HAL
open science

A Semantic Firewall for Content-Centric Networking

David Goergen, Thibault Cholez, Jérôme François, Thomas Engel

To cite this version:

David Goergen, Thibault Cholez, Jérôme François, Thomas Engel. A Semantic Firewall for Content-Centric Networking. 2013 IFIP/IEEE International Symposium on Integrated Network Management (IM2013): Mini-Conference, May 2013, Ghent, Belgium. pp.478-484. hal-00829615

HAL Id: hal-00829615

https://hal.science/hal-00829615v1

Submitted on 3 Jun 2013

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

A Semantic Firewall for Content-Centric Networking

David Goergen, Thibault Cholez, Jérôme François, Thomas Engel

Interdisciplinary Centre for Security, Reliability & Trust - University of Luxembourg
Email: {firstname.name}@uni.lu

***Abstract*—Content-Centric Networking (*CCN*) is a promising routing paradigm for content dissemination over a future Internet based on named data instead of named hosts. The *CCN* architecture has aspects that provide more scalability, security, collaborative and pervasive networking. However, several key components that secures the current Internet are still missing in *CCN*, in particular a firewall able to enforce security policies. We provide a comprehensive study of *CCN* security requirements from which we design the first *CCN*-compliant firewall, including syntax and definition of rules. In particular, based on *CCN* features, our firewall can filter packets according to both their authentication and the semantics of the content name. We also provide a performance evaluation of our prototype.**

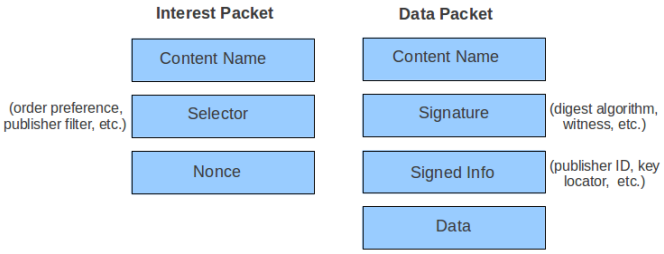

Fig. 1: *CCN* packets structure: Interest and Data

I. Introduction

Developed at PARC by Van Jacobson and his team [1], *CCN* is a proposition of Information-Centric Networking architecture. Building on the observation that today's communications are more oriented towards content retrieval (web, P2P, etc.) than point-to-point communications (VoIP, IM, etc.), *CCN* proposes a radical revision of the Internet architecture, switching from named hosts to named data to best match its current usage. In a nutshell, content is addressable, routable, self-sufficient and authenticated, while locations no longer matter. Data is seen and identified directly by a routable name instead of a location (the address of the server). Consequently, data is directly requested at the network level. To improve content diffusion, *CCN* relies on nearby data storage to save bandwidth: every content – particularly popular ones – can be replicated and stored on any *CCN* node, even untrustworthy ones. People looking for particular content can securely retrieve it in a P2P-manner from the best locations available.

This new paradigm is promising and builds on experience gained from the current Internet. In particular, security aspects have been considered in the design, including, for example, the authentication of content. However, even if some building blocks exist to secure *CCN*, real security tools are missing. For example, nothing protects against a flooding attack, as we highlighted in our previous work [2] and nothing prevents a user from downloading malicious or forbidden content. In IP networks, firewalls are usually the first level of protection, enforcing the security policy defined by the administrator within the company's network. This paper proposes the use of firewalls for *CCN* security with the following contributions:

- identification of the security needs for a functional *CCN* architecture
- design of a *CCN* firewall with semantic features
- performance evaluation of the firewall

The next section presents the *CCN* architecture in details. Section III provides a use case analysis, leading to the definition of the firewall functionalities and the language for specifying rules. Section IV introduces the design of the firewall its implementation within the *CCNx*, while Section V is dedicated to the evaluation. Finally, we present the Related work in Section VI and conclude in Section VII.

II. CCN background

A. *CCN paradigm*

CCN has two main types of packets, Interest and Data, as seen in Figure 1. A user who wants to access a given piece of content sends out an Interest packet, specifying the name of the content to all its available faces. Faces can be anything capable of serving as medium for transmitting and receiving data. A node which receives this packet and that can "satisfy" the Interest then sends the corresponding Data packet to the face from which it received the Interest. By definition, *CCN* nodes are stateful and only forward Data on the back path if an Interest was emitted beforehand (Data cannot be pushed, this makes classical denial of service attacks inefficient).Data can only "satisfy" a specific Interest if the ContentName of Interest packet is a prefix of the Data packet. *CCN* names are defined in [1] as "opaque, binary objects composed of an (explicitly specified) number of components". This structure allows a fast and efficient prefix-based lookup similar to the IP lookup currently used. This new routing paradigm is based on plain-text hierarchical naming instead of regular host IP addresses which makes names directly intuitive and DNS useless.

CCN nodes are composed of three main table structures, which handle the forwarding of packets. On the arrival of an Interest packet on any given face, the engine performs a longest-match lookup on its structures and action is taken depending on the lookup result. The first structure to be searched is the Content Store (CS). It can be seen as a buffer

memory following the LRU scheme of past Data packets. It enables a fast retrieval of currently popular demands..

If there is no match in the CS, the lookup is launched on the next structure which the Pending Interest Table (PIT), which keeps record of Interests waiting to be resolved upstream by other content source(s). If a received Interest matches an entry in the PIT, the engine compares the faces recorded for that entry. If there is already one existing, no update is made. Otherwise, the face from which the Interest was emitted is simply added to the list of already waiting faces.

If no match is found in the PIT, the engine searches in its final structure: the FIB. The Forward Information Base keeps records of potential content source(s) and works similarly to its IP counterpart, except that it stores a list of possible providers for a given name rather than only a single source. If a match is found, the engine forwards the Interest.

B. CCN security scheme

The keystone of *CCN* security is the trust in the publisher. In fact, the paradigm shift of *CCN* makes every node capable of answering a Data request. To ensure the security of communications despite untrustworthy nodes, data is authenticated by its provider, not the connections it traverses (no end-to-end encryption). *CCN* strongly relies on cryptography to authenticate content so that users can clearly know who emitted the content and can discard material from untrustworthy sources and so avoid malware. Also, encryption is used to ensure privacy.

To securely authenticate content, *CCN* must bind the content name, the content itself and the content provider. To allow this, the following information is embedded in each *CCN* data packet: $Signature(Name, Content, SignInfo)$. SignInfo includes: cryptographic digest or fingerprint of publisher's key, key or key location. Key management is another issue often discussed. In [1], several solutions are discussed which range from a PKI to PGP-like web-of-trust.

So, the *CCN* architecture, and particularly the security layer, provide the means of securing a network. Based on IP use cases, we next define the features that a firewall should have in order to enforce an efficient security policy for *CCN*.

III. Firewall design

A. Use case analysis

In current firewalls like the open-source iptables [3], the main parameters taken in consideration when writing the filtering rules are the IP address of the hosts, the communication port (linked to the application) and the status of the connection (new, established, etc.). However, the previous section illustrated that such concepts are not available in the *CCN* world, making the design of a firewall challenging. Despite its interesting security features, *CCN* still needs to provide a way for network administrators to properly enforce their security policy.

To drive the design of our firewall and the requirements of the language, we first present a list of use cases describing the various security considerations that an administrator of a CCN network may have. We first recall some general use cases addressed by an IP firewall and analyse them with regard to their applicability to the *CCN* world. We then introduce *CCN*-specific considerations before defining the necessary features of our firewall.

1) IP firewall general use cases: One of the most important use cases addressed by firewalls is the filtering of protocols according to the network policy (*IP_UC1*). A *CCN*-firewall should be able to filter communications on the same principle, being able to differentiate the different kinds of traffic (web, mail, voip, p2p, etc.). For example, web traffic (http) may be authorized in the network while other protocols (ftp, p2p, etc.) are forbidden. A second crucial capability of an IP firewall is to filter traffic according to the status of the connection (*IP_UC2*), especially to differentiate traffic originating inside the network from unexpected traffic originating outside which is usually blocked to prevent attacks. The third capability is to filter traffic from known malicious IP networks, which are blacklisted in order to avoid malware infection (*IP_UC3*). The network administrator can prevent communications to different blocks of IP address. Finally, serving both for quality of service and security purposes, a firewall can also filter unusual aggressive inbound traffic (*IP_UC4*). Typically, in order to mitigate a DoS attack, a firewall can filter traffic from a host when the number of packets emitted per second exceeds a given threshold.

When considering the *CCN* paradigm, some of the aforementioned use cases derived from an IP firewall do not make sense and others must be adapted. In particular, the notion of "connection" is deprecated. *CCN* offers only pull-based connectivity: *Data* can only follow the path previously set by the *Interests*. Any pushed traffic will be discarded by the *CCN* stack and so does not need specific rules on the firewall. Some other use cases are still relevant but must be adapted. For example, filtering the traffic based on the protocol is mainly achieved based on the service port number, which is not relevant in *CCN*. Instead, alternative ways of filtering services based on content name should be defined. Also, filtering malicious network based on IP address blocks do not make sense in *CCN*. As an alternative, *Data* can be discarded according to the identity of their content provider.

2) CCN-specific use cases: Beyond the adaptation of usual IP firewall use cases to the *CCN* paradigm, a *CCN* firewall should also directly rely on *CCN* concepts to enforce the new security model proposed by *CCN*. In fact, a *CCN* firewall can filter content based on two new parameters:

- the content provider: each piece of content should be authenticated with its provider's signature,
- the content name: the key parameter used to route content, it is always given as a plain text string which should be intelligible.

As defined in [1], every *CCN Data* must be signed by the content provider. A *CCN* firewall must check the status of the content provider and filter those known to be untrustworthy or banned according to the network policy (*CCN_UC1*). Alternatively, some content may claim to come from a particular content provider but not be signed by it and so should also be discarded (*CCN_UC2*). Beyond security considerations, some well-known content providers whose content is not relevant within the company network can also be filtered, for example web sites providing video streaming can be blocked in the same way as malicious ones in *(CCN_UC1)*.

The second interesting aspect *CCN* is the fact that content name is mandatory and makes sense. This key field allows the firewall to perform new efficient filtering based on the content name (*CCN_UC3*). For instance, specific media types can be excluded from the network (.avi, .mp3, etc.). Some content with names including specific keywords can also be filtered. This approach can even be enhanced by a semantic analysis of the name. Both kinds of filtering (based on the content provider and on the content name) can be mixed (*CCN_UC4*) to create more fine-grained rules which can, for example, only allow a node to download executable content from few specific (trustworthy) providers. When filtering content based on the content name, the direction of the traffic is important. To avoid leakage of intellectual property through users' error, the firewall can prevent some content, whose type is for instance a document (.doc, .pdf, etc.), from being shared externally, while local transfers are allowed (*CCN_UC5*).

Finally, *CCN* nodes must also be protected against aggressive traffic rates to preserve the QoS (*CCN_UC6*). This can be achieved based on the maintained statistics of faces.

A *CCN* firewall can also change the default routing and caching policies of a *CCN* node. Based on the content name or content provider, a *CCN* node can decide to supercede the general rules defined by the *CCN* stack to only store (*CCN_UC7*) specific contents.

B. Firewall features

Based on the aforementioned use cases and on the design of *CCN*, in this section we propose an initial set of features for a *CCN* firewall.

Table I summarizes the following use cases, compares the IP and *CCN* worlds, and presents the corresponding firewall features and syntactic elements.

CCN_UC1) Filtering on content providers: The signature of the content provider is mandatory for each *CCN* Data packet. More precisely, each piece of content creates a mapping triple: $M_{(Name,Content,Provider)} = (Name, Content, Sign_P(N, C))$. The signature includes the public key, its location or a cryptographic digest encrypted with the publisher's private key. To filter traffic based on content provider, the firewall must retrieve information on the provider's key from the signature and compare it with a blacklist of keys. Alternatively, to improve security, a whitelist can also be created so that only content from authorised providers will pass through. Data packets that fail the check will not be transmitted to the face and the corresponding Pending Interest is deleted.

CCN_UC2) Filtering on bad signature: The validity of the signature can also be checked while processing the signature. If, being given the provider key, the content and its name, the firewall is not able to find the same fingerprint, the content can be discarded as its source cannot be checked.

CCN_UC3) Filtering on content name: As illustrated by Figure 1, both types of *CCN* packet include the name of the content as a plain-text information. By analysing the keywords composing the name using regular expressions, the firewall can filter content based on a blacklist of the different pieces of information composing the content name, that is: the name prefix, keywords and the file extension. Interest packets that fail the check will not be written in the Pending Interest Table and Data packets that fail will not be transmitted to the requesting face. The corresponding Pending Interest is deleted.

It would be useful to filter all content with a given semantic, for example, to prohibit paedophilia-related content. Thus, our firewall can consider additional words that are semantically close to those defined by the user in the firewall rules. The semantic distance allowed can also be defined by the user.

CCN_UC4) Composition of filters: The two previous parameters (content provider and content name) must be considered together to allow the firewall to use more complex rules that can filter, or authorize, specific content type or content name according to the provider.

CCN_UC5) Filtering on content direction: The firewall should be able to differentiate local traffic (*Data* published locally or under the company's prefix) from external traffic.

CCN_UC6) Filtering on heavy traffic: The *CCN* stack includes a strategy layer which is in charge of maintaining statistics for each active communication face in order to select the best available communication channel. Based on these statistics, the firewall can detect faces showing abnormal activity in case of bottleneck and prevent so Denial of Service. This can be achieved by decreasing the rate of Interests written in the PIT due to aggressive communications. Alternatively, as Interests are also used to control the traffic stream, like a TCP reception window, the firewall could overwrite the field of cumulative requested Interests to save bandwidth.

CCN_UC7) Filtering of stored data: A caching policy can easily be enforced by the firewall. To do this, the same rules written in the firewall language can be applied to the Content Store in addition to the PIT. More precisely, if a *CCN* Data packet is eligible to be stored in the CS according the the caching algorithm (most recent, most frequent, etc.), the firewall can then check the provider signature or the content name according to the defined rules before caching the Data.

C. Rules definition language

For the ease of use and to enhance the readability of the rules, we bases our syntax closely on that used in iptables. The firewall rules are defined according to the following grammar expressed using the Augmented Backus Naur Form [4].

```
rule = r_interest | r_data | r_face
r_interest = "interest" SP direction SP
             match_interest SP "pit" SP action
r_data = "data" SP direction SP match_data
         SP ["cs"|"pit"] SP action
r_face = "face" SP number

direction =  "*"|"int"|"ext"
action = "forward"|"drop"
match_interest = content_name
match_data = content_name SP provider
content_name = "*"|reg_exp
provider  = sign_check SP provider_sign
sign_check = "0" | "1"
provider_sign = "*"|first_sign *next_signs
first_sign = hex_value
```

TABLE I: Relationship between use cases, firewall features and language. The bold part in the rule example corresponds to the main syntactic element involved in the use case

IP_UC	CCN_UC	Feature name	Syntactic element	Rule example
IP_UC1	*CCN_UC3*	Filtering on content name	`content_name`	interest * \\@**game**|**play**|**fun**\\@ **15** pit drop
IP_UC2	-	-	-	-
IP_UC3	*CCN_UC1*	Filtering on content providers	`provider_sign`	data * * 0 **123456789ABCDEF;FFFF0000AAAA** pit drop
IP_UC4	*CCN_UC6*	Filtering on heavy traffic	`r_face`	**face 200**
-	*CCN_UC2*	Filtering on bad signature	`sign_check`	data * * **1** * pit drop
-	*CCN_UC4*	Composition of filters	`match_data`	data * \\@**game**|**fun**\\@ **0 0 123456789ABCDEF;FFFF0000AAAA** pit drop
-	*CCN_UC5*	Filtering on content direction	`direction`	interest **in** \\@\\.doc$\\@ 0 pit drop
-	*CCN_UC7*	Filtering of stored data	`cs`	data * * 1 * **cs** drop

```
17 next_signs = ";" hex_value
18
19 reg_exp = "" re_posix "" number
20 re_posix = <a standard posix regexp>
21 hex_value = 1*hex
22 SP = 1*%d32 ; one or more space characters
23 hex  =  "A" | "B" | "C" | "D" | "E" | "F"
24       | "a" | "b" | "c" | "d" | "e" | "f"
25       | digit
26 number = 1*digit
27 digit = "0" | "1" | "2" | "3" | "4" | "5"
28         | "6" | "7" | "8" | "9"
```

As shown, there are three possible rules: for filtering interests (`r_interest`), data (`r_data`) or faces (`r_face`). The last case allows the firewall to filter heavy traffic by limiting the global number of packets per second defined by `number` in line 6. For other types of rules, we can first filter on the direction using `int` or `ext` to indicate respectively whether the content has been requested by the node itself or some other node. Also, the actions are the same for these rules, *i.e.* `forward` or `drop` on the PIT table which means that the firewall will forward or drop specific *Interests* in line 3 of the language definition. This is equivalent to adding or not adding an *Interest* in the PIT table. In line 5, the firewall can decide to drop certain *Data* packets. At a first sight, it might appear unfair to drop such *Data* since the corresponding *Interest* must have been accepted previously. However, an *Interest* can return *Data* with a different name when using dynamic generation of content upon request, as mentioned in Section II. In addition, a rule on *Data* can also control the caching policy by prefixing the action with `cs`. An *Interest* can only be matched according to the requested content (line 10). It is a standard POSIX regular expression delimited by `@` where `number` (line 19) describes the level to consider for semantically extending the regular expression.

Content, may also be filtered according to the provider by checking if the signature is valid (`sign_check`) and/or to the provider itself, identified by its signature, `provider_sign`, which is a sequence of hexadecimal characters. As in the grammar, we use space characters (`SP`) as seperators. There also several rules using the character `*`, which is a shorthand for a regular expression matching everything, avoiding the need to use the delimiter `@`.

In Table I, examples show how to construct rules related to the different use cases. For example, in *CCN_UC1*, *Data* signed with the keys *123456789ABCDEF* and *FFFF0000AAAA* are blacklisted. To help in understanding the concept of direction, the example in the table means that the node where the firewall is instantiated cannot share files whose extension is *.doc*, as the firewall will drop corresponding interest if they are associated with internal content (*in*). However, the node can forward interests about doc files of other nodes because the *ext* restriction is not specified.

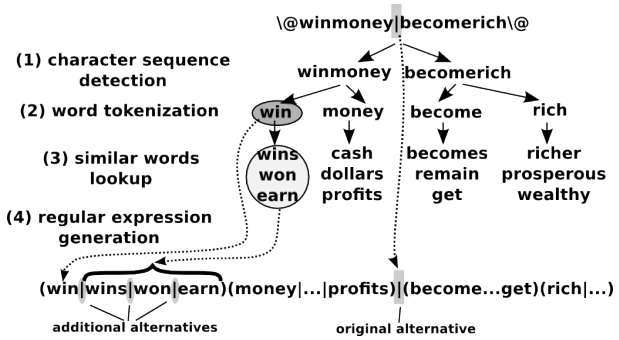

Fig. 2: Semantic extensions – short example

IV. Firewall architecture

To implement the features defined in the language, the firewall provides tools, modifies the CCNx [1] library and relies on a deployment strategy described in this section.

A. Semantic preprocessing

As highlighted in previous sections, filtering on content name uses regular expressions which can also contain meaningful words like `money`, `game`, `humour`, etc. Assuming the goal is to filter names like `winmoney` or `becomerich` which, in the current Internet, could be used in a URL to attract people to malicious websites hosting malware, the semantic extension is also able to filter related content names like for example: `wineuros`, `windollars`, `earnmoney`, `bericher`, etc. This is essential, as *CCN* provides a great freedom in naming and announcing content. To accomplish this, there are four steps as highlighted in Figure 2:

1) Sequences of 3 or more characters are extracted. This extracts only potentially meaningful words by discarding digits or regular expression symbols like `{, }` or `^`, as well as irrelevant very short sequences of characters.
2) Sequences of characters are segmented as real human-readable words
3) For each extracted word, a list of similar ones is created

4) For each list of similar words, the regular expression with alternatives together with the original word, replacing the original word in the original regular expression. This is illustrated in Figure 2 where the original pipe character (surrounded by a gray rectangle) is preserved.

For word segmentation, the technique proposed in [5] is employed. It finds the optimal word segmentation by successively dividing the sequence of characters into two parts, while multiplying the probability of each word corresponding to its frequency in text samples.

Searching for similar words is done through DISCO [6] relying also on text samples like Wikipedia. It computes the relatedness of two words $w1$ and $w2$ by finding the number of co-occurrences of these words with a third one $w3$. Assuming all possible $w3$ which occur near to $w1$ and $w2$ (two intermediate words maximum), the relatedness is computed using the mutual information metric. Using the same method, DISCO is able to get the n words most similar to another.

Therefore, when a strictly positive number n is specified after a regular expression (`number` at line 19), this leads to the generation of the n most similar words to each word extracted at step 2 in the previously described process. It is important to note that some irrelevant word like `abcde` might not have been filtered by the first step but in DISCO will not find any similar word and so no alternatives will be generated in the final regular expression.

As such semantic algorithms are resource consuming, they cannot be executed when the firewall is running. Therefore, it is done as a preprocessing step in which the original rules are read and extended.

B. Integration within the CCNx library

Given the *CCN* stack scheme from [1], our firewall works within the Security layer and directly processes content chunks. The Security layer as defined in [1] handles authentication of the incoming chunks and our firewall can request signature verification, which the Security layer will handle, giving the result to our implementation. We integrated the code of our firewall within the *CCN* daemon so that it is triggered before any modification of the *CCN* internal tables (PIT and CS). This requires only minor changes to the *CCN* standard implementation, and provides flexibility for further updates. Our firewall captures a packet once it arrives from a face and then applies any given rule on it before letting the standard process update the tables. Most rules that were defined in the previous section are applied by filtering the PIT table.

C. Management

One of the first things to do when configuring an IP firewall is to set the default security policy that will be applied in the absence of matching rules. The settings range from the most restrictive rule: *deny all, accept on match*, to the most permissive rule: *accept all, deny on match*. For our *CCN* firewall, we also provide the administrator with such configurations. The recommended default security policy is to deny all, except packets signed with the public key of the network administrator. This single exception provides a way for a node to always have its firewall rules up to date.

To tackle security issues created because of node mobility in companies (laptop computers, smartphones, etc.), we recommend a deployment of the firewall on each node rather than only on nodes providing Internet connectivity. Upon installation, the firewall software is given the public key of the network administrator to be able to authenticate its content. This allows the rules implementing the security policies of the company to be updated on a regular basis. Firewall rules are defined as content that is requested by the firewall periodically. The rules are applied if properly signed by the network administrator.

V. Experiments

A. Performance evaluation

To test our implementation we created an architecture consisting of six *CCN* nodes arranged and configured such that the packets need to go through all nodes to be delivered. Each node is a standard workstation with an Intel Pentium 4 CPU running at 3 GHz and 2 GB of RAM. For the evaluation, we used the ccnx 0.6.0 release as a reference.

One node named "Content provider" provides the content used for our evaluation, some simple raw files. The intermediate nodes served merely as routers without caching repositories. Their function was only to allow the "Consumer" node to reach the content provider. Finally the last node on the path was the node requesting our testing content. This was also our measuring point for our evaluation. Our firewall implementation was deployed on all the nodes, following our recommended deployment strategy. We performed two evaluations based on this setup, each time starting *CCN* on all nodes with the same route configuration and requesting the same content.

In a first attempt to evaluate the performance of our implementation, we measured the end-to-end transfer time for two different binary files with sizes of 500 MB and 1 GB respectively. We requested the files several times, but each time we increased the number of rules to see if there is an impact on transmission time. The rules were designed so that the Data packet was accepted when matching the very last rule so that we could be sure that all were processed. The results obtained from this evaluation are given in Figure 3. It shows that there is no visible impact when we apply our firewall to the standard *CCN* implementation, even on the most demanding configuration (1000 rules on each node, 1GB file).

A second evaluation was performed in order to compare the performance of a standard *CCN* implementation with one including our firewall. This experiment was performed several times to obtain significant values for a later comparison with our implementation. For each observation, we transferred the same binary file of 500 MB to obtain a mean transfer time. We then performed several experiments, each with a fixed number of 1000 rules, using our implementation to obtain values to compare with the standard *CCN* implementation.

Our obtained results are shown in Figure 4. We measured the end-to-end transfer time from the content provider to the consumer node. We checked whether the obtained values

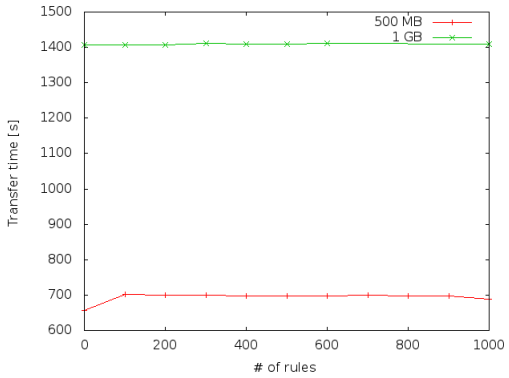

Fig. 3: Impact of the number of rules on the transfer time

TABLE II: Observation results

	Standard CCN	Firewall modification
mean μ	684.9718	697.1672
std. deviation σ	1.3463	1.5437

represent expected results by applying a normal distribution, calculating the mean transfer time and the standard deviation for both setups. The results are shown in Table II.

As can be seen in the graph, both the clean *CCN* implementation and our modification to it follow a normal distribution, as confirmed by applying Chi-square and KS-test on both measured sets, obtaining a confidence level of 91% for our implementation and 85% for the standard *CCN* implementation. From our measured observation we can deduce that the average overhead on the transfer time resulting from our implementation is the difference between both averages: $697.17 - 684.97 = 12.20$ s, which corresponds to 1.75%.

VI. Related Work

A. Security of Content-Centric Networking

So far, there has been little investigation on the security on *CCNs*. We currently lack of knowledge of possible security issues related to *CCN*. In particular, due to stateful routers (as the route between a content and the requester has to be memorised) network devices are exposed to new threats that can be exploited for attacks. Attackers may disturb the management of *CCN* nodes' tables (PIT, FIB, CS) which impact their performance and quality of service. This can lead to new possible DoS attacks or to malicious monitoring, which must be detected and mitigated. Tobias Lauinger [7] identifies several attacks related to caches, in particular denial-of-service attacks against *CCN* routers, but he only investigates one of these, "cache snooping" that enables attackers to efficiently monitor the content retrieved by their direct neighbours. Smetters et al. [8] of PARC propose authenticating the links between names and content, in addition to names and content themselves, in order to identify trustworthy content and avoid malware diffusion. However, the necessary PKI is hard to set up and if a provider becomes malicious, revocation remains a major problem that must be addressed in future work.

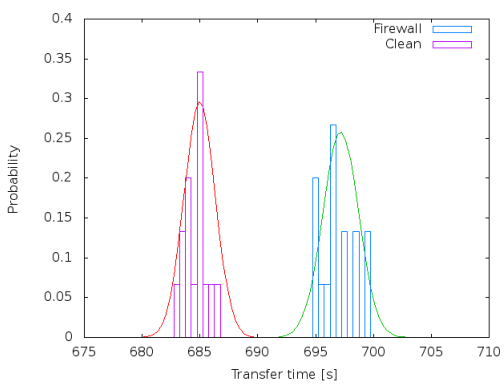

Fig. 4: Impact of a 1000-rules firewall on the transfer time

B. Firewalls

As networks expanded, security concerns increase in parallel, leading to the creation and deployment of defensive measures. Due to that, firewalls appeared in eighties and their popularity skyrocketed in the nineties. The historical evolution of firewalls during this period is described in [9]. Firewalls are generally characterized by the level on which they operate, such as the IP or application level, and by the way they process packets. A common differentiation is between stateless and stateful [10]. While the stateless approach is quite simple, trying to match data (*e.g.* a packet) to a pattern, the stateful approaches keep track of the current connections, applying different policies depending on the state of a connection. Stateless firewalls require de facto less resources and are so good candidates for deployment at the frontline. That is why in this paper, we have investigated this option in the context of *CCN* by designing a dedicated layer 3 stateless firewall.

Having to deal with more threats and protocols, recent advances in firewall aim to improve their efficiency by determining incoherency in policies [11] and optimizing rule matching order [12], while [13] proposes a dedicated approach for iptables [3]. Moreover, recent trends include protocol or application-specific firewalls, for example for SIP [14].

VII. Conlusion

In this paper, we proposed the first firewall dedicated to the *CCN* paradigm, from the use case study of the security needs of a *CCN* administrator, to the grammar definition and the implementation within CCNx. Since this paradigm relies on content-based routing, where the name of the content and its authentication are key aspects, we used these to provide innovative features relative to a regular IP firewall. In particular, we proposed to use semantic tools. Performance evaluation is very promising, since the message throughput degradation is lower than 2%. In future work, we plan to investigate potential optimization, for example by building on rule reordering research for standard firewalls or using bloom filters to accelerate rule lookups.

Acknowledgment

This work was partly funded by BUTLER and IoT6 FP7 EU projects under the grant agreements 287901 and 288445.

REFERENCES

[1] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. H. Briggs, and R. L. Braynard, "Networking named content," in *Proceedings of the 5th international conference on Emerging networking experiments and technologies*, ser. CoNEXT '09. New York, NY, USA: ACM, 2009, pp. 1–12.

[2] D. Goergen, T. Cholez, J. François, and T. Engel, "Security monitoring for content-centric networking," in *International Workshop on Autonomous and Spontaneous Security*, ser. SETOP. Springer-Verlag, 2012.

[3] "The netfilter.org iptables project." [Online]. Available: http://www.netfilter.org/projects/iptables/index.html

[4] D. H. Crocker and P. Overell, "Augmented BNF for Syntax Specifications: ABNF," Internet RFC 4234, 2005.

[5] T. Segaran and J. Hammerbacher, *Beautiful Data: The Stories Behind Elegant Data Solutions*. O'Reilly Media, 2009, ch. 14.

[6] P. Kolb, "DISCO: A Multilingual Database of Distributionally Similar Words," in *KONVENS 2008 – Ergänzungsband: Textressourcen und lexikalisches Wissen*, 2008.

[7] T. Lauinger, "Security & Scalability of Content-Centric Networking," Master's thesis, TU Darmstadt, September 2010. [Online]. Available: http://tubiblio.ulb.tu-darmstadt.de/46912/

[8] D. Smetters and V. Jacobson, "Securing Network Content," PARC, Tech. Rep., October 2009.

[9] F. Avolio, "Firewalls and Internet security, the second hundred (Internet) years," *The Internet Protocol Journal*, vol. 2, no. 2, 1999.

[10] A. X. Liu, "A model of stateful firewalls and its properties," in *Proceedings of the 2005 International Conference on Dependable Systems and Networks*. IEEE Computer Society, 2005.

[11] E. S. Al-Shaer and H. H. Hamed, "Discovery of policy anomalies in distributed firewalls," in *INFOCOM*, 2004.

[12] H. Hamed, A. El-Atawy, and E. Al-Shaer, "On dynamic optimization of packet matching in high-speed firewalls," *IEEE Journal on Selected Areas in Communications*, vol. 24, no. 10, pp. 1817–1830, 2006.

[13] R. Marmorstein and P. Kearns, "A tool for automated iptables firewall analysis," in *Proceedings of the annual conference on USENIX Annual Technical Conference*, ser. ATEC '05. USENIX Association, 2005, pp. 44–44.

[14] A. Lahmadi and O. Festor, "SecSip: a stateful firewall for SIP-based networks," in *Proceedings of the 11th IFIP/IEEE international conference on Symposium on Integrated Network Management*, ser. IM'09, 2009, pp. 172–179.