

Adding a Network Coding Extension to CoAP for Large Resource Transfer

Bertram Schütz, Nils Aschenbruck

University of Osnabrück - Institute of Computer Science,
Albrechtstr. 28, 49076 Osnabrück, Germany
{schuetz, aschenbruck}@uos.de

Abstract—This paper presents a smooth way to include Network Coding in the Constrained Application Protocol (CoAP) for large resource transmissions. Devices in the Internet of Things usually communicate using short messages with little data. In some cases, for example, requesting firmware updates, bigger resources need to be transferred. CoAP's recently finalized blockwise transfer scheme can handle large resources, but is not efficient in lossy environments. Network Coding has proven to be more error resistant. This paper demonstrates the limitations of CoAP's existing blockwise transfer scheme and presents a new approach, based on Network Coding. The evaluation compares CoAP's regular blockwise transfer to the new Network Coding extension. Measurements on an implemented client-server application with simulated losses and delay, confirm the benefits of the extension, resulting in reduced transfer durations.

Keywords—Network Coding, CoAP, IoT

I. INTRODUCTION

IoT-Devices usually communicate using short messages. Most of the time, temperature-sensors, intelligent valve-tabs or light-switches just send and receive state information. The corresponding messages contain only some bytes of payload. The Constrained Application Protocol (CoAP) [16] was explicitly designed to handle such communication. While CoAP's basic transmission scheme is sufficient for the regular communication, it does not handle large resource transfer. Those large resources are not requested by devices in their regular working routines, but are still important to guarantee a functioning, healthy IoT. Firmware updates are one type of large resources to handle. Another occurrence is given on mobile devices. In example, a fitness tracker permanently collects data and usually sends it in short periods to a server. If the connection is lost, the device keeps collecting information. Once the connection is re-established, the meanwhile grown data has to be sent, corresponding to a large resource.

The CoRE Working Group, which develops the CoAP specification, has acknowledged the problem of large resource transfer. Their approach to the solution is a block-based transmission scheme, called blockwise transfer [2]. This scheme splits a large resource into block-sized chunks. Each chunk is then transmitted as the payload of a message. The receiver can reconstruct the obtained chunk's position in the resource by processing an additional block option, which is added to the message. After receiving a chunk, the receiver requests the next one by acknowledging the corresponding received message. This scheme is sufficient, but not efficient. The receiver has to acknowledge every obtained chunk, leading to a huge number of retransmissions in lossy environments and a slow

completion time in the case of large delays. One approach, which has been proven as a viable solution for data transfer in lossy environments, is Network Coding. By transmitting coded packets, which consist of linear combinations of the original packets and can be decoded at the receiver, Network Coding can reduce the loss-indicated problems by increasing error tolerance [3] [6] [9].

The remaining part of this paper is organised as follows: In Section II, the basic principles are stated, which are needed to comprehend this work. The related work is given in II-C. Section III presents the proposed design for a smooth inclusion of Network Coding in CoAP, to solve the problem of large resource transfer in lossy environments. The design's implementation, which is used for the evaluation, is described in IV. The theoretical benefits of the proposed design are confirmed in section V, by running the implemented application in an emulated lossy environment. A critical reflection of the proposed design and evaluation is given in VI. Finally, Section VII draws a conclusion of the presented work and outlooks the next steps.

II. BACKGROUND & RELATED WORK

A. Constrained Application Protocol

The Constrained Application Protocol (CoAP) [16] is a proposed standard by the IETF Constrained RESTful environments (CoRE) Working Group. CoAP is designed to give constrained devices, for example microcontroller used in the IoT, access to the main features of a representational state transfer (REST) architecture. The features include resource requests via get, put, post and delete, commonly used in HTTP. The protocol also specifies discovery for machine-to-machine communication.

Message format: One central idea of CoAP is keeping the message overhead small, while still enabling the targeted REST-features. Therefore, CoAP's header is only 4 bytes long, with additional fields for a token and multiple possible options, followed by the payload. Figure 2 shows the specified CoAP message format, based on its presentation in [16]. The 2-bit version number *Ver* indicates the used CoAP specification. A 2-bit type field *T* indicates, if the message is a confirmable or non-confirmable request, an acknowledgement, or a reset message. *TKL* describes the length of the token. The Message ID is an 16-bit unsigned integer for duplication detection and matching between confirmable messages and their corresponding acknowledgements. The *Token* is used to pair responses with their belonging requests. A one byte *Payload Marker*

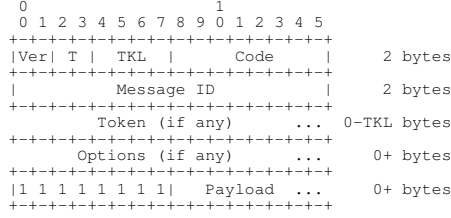


Figure 1. CoAP message format, cf. [16].

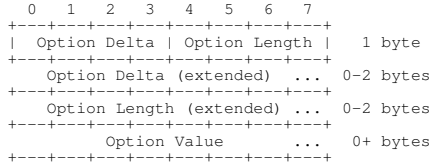


Figure 2. CoAP option format, cf. [16].

0xFF indicates the payload's start. One central design-idea of CoAP is adding predefined options to a message. By including an option, the sender can force the receiver to process the containing message in a specific way. Options have to satisfy the specified option format, shown in Figure 2. The option delta is used to order a message's options, if multiple options are included. The option length is given in bytes, including the size of the option value, which contains the parameters used when processing the option.

Blockwise transfer: CoAP's specification [16] includes a simple transfer scheme for large resource handling, called blockwise transfer. The scheme was discussed in the Internet-Draft [2], which was recently submitted to IESG. It is based on two additional options *Block1* and *Block2*. The first option is used to handle payload of request-messages, while *Block2* indicates the processing of response-payload. From a client-server perspective, a *Block1* Option is added by the Client to indicate blockwise data upload to the server, while a *Block2* option is added by the Server to indicate blockwise processing of the download's response-data. The option value of a block option is shown in Figure 3. It is equal for both block options, consisting of three fields, *NUM*, *M* and *SZX*.

- *NUM*: The *NUM* field indicates the number of the block in the transmission's resource, corresponding to the containing message's payload.
- *M*: The *M* bit, also called *more* flag, is used to indicate, if there are more bytes of data, belonging to the resource. If the flag is zero, there is no more data to process, after handling the containing message's payload.
- *SZX*: This *SZX* field is used as a compressed representation of the blocksize in bytes. The blocksize is defined as 2^{4+SZX} .

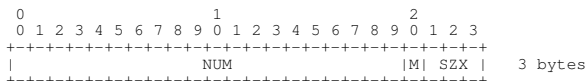


Figure 3. CoAP block option value, cf. [2]

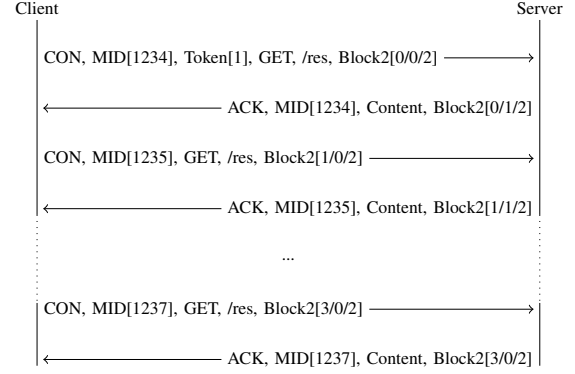


Figure 4. Blockwise Transfer, cf. [2].

The processing of a block option is shown in Figure 4. The client requests resource *res* by sending message 1234, flagged with the get method. The message contains a *Block2* option, meaning the client wants to get the resource's data attached as payload in the response. The block option is represented by its fields *[NUM/M/SZX]*. The option value *[0/0/2]* indicates that the client orders the first $2^{4+2} = 64$ bytes of the resource. The server responds with an acknowledgement to the same *MID*. Block response's option value *[0/1/2]* shows that the server accepts 64 bytes as blocksize. The activated *more* Flag 1 indicates that the resource contains more bytes to process. While the server's response contains an activated *more* flag, the request-response exchange continues. The last shown acknowledgement includes a de-activated *more* flag, indicating that the last block of the resource is transmitted.

B. Network Coding

Network Coding is a concept, settled in the field of information theory. It can be classified in two categories, called *intersession* and *intrasession coding*. Combining datastreams of different senders to increase the network's overall throughput is called *intersession coding*. Instead of coding between different streams for throughput gain, Network Coding can also be used to increase one sessions robustness and error-tolerance in lossy environments. This approach is called *intrasession coding* and is used in this paper.

Random linear Network Coding: The basic concept of Network Coding was introduced and formally defined by Ahlswede et al. in [1]. Ahlswede et al. used bitwise XOR to code two original packets into one coded packet. In general Network Coding is not limited to such a simple coding scheme. While proving the max-flow min-cut theorem for multicast, Li et al. also introduced a linear coding scheme, where a packet is defined as a vector over a finite field [10]. This Linear Network Coding approach allows to shift the view upon a packet from a physical domain to an algebraic domain. With Linear Network Coding, a packet is not only considered as a series of zeros and ones anymore, but rather as a polynomial from a Galois-Field $GF(n^q)$. For example, the 8-bit sequence 11001010 can be seen as an element of $GF(2^8)$ by treating it as the polynomial $1x^7 + 1x^6 + 0x^5 + 0x^4 + 1x^3 + 0x^2 + 1x + 0$. In the new algebraic domain, a coding operation on packets can be done by creating a linear combination of the original packets, with coefficients

taken from the finite field. For example, a node could create the coded packets q_1, q_2 from the original packets p_1, p_2 by using the linear combinations $q_1 = \alpha_{11}p_1 + \alpha_{21}p_2$ and $q_2 = \alpha_{12}p_1 + \alpha_{22}p_2$ with coding coefficients $\alpha_{ik} \in GF(n^q)$. Following [17], the resulting encoding process for packets with l symbols can be written as:

$$\begin{pmatrix} q_{11} & q_{12} & \cdots & q_{1l} \\ q_{21} & q_{22} & \cdots & q_{2l} \end{pmatrix} = C \cdot \begin{pmatrix} p_{11} & p_{12} & \cdots & p_{1l} \\ p_{21} & p_{22} & \cdots & p_{2l} \end{pmatrix} \quad (1)$$

with coefficient matrix

$$C = \begin{pmatrix} \alpha_{11} & \alpha_{21} \\ \alpha_{12} & \alpha_{22} \end{pmatrix} \quad (2)$$

Upon receiving a new linear independent packet, a sink-node adds the representing vector to its local stored Decoding Matrix D and thus increases its rank. Once enough linear independent packets are accumulated, the node can decode them by solving the represented system of linear equations using Gaussian elimination. The condition for decoding can be written as: $r(D) \geq r(G)$ where $r(D)$ is the rank of the Decoding Matrix and $r(G) = n$ is the rank of the Generation Matrix, which corresponds to the number n of original packets involved in the encoding process. This leads to the here used representation of the decoding condition: $r(D) \geq n$. The decoding condition can be seen as having a linear system of n equations with n unknowns. Using a linear solver like Gaussian elimination for decoding, the decoding process can retrieve the original packets p_1 and p_2 by using the inverted Coefficient Matrix C^{-1} :

$$\begin{pmatrix} p_{11} & p_{12} & \cdots & p_{1l} \\ p_{21} & p_{22} & \cdots & p_{2l} \end{pmatrix} = C^{-1} \cdot \begin{pmatrix} q_{11} & q_{12} & \cdots & q_{1l} \\ q_{21} & q_{22} & \cdots & q_{2l} \end{pmatrix} \quad (3)$$

The basic linear coding approach was extended by Ho et al. to Random Linear Network Coding (RLNC) in [6] and [7] by allowing nodes to independent choose random packet combinations. This scheme allows a sender to continuously create independent packets with high probability [9]. An obtained, independent packet increases the decoding matrix' rank at the receiver, leading one step closer to satisfy the decoding condition, and thus restoring the original data.

Using generations: To fit the data of a large resource in one coding-decoding step, a high ranked Generation-matrix must be used, resulting in high coding complexity, requiring a lot of memory and computing power. To tackle this problem, generation-based intrasession coding is presented in [7]. Instead of coding over all chunks from a resource's data, the data is split in so called generations. Each generation can then be used for coding, meaning the sender creates coded payloads only from data belonging to the active generation. The receiver decodes each generation, restoring the resource's data generation-wise. This scheme requires additional communication between sender and receiver, to acknowledge a decoded generation and request the next one. The smooth inclusion of generation-based, random linear Network Coding into CoAP is one main contribution of this work.

Advantages in lossy environments: Traditional block-based schemes, like CoAP's blockwise transfer, divide a resource's data into n chunks, corresponding to original packets $p_1, p_2, \dots, p_n$. The sink needs to receive each original packet

p_i to restore the resource's data. If a packet p_i is lost, the sender has to retransmit exactly the lost packet p_i . This scheme requires timeouts at the sender or additional communication by the receiver in lossy environments.

The advantages of intrasession coding results from the decoding condition. The receiver can restore the n original packets of a generation, after obtaining n independent coded packets from the generation. In the case of losses, it doesn't matter, which packet is lost. As long as n independent coded packets are received, the decoder can restore the original data with high probability [9].

C. Related Work

The fundamentals of Network Coding were presented by Ahlswede et al. in [1]. Koetter and Médard introduced an algebraic framework for Network Coding [9], which set the foundation for further research. Random linear Network Coding was presented in [7]. Heide et al. presented an evaluation on the effects of different coding parameters in [5]. Their recommendations are considered and used in this work, with respect to the special requirements for CoAP and it's targeted devices.

The CoAP specification is given in [16], including block-wise transfer, which was discussed in the now expired draft [2]. A related approach to include Network Coding for constrained devices was conducted by Salhi et al. in [15]. The french researcher added Network Coding to Zigbee sensor networks to increase bandwidth utilization and reliability. Their approach features XOR coding instead of random linear Network Coding and is not smooth compatible to the CoAP specification, due to a special designed echo-feedback mechanism. The closest related work to this paper was done by Järvinen et al. in [8]. The group from Helsinki did not use Network Coding, but conducted an experimental evaluation of alternative congestion control algorithms for CoAP, tackling a similar problem with a different approach. Järvinen et al. showed that TCP-like congestion control mechanisms can reduce a client's completion time. The focus from Järvinen et al. lies on congestion control mechanisms, while the here presented work uses Network Coding. Both approaches tackle a similar problem in packet delivery.

III. ARCHITECTURAL DESIGN

To include Network Coding smoothly in CoAP, features from CoAP's specification were used. The final implemented scheme is shown in Figure 5. It's details will be presented in this section. The here proposed Network Coded transfer scheme is based on two new options, called *NC_Control* and *NC_Coefficients*.

A. NC_Control

To enable reliable, successful decoding, the encoder at the sender and decoder at the receiver need to be initialised with the same three coding parameters: number of symbols per generation, size of one symbol in bytes and a finite field indicator. The here presented approach uses a new defined option to communicate those parameters, called *NC_Control*. By adding the *NC_Control* option to a request, the client indicates it's wish for a network coded transmission, as shown

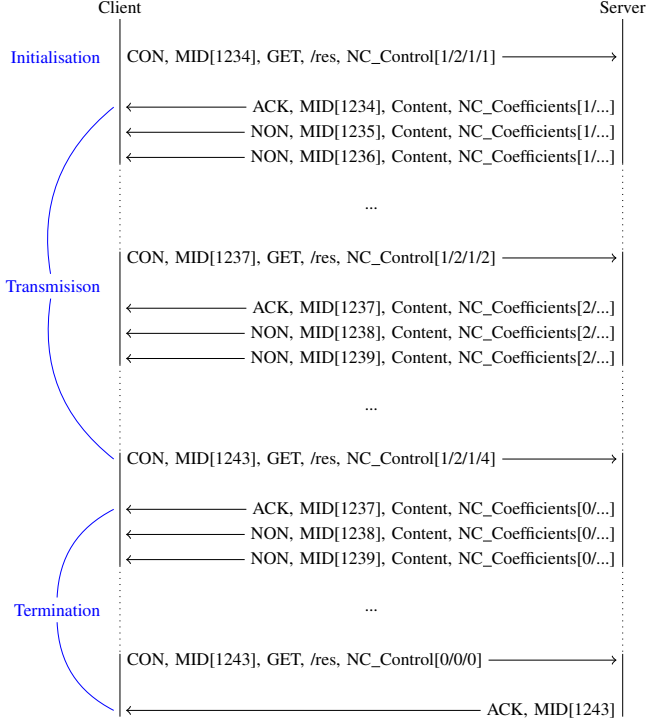


Figure 5. Network Coded transfer.

in Figure 5. The option is defined as critical and must be processed by the server, or the including message is rejected. The coding parameters are included in the *NC_Control* option value as the fields *NOS*, *SOS* and *FFE*. The *NC_Control* option value format is shown in 6. The fields are defined as following:

***NOS (Number Of Symbols)*:** The number of symbols parameter indicates how many symbols belong to one generation. This corresponds to the size of the coding window. It can be understood as the number of original data-chunks, from which the encoder can build the coded payload. In the new *NC_Control* option, *NOS* is defined as a one byte long unsigned integer. Due to its 8-bit length, *NOS* is limited to 256. A higher number of symbols leads to a higher coding complexity. A careful selection of the used *NOS* is vital in this scope, reflecting a compromise between a possible Network Coding gain and an overwhelming coding complexity for constrained devices.

***SOS (Size Of Symbol)*:** This parameter indicates the size of one symbol in bytes. It is defined as a 10-bit unsigned integer. This enables a maximum symbol size of 1024 bytes, similar to the blocksize of the blockwise transfer scheme [2, Section 2.2.]. This maximum is not limited to avoid IPv6 fragmentation, and should thus not always be used, following a basic concept of CoAP's specification.

***FFE (Finite Field Exponent)*:** To encode a chunk of data, the encoder creates a linear combination of an original chunk using a coding coefficient. The coefficients are taken from a finite field $GF(2^q)$. The finite field exponent *FFE* indicates, which finite field $GF(2^{FFE})$ is used for coding. A higher exponent leads to a better decoding probability, but also an

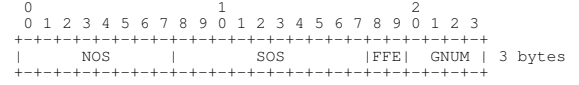


Figure 6. *NC_Control* option value.

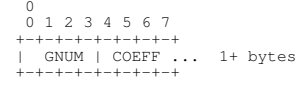


Figure 7. *NC_Coefficients* value.

increased coding complexity. Due to the efficient operations in binary $GF(2^q)$ fields, only the finite fields $GF(2^{2^1})$, $GF(2^{2^2})$ and $GF(2^{2^3})$ are considered. Those fields allow decoding with high probability for the available *SOS* and *NOS* parameters [5]. Fields, larger than $GF(2^{2^3})$, have proven to be not useful for coding, due to the increased entropy of the coding vector overshadowing its benefits [5].

***GNUM (Generation-Number)*:** For a generation-based Network Coding scheme, it is essential to match a coded packet to its generation. If a receiver adds a received packet from generation x , into its decoding matrix for generation y , the decoding process can fail. To ensure the correct generation-matching between encoder and decoder, the *NC_Coefficients* option contains the generation-number *GNUM*, defined as a 4 bit unsigned integer field.

B. *NC_Coefficients*

After communicating the coding parameters, the sender can create and send coded payload. For successful decoding, the receiver has to know, which coefficients were used to encode the message's payload and match the generation's encoder and decoder. Our approach defines a new CoAP option *NC_Coefficients* to distinguish between coded payload and coding coefficients and handle the generation-matching. The *NC_Coefficients* option value is shown in Figure 7. The option value contains a field *COEFF*, for the coding coefficients, and a second field *GNUM*, representing the generations-number.

***COEFF (Coding Coefficients)*:** It would be possible to include the coefficients at the start or end of the payload, as it is done in [5]. This approach works, if receiver and sender know about the number and position of coefficient bytes in the transmitted payload. The here presented solution, using the option feature of CoAP's specification, is much smoother. Payload and coefficients are strictly separated. The coefficients can be parsed independently into the decoder, enabling the usage of different decoders.

***GNUM (Generation-Number)*:** The generation-number *GNUM* is used to match encoder and decoder. Its definition in the *NC_Coefficients* option is equal to the definition in the *NC_Control* option. *GNUM*'s further usage will be presented in the following section.

C. Transfer scheme

The designed transfer scheme is here presented step by step, as shown in Figure 5. The scheme can be divided into the three phases: initialisation, transmission and termination.

1. *Phase - Initialisation:* The transfer starts with a request. In the here used example, shown in Figure 5, the client creates a get-requests for resource *res*. For better understanding, this paper presents the transfer scheme using a get-request example, but the scheme could also be used with CoAP's put or post method. Before transmitting the request, the client initialises a decoder with predefined parameters, regarding the client's computing power and memory.

In the here presented scheme, the coding parameters are chosen by the client. This is done due to the fact, that in IoT-topologies, clients usually face stronger constraints than servers. A late parameter negotiation, as done in CoAP's blockwise transfer [2, Section 3.1.], is still possible, but not further presented here.

The coding parameters and the first generation's number are forged into an *NC_Control* option, which is added to the outgoing, confirmable (*CON*) message. The generation-number *GNUM*, of the first generation, is set to 1. Generation-number 0 is reserved for later usage in the termination-phase.

2. *Phase - Transmission:* When receiving a confirmable message from the client, the server processes the included *NC_Control* option. The server creates an encoder for generation *GNUM*, with the coding parameters *NOS*, *SOS* and *FFE*, obtained from the option.

As stated in CoAP's specification [16, Section 4.2.], a received confirmable message must be acknowledged by a response. There are two possible response schemes allowed in CoAP. The server can either confirm a message by sending an acknowledgement with piggybacked payload [16, Section 5.2.1.], or first send an empty message, just to inform the target about the reception, then satisfy the request in a second message. The latter is called a separate response [16, Section 5.2.2.]. The here proposed scheme combines ideas of both response-approaches. First, a piggybacked acknowledgement (*ACK*) is send, fulfilling the protocol's specification, then a number of non-confirmable responses (*NON*) are following. As specified in [16, Section 5.3.2.], the piggybacked response has to echo the request's message-ID and token, while separated responses just have to match the token.

In a regular CoAP transmission, the client would drop the request from its queue, after the confirmable message was acknowledged. Network Coded requests are not dropped by the client, until the last generation is decoded. This is handled similar in blockwise transfer.

After sending the first acknowledgement, the server keeps sending non-confirmable messages, containing an *NC_Coefficients* option and coded payload, created by the encoder for the corresponding generation *GNUM*. The server keeps sending coded packets, until the client orders the next generation, or a maximum transmissions counter is exceeded, which protects the sender in case of a client's breakdown.

When receiving a response, containing an *NC_Coefficients* option, the client pushes the extracted coefficients and the message's coded payload into the generation's decoder, indicated by *GNUM*. If a generation is successful decoded, the client can write down the restored generation-data to the resource's local representation. The generation-number indicates the data's position in the resource. The client then orders the next generation, by sending another confirmable message to the server. This message includes the next generation number in the added *NC_Control* option. After

	s_0	s_1	s_2	s_3	s_4
$p_{1,1} =$	1	0	0	0	0
$p_{1,2} =$	0	1	0	0	0
$p_{1,3} =$	0	0	1	0	0
$p_{1,4} =$	0	0	0	1	0
$p_{1,5} =$	0	0	0	0	1
$p_{2,1} =$	0	0	1	1	0
$p_{2,2} =$	0	1	0	1	1

Figure 8. Systematic Coding.

receiving this message, the server starts sending coded payload from the next generation, resulting in a loop of transmission-phases, until the termination-phase is inducted.

3. *Phase - Termination:* The termination-phase is started by the server, when processing the last generation of the requested resource's data. This is communicated from the server to the client by using a *GNUM* of 0 in the responses *NC_Coefficients* option. The server keeps responding, as done in the transmission-phase, until a stop-message from the client is received. After decoding the last generation, the client sends a confirmable message to the server, including a *NC_Control* option, with *GNUM* also set to 0. This message is understood as a stop-message. The here presented usage of the generation-number is similar to the usage of the *more* flag in CoAP's blockwise transfer. When receiving the stop-message from the client, the server acknowledges it and stops creating coded packets. The transfer is completed.

D. Coding Scheme

This work uses a basic random linear Network Coding scheme with systematic coding. Lucani et al. presented systematic coding in [11]. The idea can be added to most coding schemes. Systematic coding transmits a generation in two stages. First, every symbol, corresponding to one data-chunk, is transmitted uncoded. If no losses occur, systematic coding results in almost no coding complexity, and is similar to a regular UDP session.

In the second step, redundant, coded combinations are send, to make up possible losses. The second step is equal to regular random linear Network Coding and could be substituted by other coding schemes. Figure 8 shows a systematic coding approach for a Generation with size $n = 5$, including the symbols s_k . If, for example, the packets $p_{1,3}$ and $p_{1,4}$ of the uncoded transmission step $p_{1,k}$ were lost, the receiver could still decode the generation, due to redundant data in the second step's packets $p_{2,1}$ and $p_{2,2}$.

IV. IMPLEMENTATION

The proposed design was implemented using cantcoap [12], which is an open source library, written in C++. It offers a basic set of functions, to develop a CoAP application. The main functionality is encoding and decoding of CoAP-PDUs. Blockwise transfer is not implemented, a retransmission scheme for lost messages is also missing. This work includes the missing features, using a layer-based approach, shown in Figure 9. The application's structure is modelled after CoAPthon [18], another open source CoAP library, written in Python.

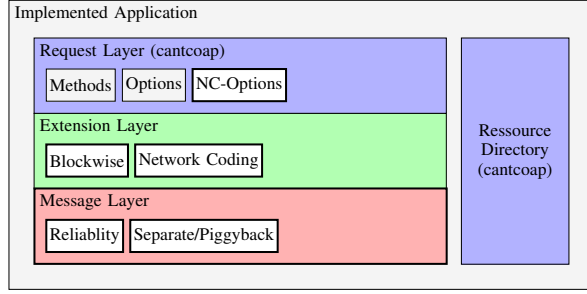


Figure 9. Structure of the implemented application, cf. [18].

The Network Coding extension includes Kodo [13], which is a free for research Network Coding library, published under a custom license. Kodo is written in C++. There are high level bindings in the *kodo-cpp* project, offering easy access to basic functionalities. The here presented application needs low level access, to separate coded payload and coding coefficients. Thus the *kodo-rlnc* project is used, enabling to the required methods.

When a message is processed, the Network Coding Layer will handle the new Network Coding options *NC_Control* and *NC_Coefficients*, as presented in Sections III-A and III-B. The block options are handled, according to the CoAP specification [16].

V. EVALUATION

A. Setup

For this evaluation, the implemented client and server were run on the loopback interface of a consumer Notebook, using a i5-3360M CPU @ 2.80GHz processor and 8GB Memory. Ubuntu 14.04.4 LTS was selected as operating system, using an unmodified, generic Linux 3.19.0-58 kernel. A lossy environment was emulated using netem [4], by adjusting packet loss rate and delay. Packet corruption and reordering are not considered, to focus on a comparison of the effects, induced by delay and loss.

In all measurements, the transmitted resource had a length of 65536 bytes, containing random characters. To reduce the measurement's complexity, the default retransmission timeout *ACK_TIMEOUT* was reduced, from two second, as suggested in the CoAP specification [16, Section 4.8], to one second.

For every shown combination of packet loss, delay and transfer scheme parameters, 50 replications were performed. The client's completion time was measured. The presented results are the arithmetic means of these measurements. If the measurements include packet loss, the 95 percent confidence intervals are shown in the corresponding figures.

B. Blockwise Transfer

The completion time of the blockwise transfer is influenced by the chosen blocksize, which depends on the *SZX* parameter in the block option value. The blocksize is defined as 2^{4+SZX} , allowing blocksizes of 16, 32, 128, 256, 512, and 1024 bytes.

Figure 10 shows the effect of delay on the client's completion time for different blocksizes. A smaller blocksize means, the requested resource has to be split in more chunks,

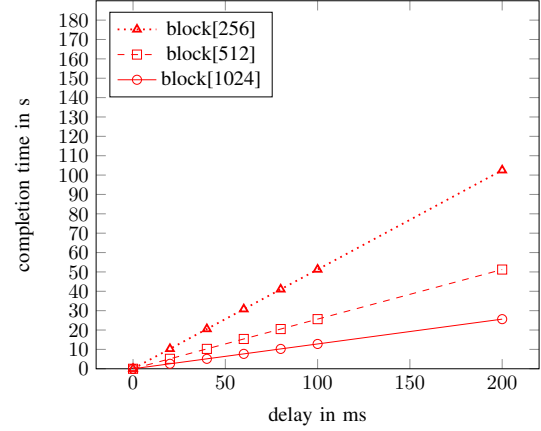


Figure 10. Effect of delay on blockwise transfer for different blocksizes, fixed 0% packet losses.

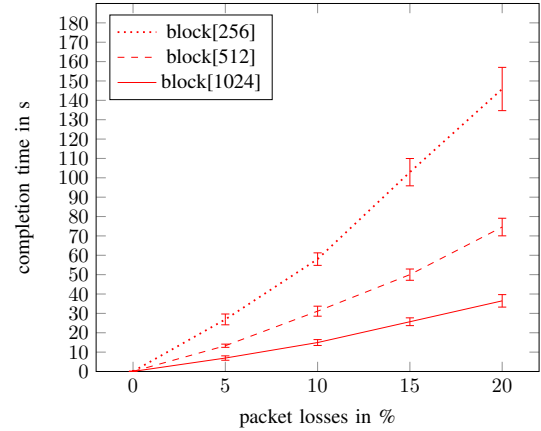


Figure 11. Effect of losses on blockwise transfer for different blocksizes, fixed 0ms delay.

leading to an increased number of transmissions. Due to the request-acknowledgement system of blockwise transfer for each message, every transmitted chunk is influenced by the delay twice. For the here chosen example of a get-request, the first delay is happening on the chunk-request from client to server, the second delay occurs on the server's response. A bigger blocksize indicates that a lower number of packets must be transmitted, leading to less occurrences of delay, resulting in a faster completion time. Due to the dependency between number of transmissions and number of delay-occurrences, a longer delay forces a linear increase in completion time.

The effect of message-loss is shown in Figure 11. If a larger blocksize is chosen, the resource is split in less chunks, which must be transmitted. This indicates less chunk-requests by the client, leading to a lower number of losses, resulting in less re-transmissions. An increased percentage of losses is followed by an exponential increase in completion time, due to the fact, that losses can happen independently on the request-message and the server's response-message. The presented measurements are done in a single-hop setup. The effect of message-loss is even stronger in a multi-hop topology

C. Network Coded Transfer

In the Network Coding extension, there are two parameter directly affecting the transfer duration: *NOS* and *SOS*. The *FFE* parameter does not directly influence the completion time, but is also important for the transfer scheme. A high enough *FFE* must be chosen to guarantee successful decoding with high probability, depending on the *SOS* and *NOS* parameters. Following [5], *SOS* and *NOS* shall be chosen as high as possible, to allow maximum throughput. The here shown parameter combinations are based on Kodo example applications [13]. For a deployable implementation of the here proposed design, the parameters are restricted by constraints of the hardware used and should be considered carefully.

In a generation-based Network Coding scheme, the client requests generations of data-chunks, instead of single chunks. Changes to *SOS* and *NOS* have a similar effect on the completion time, regarding losses and delay. This is based on the relation between the resource-size s_r , the required generations g_t ,

the generation-size s_g and the coding parameter *SOS* and *NOS*. The relation is given in the following equation:

$$g_t = \frac{s}{s_g} = \frac{s}{(SOS \cdot NOS)} \quad (4)$$

To successful transfer a file of size s_r , at least g_t generations have to be transferred. Each of this generation contains s_g bytes, defined by *NOS* and *SOS*. Higher coding parameter lead to a larger generation-size, resulting in less required generations and thus in a lower effect of losses and delay. The reverse effect is given, when using lower coding parameter.

In Network Coding, responses do not need to be retransmitted. The responding sender does not realise, if a coded packet gets lost. Due to the rateless nature of Network Coding, the sender can create a new coded packet and keep transmitting. To ensure a successful transfer, only the loss of a new generation-request must trigger a retransmission. The less generations needed, the lower is the effect of losses and delay. This is shown in the Figures 12 and 13. The parameter combinations [*SOS*/*NOS*], which result in larger generation-sizes, achieve lower completion times. A larger generation-size is the key for completion time reduction, not only to resist message-losses, but also to reduce delay-induced effects.

D. Comparison

The direct comparison, shown in Figure 14, reflects the previous results. Blockwise transfer is limited, due to its acknowledgement system. The shortest completion time for blockwise transfer can be achieved, by using the maximum blocksize of 1024 bytes. When choosing a parameter combination, which results in a generation-size, larger than 1024 bytes, the advantages of Network Coding are obvious. But even if the generation-size is smaller than the maximum blocksize, the coding extension results in a shorter completion time. This can be seen, when comparing nc[64/8] and block[1024] in Figure 14. The generation-size of nc[64/8] is only 512 bytes, but the scheme still performs significant better under losses. This demonstrates the beneficial loss resistance of the here presented Network Coding extension.

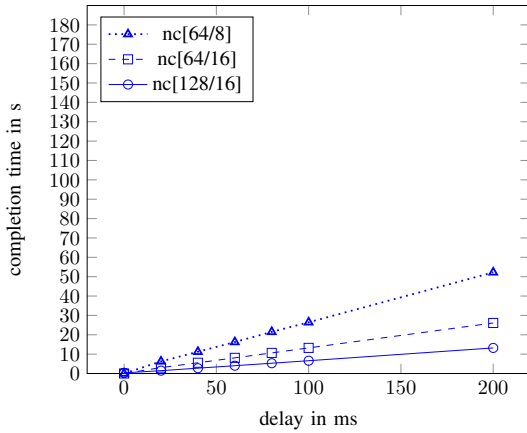


Figure 12. Effect of delay on Network Coding transfer for different *NOS* and *SOS* parameter, fixed 0% packet losses.

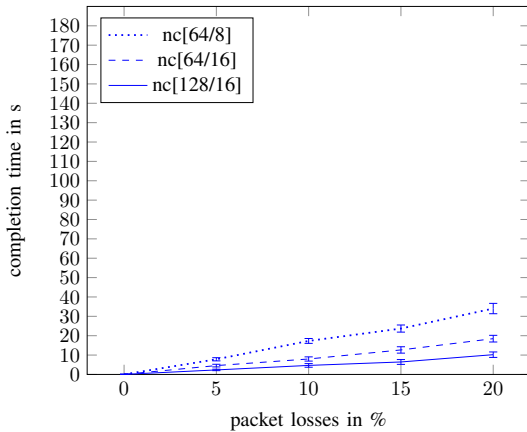


Figure 13. Effect of losses on Network Coding transfer for different *NOS* and *SOS* parameter, fixed 0ms delay.

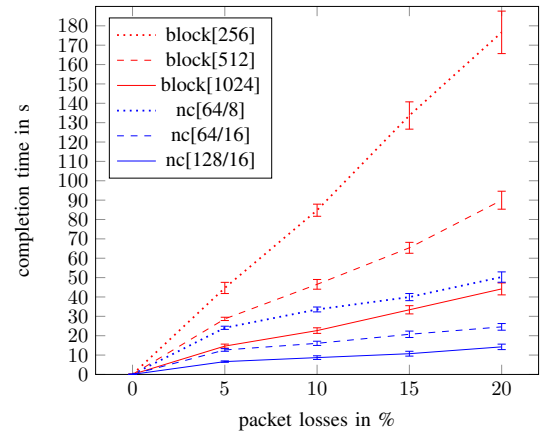


Figure 14. Comparison between blockwise transfer and Network Coding transfer in a lossy environment, fixed 60ms delay.

VI. DISCUSSION

One critical aspect of the given evaluation is the use of a limited CoAP library. The implemented application is kept simple and not designed for transmission speed. This leads to inaccuracies, when comparing the here stated completion times with other publications, using different CoAP implementations. It does not reduce the quality of the presented evaluation, because both compared transfer schemes are equally affected by the application's performance.

The evaluation was done on Notebooks, which have more memory and computing power, compared to most IoT-devices. The results have to be verified again for constrained devices, especially when the devices are so limited in processing power, that the Network Coding processing consumes noticeable time. This leads to a real world deployment, which will be done in future work.

This work features a pretty basic Network Coding scheme, using systematic, random linear Network Coding. Several other coding schemes have been proposed, which could improve the here presented results. Those schemes are namely triangular Network Coding [14] and different types of sparse coding, including precharging schemes, which are available in [13]. Also the effects of systematic coding in lossy environments have to be further investigated.

The here presented Network Coding approach shows advantages over the basic blockwise transfer. It has to be discussed, if the blockwise scheme itself can be improved. The simplest approach would be to extend the block options SZX field, allowing bigger block sizes. Another approach was done by Järvinen et al, as stated in the related work, Section II-C. The group researched alternative congestion control mechanisms for CoAP [8]. Both approaches tackle a similar problem in packet delivery. A comparison between both solutions would be interesting.

VII. CONCLUSION & FUTURE WORK

This paper proposed a smooth Network Coding extension for CoAP, to handle large resource transfer. The presented design is based on two new CoAP options, which satisfy the specification's option format. The theoretical advantages of the new extension over CoAP's existing blockwise transfer were verified on the basis of measurements. The here presented transfer scheme has proven to reduce the client's completion time in case of losses and delay. If a targeted hardware is able to effectively handle generations, larger than the maximum blocksize of 1024 bytes, the Network Coding extension can reduce the completion time for large resource transfer.

The next step will be a deployment of the developed CoAP application on real IoT-devices. Based on the here proposed solution, we will test and evaluate the throughput and requirements of different Network Coding schemes for the deployed hardware.

ACKNOWLEDGEMENTS

This work was supported by the "Stifterverband für die Deutsche Wissenschaft" (H170 5701 5020 20951).

REFERENCES

- [1] R. Ahlswede, N. Cai, S.-Y. R. Li, and R. W. Yeung, "Network information flow," *Information Theory, IEEE Transactions on*, vol. 46, no. 4, pp. 1204–1216, 2000.
- [2] D. C. Bormann and Z. Shelby, "Block-wise transfers in CoAP," Internet Engineering Task Force, Internet-Draft draft-ietf-core-block-20, Apr. 2016. [Online]. Available: <https://tools.ietf.org/html/draft-ietf-core-block-20>
- [3] A. F. Dana, R. Gowaikar, R. Palanki, B. Hassibi, and M. Effros, "Capacity of wireless erasure networks," *Information Theory, IEEE Transactions on*, vol. 52, no. 3, pp. 789–804, 2006.
- [4] T. L. Foundation. (2016) netem. [Online]. Available: <http://www.linuxfoundation.org/collaborate/workgroups/networking/netem>
- [5] J. Heide, M. V. Pedersen, F. H. Fitzek, and M. Médard, "On code parameters and coding vector representation for practical rnc," in *Communications (ICC), 2011 IEEE International Conference on*. IEEE, 2011, pp. 1–5.
- [6] T. Ho, R. Koetter, M. Médard, D. R. Karger, and M. Effros, "The benefits of coding over routing in a randomized setting," 2003.
- [7] T. Ho, M. Médard, R. Koetter, D. R. Karger, M. Effros, J. Shi, and B. Leong, "A random linear network coding approach to multicast," *Information Theory, IEEE Transactions on*, vol. 52, no. 10, pp. 4413–4430, 2006.
- [8] I. Järvinen, L. Daniel, and M. Kojo, "Experimental evaluation of alternative congestion control algorithms for constrained application protocol (coap)," in *Internet of Things (WF-IoT), 2015 IEEE 2nd World Forum on*. IEEE, 2015, pp. 453–458.
- [9] R. Koetter and M. Médard, "An algebraic approach to network coding," *IEEE/ACM Transactions on Networking (TON)*, vol. 11, no. 5, pp. 782–795, 2003.
- [10] S.-Y. R. Li, R. W. Yeung, and N. Cai, "Linear network coding," *Information Theory, IEEE Transactions on*, vol. 49, no. 2, pp. 371–381, 2003.
- [11] D. E. Lucani, M. Médard, and M. Stojanovic, "Systematic network coding for time-division duplexing," 2010.
- [12] A. Mills. (2016) cantcoap. [Online]. Available: <https://github.com/staropram/cantcoap>
- [13] M. V. Pedersen, J. Heide, and F. H. Fitzek, "Kodo: An open and research oriented network coding library," in *Networking 2011 Workshops*. Springer, 2011, pp. 145–152.
- [14] J. Qureshi, C. H. Foh, and J. Cai, "Optimal solution for the index coding problem using network coding over $GF(2)$," in *Sensor, Mesh and Ad Hoc Communications and Networks (SECON), 2012 9th Annual IEEE Communications Society Conference on*. IEEE, 2012, pp. 209–217.
- [15] I. Salhi, Y. Ghamri-Doudane, S. Lohier, and G. Roussel, "Reliable network coding for zigbee wireless sensor networks," in *Mobile Adhoc and Sensor Systems (MASS), 2011 IEEE 8th International Conference on*. IEEE, 2011, pp. 135–137.
- [16] Z. Shelby, K. Hartke, and C. Bormann, "The constrained application protocol (coap)," RFC 7252, IETF CoRE Working Group, 2014. [Online]. Available: <http://www.ietf.org/rfc/rfc7252.txt>
- [17] J. K. Sundararajan, D. Shah, M. Médard, M. Mitzenmacher, and J. A. Barros, "Network coding meets tcp," in *INFOCOM 2009, IEEE*. IEEE, 2009, pp. 280–288.
- [18] G. Tanganelli, C. Vallati, and E. Mingozzi, "Coapthon: Easy development of coap-based iot applications with python," in *Internet of Things (WF-IoT), 2015 IEEE 2nd World Forum on*. IEEE, 2015, pp. 63–68.