIM, vol. 3, pp. 133–138, 01 2005.

[14] F. Moutinho, L. Paiva, P. Malo, and L. Gomes, "Semantic annotation of data in schemas to support data translations," in IECON 2016 - 42nd Annual Conference of the IEEE Industrial Electronics Society, 2016, pp. 5283–5288.

[15] F. Moutinho, L. Paiva, J. Köpke, and P. Malo, "Extended semantic annotations for generating translators in the arrowhead framework," IEEE Transactions on Industrial Informatics, vol. 14, no. 6, pp. 2760–2769, 2018.

[16] G. Amaro, F. Moutinho, R. Campos-Rebelo, J. Köpke, and P. Malo, "Json schemas with semantic annotations supporting data translation," Applied Sciences, vol. 11, no. 24, 2021.

[17] M. S. Mahdavinejad, M. Rezvan, M. Barekatain, P. Adibi, P. Barnaghi, and A. P. Sheth, "Machine learning for internet of things data analysis: a survey," Digital Communications and Networks, vol. 4, no. 3, pp. 161–175, 2018.

[18] K. R. Dalal, "Analysing the role of supervised and unsupervised machine learning in iot," in 2020 International Conference on Electronics and Sustainable Communication Systems (ICESC), 2020, pp. 75–79.

[19] S. P. Khedkar and R. AroulCanessane, "Machine learning model for classification of iot network traffic," in 2020 Fourth International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC), 2020, pp. 166–170.

[20] M. Bauer, "Iot virtualization with ml-based information extraction," in 2021 IEEE 7th World Forum on Internet of Things (WF-IoT), 2021, pp. 915–920.

[21] J. Fürst, M. F. Argerich, B. Cheng, and E. Kovacs, "Towards knowledge infusion for robust and transferable machine learning in iot," Open Journal of Internet Of Things (OJIOT), vol. 6, no. 1, pp. 24–34, 2020.

[22] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," ArXiv, vol. 1409, 09 2014.

[23] "OpenNMT, An open source neural machine translation system." https://opennmt.net/.

[24] P. Tennage, A. Herath, M. Thilakarathne, P. Sandaruwan, and S. Ranathunga, "Transliteration and byte pair encoding to improve tamil to sinhala neural machine translation," in 2018 Moratuwa Engineering Research Conference (MERCon), 2018, pp. 390–395.

[25] C. Jennings, Z. Shelby, J. Arkko, A. Keranen, and C. Bormann, RFC8428 – Sensor Measurement Lists (SenML), Std. IETF, 2018.