RICE: Remote Method Invocation in ICN

Michał Król
UCL
m.krol@ucl.ac.uk

Karim Habak
Georgia Tech
karim.habak@gatech.edu

David Oran
Network Systems Research & Design
daveoran@orandom.net

Dirk Kutscher
Huawei
dirk.kutscher@huawei.com

Ioannis Psaras
UCL
i.psaras@ucl.ac.uk

ABSTRACT

Information Centric Networking has been proposed as a new network layer for the Internet, capable of encompassing the full range of networking facilities provided by the current IP architecture. In addition to the obvious content-fetching use cases which have been the subject of a large body of work, ICN has also shown promise as a substrate to effectively support remote computation, both pure functional programming (as exemplified by Named Function Networking) and more general remote invocation models such as RPC and web transactions. Providing a unified remote computation capability in ICN presents some unique challenges, among which are timer management, client authorization, and binding to state held by servers, while maintaining the advantages of ICN protocol designs like CCN and NDN. In this paper we present a unified approach to *remote function invocation in ICN* that exploits the attractive ICN properties of name-based routing, receiver-driven flow and congestion control, flow balance, and object-oriented security while presenting a natural programming model to the application developer.

CCS CONCEPTS

• **Networks → Naming and addressing**; **In-network processing**; *Network architectures*; Session protocols;

KEYWORDS

Information Centric Networks, Named Data Networking, in-network processing, naming, thunks

ACM Reference Format:
Michał Król, Karim Habak, David Oran, Dirk Kutscher, and Ioannis Psaras. 2018. RICE: Remote Method Invocation in ICN. In *ICN '18: 5th ACM Conference on Information-Centric Networking (ICN '18), September 21–23, 2018, Boston, MA, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3267955.3267956

1 INTRODUCTION

Much of today's network traffic consists of data sent for processing to the cloud and web-servers exchanging high volumes of dynamically generated content. While today's ICN networks can deal efficiently with static data delivery, they have difficulty handling service/function invocation [24]. In view of these limitations, multiple works have recently tried to extend ICN's capabilities to deal with dynamic content.

Notable among these efforts, Named Function Networking (NFN) [29] and Named Function as a Service (NFaaS) [17] extend ICN's named data access model to a remote function invocation capability, enabling consumers to request the network to execute functions remotely. In NFN [29], for instance, function invocation corresponds to independent computational processes, evaluated as expressions in a functional programming model.

ICN provides several attractive benefits compared to remote invocation over current network protocol stacks (e.g. CORBA, RESTful HTTP[11]). Name-based routing allows the network to optimise the placement of computations with automatic load distribution and failure resiliency. The built-in object-based security of ICN frees application designers from the need to craft custom solutions in the common cases where channel security alone is insufficient. Short-term caching brings latency benefits under transient error conditions and mobility events, while long-term caching can substantially reduce server load for referentially transparent computations.

There have been several approaches for integrating computation with ICN, as we discuss in more detail in Section 2. However, when using them to realize real-world applications like web-style interactions, several additional aspects beyond the fundamental Named Function invocation concept need to be addressed:

- **Consumer authentication and authorization:** a producer should not blindly answer any consumer request. In basic ICN, this protection is provided by cryptographic data object integrity and encryption, *i.e.,* only authorized consumers are able to decrypt a received data object. In a Named Function Networking environment, the computation may be an expensive operation, so just relying on encryption and performing computations without validating consumer authorization may critically impede scalability of the whole approach.
- **Parameter passing:** Remote function execution typically requires a set of input parameters/arguments. In dynamic web content creation for example, the volume of such parameters (in bytes) can easily surpass the volume of the actual returned data objects [24]. Adding larger sets of parameters to Interest messages can introduce additional unsolicited

ICN '18, September 21–23, 2018, Boston, MA, USA

ACM ISBN 978-1-4503-5959-7/18/09…$15.00
https://doi.org/10.1145/3267955.3267956

traffic in ICN networks that could interfere with congestion control.

Accommodating non-trivial computations: Unlike responding to an Interest message with a (possibly pre-generated) static Data message, constructing responses by performing general computations (that could in turn invoke further remote computations) may take relatively long. In CCN/NDN, forwarders keep Interest state for matching received Data messages. The design and dimensioning of Pending Interest Tables (PITs) is typically based on the assumption that corresponding Data messages are received in a time frame that is based on typical network RTTs (e.g., order of 10s or 100s of milliseconds for wide-area networks). PIT state is, therefore, short-lived by design. As a result, Application Timescales can differ significantly from Network Timescales, which must be considered by a general purpose function invocation scheme.

In this paper, we aim to overcome these three limitations by enhancing the ICN model with function-oriented capabilities while preserving the core architectural and protocol design elements of ICN networks. RICE is a general-purpose network-layer framework that is currently missing from the ICN literature and can be applied to any named-function networking context. The main contributions of this paper are, therefore, as follows:

- We introduce a secure, 4-way handshake for ICN networks in order to achieve shared secret derivation, consumer authentication and parameter passing.
- We employ the concept of thunks [14] from the programming language literature to decouple method invocation from the return of results to enable long-running computations. The thunk is used to name the results for retrieval purposes.
- We discuss additional data acquisition techniques in the context of dynamic content.
- Finally, we provide a security evaluation, extensive simulation and a prototype of the presented mechanisms.

The ultimate goal of the proposed framework is to *enable in-network function execution with client authentication and non-trivial parameter passing, to support cases where computation takes longer than PIT expiry time.* We achieve this goal by decoupling application processing time from PIT timers and network RTT. We argue that this is a necessary feature of any name-based remote function invocation scheme, where computation is accommodated in distributed compute spots in local or wide area networks at the core or edge of the network. The mechanisms we propose follow ICN principles and require minimal and short-lived additional network state.

The remainder of the paper is organized as follows. In Section 2 we summarize previous work on function invocation for ICN. The model and the design goals are discussed in Section 3, while the design details of RICE are presented in Section 4. In Section 5 we provide a security analysis. Section 6 presents initial results evaluating the design with simulations, a comparison to alternative approaches, and a real-world prototype.

2 RELATED WORK

Various approaches have been proposed to integrate dynamic computation into ICN. In principle, generating a Data message as a response to an ICN Interest message can always involve computation – from a protocol perspective, ICN is not limited to static data delivery only. However, in order to specify computation requests, to optimize their execution in a distributed system, and to implement the corresponding platforms, different proposals have been made in the past.

Named Function Networking (NFN) [29] and Named Function as a Service (NFaaS) [17] are two exemplary developments towards dynamic in-network computations in NDN. NFN and NFaaS extend ICN to execute functions/services at any element in the network such as intermediate/edge nodes and end-user devices. While in ICNs content is fetched from the network, named functions provide results of dynamic computed content by evaluating expressions. NFN utilises λ-expressions in the names to identify the requested computations, while NFaaS focuses on hierarchical names and encapsulating functions in unikernels [19].

Within CCN [16], SCN [6] and NextServe [21] were proposed to enhance the network's capabilities by serving dynamic content. Soccer [26] builds a control layer on top of CCN for the manipulation of the underlying Forwarding Information Base (FIB) so that it always points to the best service instance at any point in time. In CCNxServe [28], authors focus on dynamic service deployment and scalability in a content-centric networking implementation.

While these implementations differ significantly among themselves and to our framework, they also share some common features with RICE:

- They use Interests to request computation and allow computation requests to be satisfied with cached results of referentially transparent functions.
- It remains transparent to the client whether the resulting data is pre-generated, served from a cache or generated on-demand.

Closer to our work, Gasparyan et al. presented an extension, enhancing SCN with session support [12]. The system introduces 3 new types of messages and a 2-way handshake that creates permanent FIB entries allowing the consumer to reach the same producer multiple times. However, the framework does not allow for parameter passing, creates permanent entries that can be exploited by malicious nodes and is susceptible to Denial of Service attacks.

None of the related work, however, includes the necessary mechanisms to deal with *i) consumer authentication and authorization, ii) parameter passing, and iii) dynamic content retrieval* [29] [17] [6] (apart from [12], which comes with its own weaknesses).

All of these needed capabilities interact with the question of how to manage Network versus Application Timescale. If the data generation time is longer than the PIT expiry timer, the future data name has to be communicated to the consumer who then will attempt to fetch the newly produced data by sending regular Interests. Such an approach increases the overall communication delay and requires the producer to globally advertise the newly created content, ultimately limiting the usability of those systems.

In [25] and [8], authors present application-level NACK packets to notify consumers when dynamically generated content will be ready in order to avoid too frequent polling. In this paper, we add to the discussion by presenting new results and allowing consumers to reliably reach the corresponding producers.

3 MOTIVATION AND DESIGN GOALS

Our analysis of existing approaches has led us to rethink the relationship of ICN and distributed computing. Many remote method/-function invocation schemes and frameworks are fundamentally trying to leverage the ICN Interest-Data interaction scheme to request function execution and then return results to the initiating party. Such schemes could potentially be used for realizing web-style communication, service chaining and other application services.

However, this approach is not viable for all but the most trivial applications:

1) In ICN, Interest forwarding establishes *pending Interest* state in the network. *Pending Interest Tables* (PITs) keep state per Interest message as "breadcrumbs" that are used to forward Data messages back to the consumer. In addition to that, PITs also enable *Interest aggregation* (suppression of Interest forwarding when there is already a pending Interest for the same named data object). This, together with opportunistic caching, enables ICN's efficiency and producer offloading features. Correct ICN forwarding requires Pending Interest state to expire fast (on the order of network RTT) so that retransmitted Interests will actually be forwarded upstream. ICN Interest retransmissions are not an exception in ICN: they can occur in the presence of packet loss, congestion, or during consumer mobility. Because Pending Interests *have* to expire, producers cannot take an arbitrarily long time to respond to Interests – otherwise their Data messages would just be dropped on the their way downstream (because there would not be any next-hop forwarding information). This makes the straightforward ICN Interest-Data interaction unsuitable for invoking computations that take longer than a few RTTs: application/computation completion Timescales can be orders of magnitude larger than ICN Network Timescales.

2) ICN congestion control relies on the notion of flow balance, *i.e.,* every Data message corresponds to exactly one Interest message. ICN's receiver-driven congestion/flow control can thus adjust the Interest sending rate to regulate the Data sending rate at producers (or on-path caches). The assumption is that Data messages consume the bulk of the resources on network links and in forwarding queues. Therefore, Interest messages act as congestion control signals (in the reverse sense as TCP ACKs work) and are not intended to carry much additional data other than the information needed to forward the Interest correctly. Adding additional data (for example, function parameters) would thus lead to more traffic (more bits) in the network carried in messages that are preferentially dropped (or NACKed) under congestion. Dropped Interest messages are typically only observable through Data message time out. We therefore concluded that requiring large Interest messages for function invocation is not viable for non-trivial computation requests.

With these observations in mind, we designed RICE as a general *Remote Method Invocation* (RMI) service – providing a robust and secure basis for a wide range of applications and in-network computation scenarios, including scenarios where method invocation requires a significant amount of input data and involves computations that take significantly longer than a network RTT to complete.

RICE is independent of any particular function execution environment. RICE provides all the required ICN protocol mechanisms and conventions for clients and servers and can serve as an underlying platform to support frameworks such as NFN and NFaaS (as well as any other in-network compute framework that utilises core ICN principles). In the description of RICE we use the following terminology:

Consumer: ICN protocol entity that is sending an Interest message

Producer: ICN protocol entity that is sending a Data message, replying to a received Interest message

Client: RICE protocol user that wants to request a remote method invocation

Server: RICE protocol user that is processing and answering the remote method invocation request (this may be a logical entity that is represented by more than one ICN protocol entity)

3.1 Design Goals

We have chosen the following design goals for guiding the development of RICE :

Decouple application time scale from network time scale: RICE does not map remote method invocation directly to one Interest/Data exchange for the reasons discussed above. We want clients to request remote method invocation from a server without changing the network behavior with respect to Pending Interest management.

Support client authorization: A RICE server should be able to authorize clients before committing resources such as state, processing power etc. A server that blindly accepts any RMI request opens itself to computational and/or memory overload attacks. This also implies that the authorization mechanisms must be designed so that performing authorization itself does not overload servers and open a vulnerability to computational attacks.

Support non-trivial method invocation with arbitrarily complex parameter sets

Be robust and ICN-friendly to a mix of RICE and non-RICE traffic: we want RICE to coexist seamlessly in existing ICN networks, *e.g.,* it should adhere to the same flow balance principles. The main consequence is that we do not transmit RMI parameters in Interest messages.

Support non-trivial, long-running computations with large amounts of result data

Support session-like interactions, where a client and a server use a sequence of RICE exchanges: RICE result data should be chunkable, and it should also be possible to retrieve result data that is generated over time, for example in multiple invocations in the same "session".

Allow ICN caching for referentially transparent method invocations: ICN generally supports location-independent data access and opportunistic caching. In RICE, we want to support referentially transparent functions efficiently, i.e., function expressions that can be replaced with the result of the actual function execution. In other words, RICE is able to cache the result from such function invocations and enables the network to answer subsequent Interest messages from caches. Since not all functions are referentially transparent (some functions may depend on other data from the environment that is not specified in arguments), our framework distinguishes between functions that are referentially transparent and those that are not.

Adhere to ICN principles: RICE should not give up important ICN principles, such as flow balance, implicit support for consumer mobility, consumer anonymity (no source addresses). This last property is worth mentioning, because some ICN extensions/applications rely on the fact that one end of an interaction would provide a globally routable "source" address to the other end to achieve "callback" behavior or generally enable bidirectional communication. Since such schemes would expose client identity information to the network (and to peers), we deem this approach an unacceptable deviation from ICN principles. Client identities should be exposed only to the application layer. Other forms of identifiers that help the network for maintaining reverse forwarding state to clients should be designed so that they do not expose clients' identities.

Be compatible with ICN extension mechanisms: Some recent proposed extensions make benign changes to ICN forwarding behavior (without compromising general interoperability). One example would be fast forwarding information updates using technique such as Map-ME [5]. RICE should be designed to work with such extensions. In this particular case, this means that RICE should support client and server mobility in a Map-ME-enhanced network.

Make minimal changes to ICN protocols and forwarder behavior: We want to allow for some changes to ICN forwarder behavior, but these should be limited and designed so that they do not complicate forwarder implementations or impair their performance.

Our general guideline for achieving these goals is to prioritize robustness, security and scalability over absolute efficiency (with respect to number of handshakes and message exchanges), while still arriving at a design with reasonable efficiency.

4 RICE REMOTE METHOD INVOCATION

Remote Method Invocation (RMI) operations in RICE are split in two ICN interaction phases: 1) RMI Initiation (eventually triggering the remote method execution), and 2) the Result Retrieval. The RMI Initiation phase is designed to complete in Network Timescale (on the order of a few RTTs), whereas the remote method execution is decoupled from that and can take as long as required. The Result Retrieval phase can consist of several ICN Interest-Data exchanges (for chunked results or for computations that generate results iteratively).

4.1 Naming

We describe a generic naming scheme for remote method invocation that can be used with multiple existing frameworks. We distinguish between *Function Names* and *Thunk Names* (Fig. 1). A function name identifies a method requested by a client that can be executed by any server able to perform the computations. In contrast, a thunk name identifies a specific method instance already being run on a specific node.

When a client initially requests the invocation of a referentially transparent function, the name in the Interest must unambiguously specify both the invoked function and the set of input parameters. The function part can be system-specific (*e.g.,* λ-expressions in [29] or expressed as a hierarchical name structure as in [17]), but must unambiguously identify the function to invoke. The input parameters can be represented directly when very small in size (*e.g.,* username, number of iterations to perform), or as a hash when larger.[1] One trade-off is the increased size of the Interest message versus the extra round trip for fetching the actual input parameters. Another, perhaps more important consideration is that the parameter values are protected by existing ICN encryption mechanisms when fetched via reference by the server, but would require custom confidentiality mechanisms by the application if sent in the Interest message.

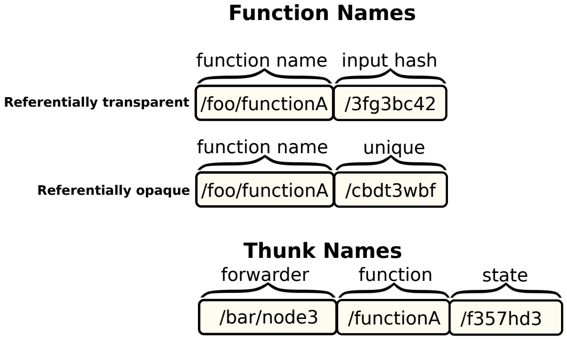

Figure 1: Naming convention.

For referentially opaque functions, the result can be different even when using the same input parameters. Every invocation (either from the same client or from other clients) must lead to a new computation instance. Therefore, Interests for triggering referentially opaque function invocations must use a unique name for each invocation, while Interest retransmissions from the same client must use the same name. This prevents the network from aggregating the corresponding Interests and limits cached responses to only answer retransmitted interests from a individual consumer. To achieve this, the client includes a name component that distinguishes its request from those generated by other clients. This component should be chosen by the client such that other nodes cannot predict the value.[2]

The thunk name must unambiguously identify the server's forwarder, the instantiated function and the function's internal state[3]. In that way, when the client uses the thunk name in consecutive requests, the Interest can be forwarded to the correct server and dispatched to the application possessing the associated result data. With thunk names, we do not need to distinguish between names for referentially transparent and opaque functions. They unambiguously identify a handler to a function execution instance and it is up to the server to return the same or different handlers to multiple clients.

4.2 Handshake

We propose a 4-way handshake which serves the purposes of *i) deriving a shared secret, ii) authenticating and authorizing clients, and iii) providing input parameters to functions.* In the following, we begin by describing the 4-way handshake and we continue by showing how this handshake serves those three purposes.

[1] The choice of the actual hashing method can be left to the application.

[2] Using predictable information (*e.g.,* a MAC address) opens an additional attack surface and can leak client's sensitive information.

[3] *i.e.,* input parameters, chunk number

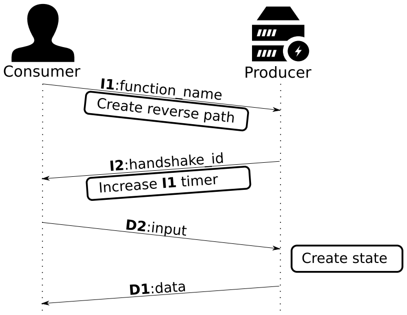

Figure 2: 4-way handshake. The procedure enables bidirectional information exchange between the consumer and the producer.

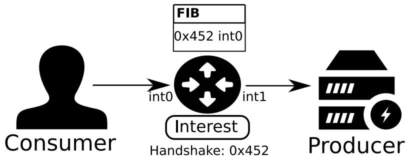

Figure 3: Handshake temporary FIB entry.

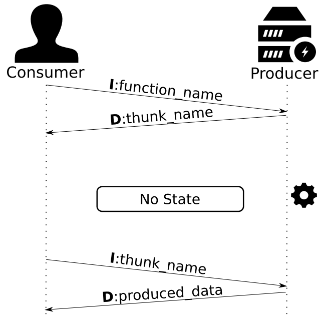

Figure 4: Thunks. The consumer retrieves function thunk name and uses it to download the result when the computation is finished.

A client starts a 4-way handshake by sending an Interest message *I1* towards a server (Fig. 2). Similarly to the TCP SYN flag, the message contains an additional TLV *Handshake* field[4]. The field contains an identifier that is chosen by the client and distinguishes among different handshakes (Fig.3). Upon receipt of the Interest, each forwarder creates a corresponding PIT entry as with regular Interests. Forwarders also inject a new temporary entry in the FIB formed from the identifier and pointing to the face on which the Interest was received. This FIB entry has a short expiry time after which it is deleted (we experiment with different expiry time values and discuss the implications for FIB performance in Sec. 6).

When the server receives the Interest, it responds with an *I2* Interest message and forms the name from the received identifier. The Interest thus follows the previously established FIB entries towards the client. The forwarder also increases the expiry timer of *I1*’s PIT entry. At this point, the client must be ready to respond and send a *D2* Data message.[5] When the server receives *D2* it allocates resources for the computation for the client and sends back a confirmation in *D1* Data message. Delivery of *D1* is possible despite the extra *I2/D2* exchange because *I1’s* PIT entry timer was increased by *I2.*

Interest *I1* of the first handshake contains a function name that is delivered to a server advertising the corresponding prefix. At the end of the first handshake, in *D1*, the server returns its thunk name. This name can be then used by the client in *I1* message of consecutive handshakes to assure future invocations reach the same function instance.

4.2.1 *Shared Secret Derivation.* Common secret derivation plays an important role in many security protocols. Once a common secret is established, both parties can encrypt the communication using symmetric cryptography.[6] Common secret derivation (*e.g.,* via Diffie-Hellman Key Exchange) requires sending data in both directions between the involved parties and is clumsy to implement in vanilla ICN networks where not all nodes have globally routable prefixes. *Using one or multiple handshakes as described above allows use of any symmetric key scheme to secure the communication over arbitrary paths.*

4.2.2 *Client Authentication.* Upon reception of a method invoking Interest, the server normally wants to verify the identity of the client and apply an authorization policy. A naive approach would include client credentials and function parameters directly in the Interest name or payload. However, as noted earlier this can significantly increase the size of the message and lead to additional, non-congestion-controlled, messages in the network[24]. A straightforward way of authenticating clients in ICN would be to use either signed Interests or command Interests. While theoretically this could be done, in practice it would introduce problems for the design of scalable, high-performance servers. If a server had to validate client signatures and certificates before being able to decide how to process (or not) a request, this could lead to significant server load.

Reliable and secure authentication requires multiple messages to be exchanged between the server and the client. *Utilizing the handshake for client authentication, specially the exchanged* I2/D2 *messages, we achieve this goal and we decouple it from function invocation and input parameter passing.* In addition, by using the *I2/D2* messages for client authentication and avoiding adding authentication information in the original interest (*I2/D2*), our handshake mechanism becomes protected against any record-and-replay attack by a malicious party which can intercept the traffic. Once authenticated, the client creates a security token that can be included as a last component of the thunk name. In further communication such as commands (*i.e.,* pause, stop) or referentially opaque functions, the token can changed for each consecutive Interest message (i.e. based on the last received Data message).

It should be noted that RICE can rely on idiomatic ICN mechanisms for server authentication, i.e., validate the signatures on Data or signed Interest messages. The client could also encrypt input parameters using the public key of the server. We leave a detailed description to future work.

Following ICN principles, clients should not authenticate the server performing computations, but rather authenticate the returned result. If submitted input contains confidential data, it can be encrypted and shared using existing ICN access control techniques [15].

[4] This description assumes NDN[30] implementation of ICN. However, our solution can be easily adapted to other implementations such as CCNx[2]

[5] Authorization information and parameters may be larger than what can be carried in a single *D2* message. Therefore, multiple *I2/D2* exchanges using the built-in chunking capabilities of the underlying ICN protocols can be employed

[6] Symmetric encryption is less CPU-intensive than public key infrastructure (PKI) and can be easier to implement when the same function is hosted by different servers, and the client does not know which one will handle the request.

4.2.3 Input Parameters. Many functions require considerable input to complete the requested task (*e.g.,* frames for image processing). When the data is hosted on the network with a routable name, the server can simply request it with Interest messages. However, in common cases the input is present only at the requesting client, who typically does not possess a routable prefix.

We model input parameters to remote functions on the passing of arguments "by reference" in regular function calls. Since the server cannot access memory in the client, we use the *I2/D2* Interest/Data Exchange as a "callback" from the server to the client to fetch the input parameters. This preserves the ICN principle that data is never pushed to the server if not requested. The server piggybacks on the PIT entry established by *I1* to reach the client and pull the required data. If the input contains sensitive information, the client can encrypt it using a common secret derived in previous handshakes as described above.

4.3 Dynamic Content Retrieval using Thunks

In dynamic content generation or function execution, the server may require a significant amount of time to create the requested data. In this section, we describe our use of thunks to allow clients to retrieve computation results.

Thunks are illustrated in Fig. 4. The client starts by sending an Interest with a function name. Thunks allow multiple clients calling the same referentially transparent function with identical parameter sets to share one PIT entry and efficiently retrieve the data, while keeping entries separate for referentially opaque computations. Upon receipt of the Interest, the server starts the requested computation and immediately responds with a thunk Data message. For the network, the thunk does not differ from a regular Data message that consumes the pending PIT entry. The payload of the message contains a *thunk name* and an estimated completion time. The client waits for the time indicated in the server's response and issues a new Interest with the received thunk name.

If the computation has finished, the function responds with a Data message containing the result. If the server mis-estimated the completion time and the data is not ready, it returns the same thunk target with an updated completion time estimate.

Thunks allow for efficient dynamic content retrieval while minimising the state present on intermediary nodes. In case of client mobility, Thunks provide an extra advantage: when the client changes its location during the data generation, the established thunk remains valid and can be used to retrieve the result from the client's new location.

4.4 Alternative Retrieval Techniques

While thunks represent the most robust approach in a wide range of scenarios (see Sec. 6 for detailed results), in this section we consider hypothetical alternative techniques to retrieve dynamic content.

4.4.1 Network Timescale Interest Timers. In this approach the client assumes computations complete in a small number of network RTTs and hence sets his *Interest Lifetime* based on Network Timescale.[7] The client sends its first Interest using a function name and upon its reception, the server starts generating the content. If the generation process takes longer than the Interest Lifetime, the PIT entry expires. In this case, the client resends the Interest using the same name in an attempt to recover since it cannot tell the difference between a lost interest and a delayed response (Fig. 5). When the data is created, it is immediately sent to the client. Apart from the increased overhead of sending multiple Interests, the *Network Timescale* approach comes with another major weakness: when multiple servers are present, the network in general does not guarantee delivery of Interests from the same client to the same server.

4.4.2 Application Timescale Interest Timers. In this alternative approach to *Network Timescale* , the client sets *Interest Lifetime* long enough to cover the entire computation time (in addition to the network latency). Therefore PIT entries are kept during the whole data generation process (Fig. 6). The client needs to send only one Interest that will be satisfied when the data is created. In error-free conditions, nodes exchange only the minimal 2 messages to retrieve a data chunk, and unlike Network Timescale, Application Timescale can support multiple servers. However, the intermediary nodes need to keep state for the whole time of data generation. Moreover, if the Interest is lost, the client does not detect the loss and send a new Interest until the Application Timescale timer expires.

4.4.3 Interest Acknowledgements. It is possible to ameliorate the long loss recovery time of Application Timescale approach by adding Interest Acknowledgement messages to the ICN protocol (Fig. 7). The client can therefore use an *Interest Lifetime* of Network Timescale. The client starts by sending an Interest with a function name. Upon reception of the message, the server starts the computations and responds with an ACK message. This (provisional) acknowledgement is a separate type of message and must be recognised by all the forwarders. The ACK does not consume the pending PIT entry in the forwarders of the original Interest message, but increases its expiry time by enough to cover the expected response time. The PIT entry will thus be maintained until the requested computations are finished. When the server finishes the task it simply responds with a Data message. The addition of ACK packets allows for faster detection and recovery of Interest loss, but the solution remains vulnerable to network dynamics and client mobility[8].

5 SECURITY AND PRIVACY ANALYSIS

By building on a well-studied ICN protocol framework, our approach to remote method invocation shares the fundamental security advantages and difficulties of those protocols [3, 7, 23]. RICE employs native CCN/NDN machinery for cryptographic data integrity, origin authentication, confidentiality, and key management [20]. Similarly, we share the privacy problems and limitations of extant ICN protocols [13]. In general, the additional threat that needs to be addressed is a computational or state-creation attack against a server by un-authorized clients. These threats request long-running computations, or flood the server with remote invocation requests that start useless computations [4], [18].

[7] RTT estimation is important for anything to work effectively, irrespective of whether one chooses to operate the remote method invocation using Network or Application Timescale. For example, nearly all workable congestion control schemes rely on some form of RTT estimation.

[8] Acks can also introduce a conundrum for security - if they are not signed by the server it is possible for off-path attackers to manufacture them. If they are signed, there is increased exposure to computational attacks on the server.

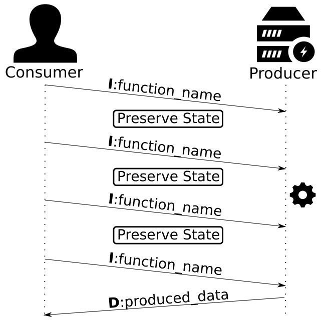

Figure 5: Network Timescale. The Interest Lifetime is set based on network RTT estimation. The consumer resends the Interest if the computation takes more time.

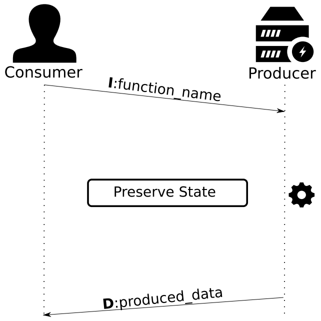

Figure 6: Application Timescale. The Interest Lifetime is set based on estimated computation time.

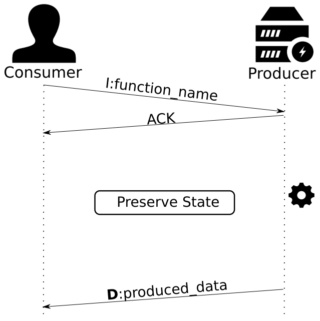

Figure 7: Acknowledgements. The Interest Lifetime is set based on network RTT estimation. The returning ACK inflates PIT expiry timer to match estimated computation time.

In this section, we concentrate on the vulnerabilities associated with RICE, and provide a brief security analysis of our machinery. The proposed 4-way handshake provides secure input parameter passing. It does so by fetching the parameters via the Interest-Data exchange "callback" from the server to the client. Such callbacks could open a reflection attack via interest flooding [7] if a globally routable name were used for the callback operation. However, the reverse-path mechanism and non-routable name RICE uses confines knowledge of the client's input parameter set to the server and on-path forwarders. The remaining vulnerability is the need to maintain forwarding state for the entire duration of the four-way handshake, rather than just a two-way exchange.

When a client sends *I1* towards a producer, it creates additional state in FIB tables on all the intermediary forwarders. However, similarly to PIT entries, the created state is purged when its timer expires. If the timer is set to the same value as the PIT entries divided by 2, it does not expand the attack surface and existing Interest flood prevention techniques can be applied [3]. Upon reception of *I1*, the producer does not create any local state or allocate resources for the client. This protects our system from DoS attacks similar to the TCP SYN flood attack [9]. When the producer responds with the *I2* message, that follows the trail created by *I1*. Such an approach assures that *I2* is delivered only to the client initiating the session and eliminates the threat of using the producer as a spam bot. Following ICN principles, the data is effectively pulled by the producer from the consumer assuring that the producer does not receive large volume data that it did not request.

6 EVALUATION

Our evaluation focuses on the core RICE feature – the dynamic content retrieval using Thunks. We implemented RICE using NDN [30] and did our evaluation on ndnSim v2.5 [22]. The default PIT entries expiry time is set to 1*s*. We make our implementation open and publicly accessible to the research community.[9]

6.1 Handshake Evaluation

In this section, we evaluate the overhead introduced by the 4-way handshake between a server and a client. We start by evaluating the number of messages sent by both nodes for different loss rates (Fig. 8). We provide results for different expiry time values for the ephemeral FIB entries. When the expiry time value is set to *1xRTT*, increasing the loss rate causes many more messages being exchanged by both parties. Even if a single message is lost, the whole process must be started from the beginning, increasing the overhead. When setting the expiry time value to *2xRTT* this influence is much less visible. When *I2* is lost (Sec. 4.2), the server can resend it without waiting for the client to restart the whole process. However, increasing the expiry time to higher values above *2xRTT* does not further reduce the overhead.

We repeat the test, this time measuring the completion time of the whole process (Fig. 9). Similarly to the number of exchanged messages, the completion time is heavily influenced by message loss when the expiry time is set to *RTT*. When setting the expiry time to *2xRTT*, both parties can resend single messages instead of starting the whole process from scratch. The completion time thus increases less with message loss rate. Again, further increasing the expiry timer does not significantly reduce the completion time.

We conclude that *2xRTT* represents an optimal value for the ephemeral FIB entry expiry time and the best trade-off between the state size and the number of exchanged messages.

6.2 Data Retrieval Techniques

Next, we compare Thunks against Acknowledgements, Network Timescale, and Application Timescale. We perform the evaluation in a simple line topology containing 3 nodes: a client, an intermediary node and a server. We set data generation time to 5*s*, Network Timescale to 1*s* [10], loss rate to 0.01 and Application Timescale to 5*s* unless stated differently. For the evaluation, we first assume that the server is able to correctly estimate the time required to generate the requested data and then we evaluate the impact in bandwidth and delay of mis-estimation.[11] When evaluating Application Timescale, we assume that the PIT expiry time is equal to the data generation time.

6.2.1 Interest Satisfaction Time. We begin by measuring how much time the network requires to deliver the generated data to the client.

[9] https://github.com/mharnen/timers

[10] default PIT expiry time value in NDN

[11] Inaccuracy in the time estimation of computation for real systems could be biased either way depending on the relative importance of bandwidth (for retransmitted thunk interests) versus delay (due to waiting longer to transmit the thunk)

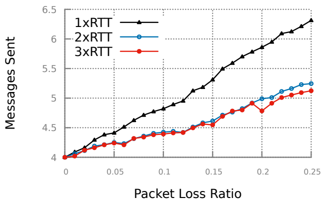

Figure 8: Handshake Overhead.

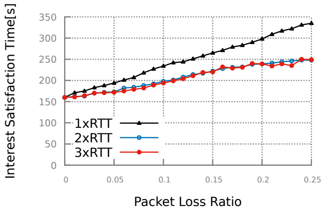

Figure 9: Handshake Completion Time.

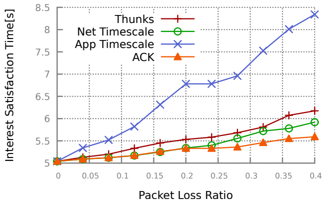

Figure 10: Interest Satisfaction Time.

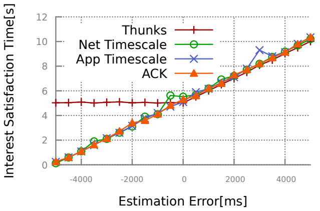

Figure 11: Number of Interests sent to retrieve a data chunk.

Fig. 10 presents the impact of increasing packet loss rate. Without any lost messages, all the approaches achieve the same minimal Interest satisfaction time. With increasing loss rates, Thunks, Network Timescale and ACKs require slightly more time to deliver the data. This is because when an Interest is lost, those approaches need to wait for 1*s* for the PIT entry to expire and then retry the transmission. In contrast, when using Application Timescale, the PIT entry expires only after 5*s*, significantly delaying any re-transmissions. When using Network Timescale only loosing the last Interest (sent when the computation is finished) increases the satisfaction time. When previous one are lost the satisfaction time remains unaffected. This applies to ACK messages when using ACKs as well. This is why Thunks achieve slightly higher satisfaction time.

We proceed by evaluating the impact of generation time miscalculation. Servers can only estimate the amount of time required to compute the results and this estimation can be imprecise. Fig. 11 shows the impact of over (negative values) or underestimating the data generation time. When the data is generated before the declared time, ACKs, Network and Application Timescales can retrieve the data right away. However, Thunks do not keep state on intermediary nodes and thus cannot react before the declared time. If the server needs more time to compute the results, it sends back an updated time generation value instead of the actual data. In this case, all the solutions behave alike – adapt and retrieve the data as soon as it is ready.

6.2.2 Overhead. Next, we focus on the number of messages required to retrieve one data chunk. When using Application Timescale, the client sends only one Interest that will be satisfied when the data is generated. A client using Acknowledgements also has to send only one Interest per Data message. However, in this case, the server needs to confirm its reception by generating an acknowledgement. When using Thunks, the client generates two messages per data chunk - the first one to get the thunk name and the second one to retrieve the generated data. Finally, with Network Timescale, the client sends an Interest every second regardless the data generation time.

Fig. 12 presents the introduced overhead for different message loss rates. All the solutions are only slightly affected by the increasing message loss ratio. When an Interest is lost, all the techniques simply perform a re-transmission. Network Timescale, in spite of sending more packets maintains almost unchanged overhead. This is because the client needs to send an additional Interest only when the last one transmitted before retrieving the result is lost.

We then investigate the impact of increasing generation time on the number of sent Interests (Fig. 13). Thunks, ACKs and Application Timescale maintain the same number of exchanged messages. In contrast, when using Network Timescale, the client generates one Interest per second. The overall number of sent Interests thus increases linearly with the data generation time.

We proceed, by checking the impact of generation time miscalculation (Fig. 14). When the data is produced before the estimated time, Network Timescale decreases the number of generated Interests, while other techniques remain unaffected. If the deadline is even slightly exceeded, ACKs and Application Timescale have to repeat the data retrieval process. On the other hand, thunks tolerate up to 1*s* underestimation without sending additional packets. Network Timescale increases the number of generated Interests proportionally to the data generation time.

6.2.3 State size. We measure the maximum amount of state kept at intermediary forwarders expressed in number of pending PIT entries. Each client sends Interests for 10 different functions per second.

Fig. 15 presents the impact of increasing data generation time. Acknowledgements, Network and Application Timescales keep state on the intermediary node through the whole time of data generation process (Network Timescale, refreshes the state every second). When the server takes longer to generate the data, the state increases linearly as well. On the other hand, Thunks do not keep any state on the intermediary nodes when waiting for data and are not influenced by changing the generation time.

We continue by investigating the impact of loss rate on the maximum state size with data generation time fixed to 5*s* (Fig. 16). Again, thunks remain almost unaffected by loss rate. The small amount of state is experienced only when an Interest is lost and an entry remains in the PIT table until it expires. With increasing loss rate, all the other solutions take longer to complete the data retrieval and state size is significantly increased.

The two previous tests show only the highest recorded state for each investigated scenario. We provide a better insight into the state size by showing state size evolution in Fig. 17 for 5*s* data generation time and 0.1 loss rate. Thunks record the lowest amount of state that rises only when a message loss occurs. On the other hand all the other solutions quickly increase the state stored on the intermediary forwarders and keep it high during the whole simulation period.

6.2.4 Large Topology Evaluation. We continue the evaluation on a large network using the RocketFuel 1239 - Sprintlink topology [27], containing 319 nodes. We choose routers having only one link as clients (30 nodes) and randomly deploy 6 servers. We consider a domain of 100 services, randomly spread among the servers so that each service is present on at least 3 servers. Each service requires

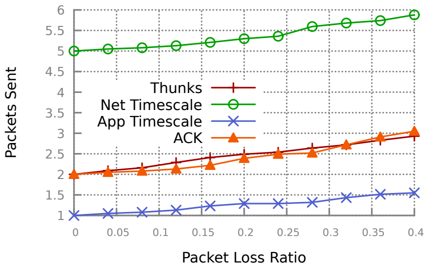

Figure 12: Packets sent for different loss rates.

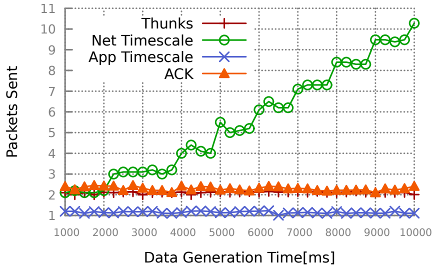

Figure 13: Packets sent for different Data Generation Times.

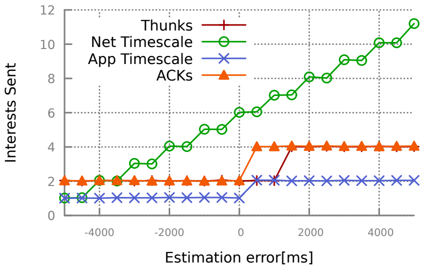

Figure 14: Number of Interests sent to retrieve a data chunk.

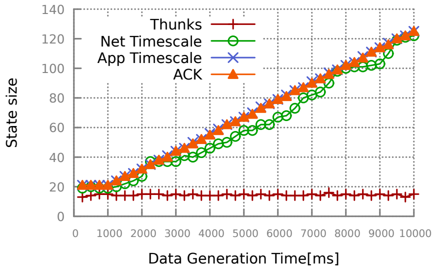

Figure 15: Forwarder state size.

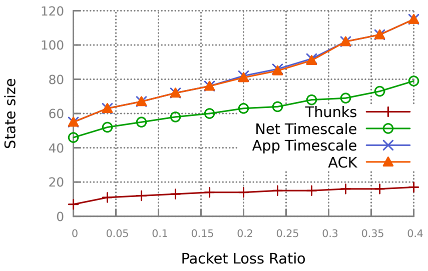

Figure 16: Forwarder state size.

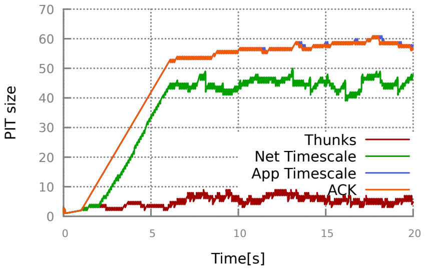

Figure 17: Evolution of state size in time.

5s to complete. Each consumer generates 1 request per second for 1 out of 5 randomly chosen services. We assume that each server can execute at most 5 function at the same time and use a load balancing round robin strategy on forwarding nodes.

Fig. 18 presents cumulative distribution of Interest satisfaction time for referentially opaque methods. Thunks retrieve the majority of the results within 5s which represents the optimal value. Network Timescale records much higher retrieval time. This is because consecutive Interests for the same method can be forwarded to different servers triggering parallel computations. This also applies to Application Timescale and Acknowledgements when an Interest is lost. Application Timescale performs worse than Acknowledgements as it requires more time to perform a re-transmission.

We repeat the same tests for referentially transparent methods and enable result caching (Fig. 19). Interests can now be directly satisfied by already computed results, decreasing the satisfaction time for all the techniques. Surprisingly, thunks experience lower rate of returned cached results (values below 5s) than other solutions. This is because results can be cached under method or thunk names (when requested for the first time) directly decreasing cache hit ratio. One possible remedy entails using thunk names only as forwarding hints [1] and keeping method names in all the Interests. However, such a solution requires support for forwarding hints by every forwarder in the network.

All the techniques except network time require the producer to know or estimate the time required to generate the result. However, while with thunks the consumer can receive this information in the first Data message, with Application Timescale and ACKs the consumer needs to know it in advance to correctly inflate the duration of the PIT entries. Thunks also exhibit the lowest overhead and scale well with increasing number of requests, but they come with a cost of the consumer not being notified if the data is generated before the extimated completion time.

6.3 FIB performance considerations

Forwarding tables in routers are usually designed with algorithms strongly biased for reads over writes. Our handshake design however requires changes to the FIB at a rate that is some fraction of the rate of change to the PIT. One approach of course is to choose a FIB access algorithm with excellent write performance and no read/write locks that could interfere with normal forwarding. An alternative is to use a separate data structure for the temporary FIB entries that has balanced read and write performance. It is relatively straightforward to do the necessary "lookaside" into this temporary FIB table since the names used for the I2 Interest messages can be identified with an architectural constant name prefix whose longest prefix match points to the temporary FIB. We however did not evaluate or implement the separate temporary FIB described here.

6.4 Prototype

To confirm the performance and efficiency of RICE in a real-world scenario, we implement and evaluate a simple Optical Character Recognition (OCR) algorithm [10]. This setting allowed us to illustrate the execution of functions requiring large input. We assume 2 clients residing on Raspberry PIs 3 Model B and a server on a Dell XPS 13 laptop. Our function is initialised on a server with a model of each letter of the alphabet; the model can then be fed with input images to detect the list of embedded letters. Clients randomly submit 1 of 5 prepared 1.6KB images to the server. The server processes each image 500 times, to extend the execution time.

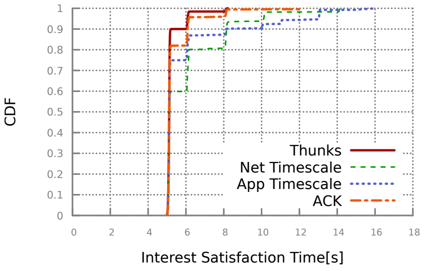

Figure 18: Interest Satisfaction Time for referentially opaque methods.

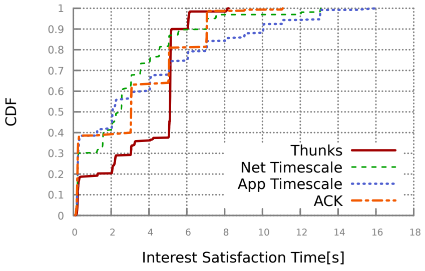

Figure 19: Interest Satisfaction Time for referentially transparent methods.

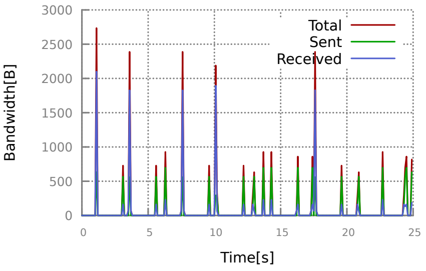

Figure 20: Prototype. Bandwidth used by different workers.

Fig. 20 presents the number of sent and received bytes measured on the server using wireshark and 100ms resolution. We can observe clients submitting images using the 4-way handshake (high 2000B spikes). After 2s, the submitting client uses thunks to retrieve the computed result (low 500B spikes). While the experiment continues, the server accumulates cached results and is able to respond to the handshake directly with computed data. We thus observe only 5 high spikes for 5 different submitted images and multiple low spikes to retrieve the results.

7 CONCLUSION

We presented RICE, a framework for Remote Method Invocation in ICN. We evaluate various trade-offs in the protocol dynamics associated with long-running computations, and provide a natural set of primitives to enable both client authorization and efficient and secure input parameter passing. We show that modest protocol extensions can achieve efficient and architecturally clean remote invocation while preserving the attractive properties of a pull-based ICN design like CCN or NDN.

RICE demonstrates how ICN can enable the elegant integration of dynamic computing and named data access, with low overhead, hostname and address independence, security and high efficiency through caching and result data sharing. We believe that RICE is a robust foundation for many ICN compute applications, including web access, distributed NFN, and distributed edge computing environments, where data needs to be processed close the data source before being sent to the back-end permanent storage point.

Our plans for future work include studying potential optimizations for efficient client/server and network interaction, for example the use of NACKs and forwarding hints, implementing highly scalable server systems, and testing RICE in different application scenarios. This would also include more in-depth analysis of specific protocol features such as client authentication and parameter passing.

8 ACKNOWLEDGEMENTS

Börje Ohlman participated in the design discussions that led to this paper and provided valuable insights and feedback. The authors are grateful to the ACM ICN’18 anonymous reviewers and our shepherd Peter Steenkiste for their constructive comments and suggestions. Michał Król is supported by the EC H2020 ICN2020 project under grant agreement number 723014, and Ioannis Psaras is supported by the EPSRC INSP Early Career Fellowship under grant agreement number EP/M003787/1.

REFERENCES

[1] [n. d.]. Forwarding Hints. https://named-data.net/doc/NDN-packet-spec/current/interest.html. ([n. d.]).

[2] 2018. Project CCNx. http://www.ccnx.org/. (2018).

[3] Alexander Afanasyev, Priya Mahadevan, Ilya Moiseenko, Ersin Uzun, and Lixia Zhang. 2013. Interest flooding attack and countermeasures in Named Data Networking. In *IFIP Networking Conference, 2013*. IEEE, 1–9.

[4] Mustafa Al-Bassam, Alberto Sonnino, Michał Król, and Ioannis Psaras. 2018. Airtnt: Fair Exchange Payment for Outsourced Secure Enclave Computations. (2018). https://www.ee.ucl.ac.uk/~uceeips/files/airtnt-payments-v1.pdf

[5] Jordan Augé, Giovanna Carofiglio, Giulio Grassi, Luca Muscariello, Giovanni Pau, and Xuan Zeng. 2016. MAP-Me: Managing Anchor-less Producer Mobility in Information-Centric Networks. *arXiv preprint arXiv:1611.06785* (2016).

[6] Torsten Braun, Volker Hilt, Markus Hofmann, Ivica Rimac, Moritz Steiner, and Matteo Varvello. 2011. Service-centric networking. In *Communications Workshops (ICC), 2011 IEEE International Conference on*. IEEE, 1–6.

[7] Seungoh Choi, Kwangsoo Kim, Seongmin Kim, and Byeong-hee Roh. 2013. Threat of DoS by interest flooding attack in content-centric networking. In *Information Networking (ICOIN), 2013 International Conference on*. IEEE, 315–319.

[8] Alberto Compagno, Mauro Conti, Cesar Ghali, and Gene Tsudik. 2015. To NACK or not to NACK? negative acknowledgments in information-centric networking. In *Computer Communication and Networks (ICCCN), 2015 24th International Conference on*. IEEE, 1–10.

[9] Wesley M Eddy. 2007. TCP SYN flooding attacks and common mitigations. (2007).

[10] Line Eikvil. 1993. Optical character recognition. *citeseer. ist. psu. edu/142042. html* (1993).

[11] Roy Thomas Fielding. 2000. *Architectural Styles and the Design of Network-based Software Architectures*. Ph.D. Dissertation. University of California, Irvine.

[12] Mikael Gasparyan, Guillaume Corsini, Torsten Braun, Eryk Jerzy Schiller, Saltarin de Arco, and Jonnahtan Eduardo. 2017. Session Support for SCN. (2017).

[13] Cesar Ghali, Gene Tsudik, and Christopher A. Wood. 2016. (The Futility of) Data Privacy in Content-Centric Networking. *Proceedings of the 2016 ACM on Workshop on Privacy in the Electronic Society - WPES’16* (2016). https://doi.org/10.1145/2994620.2994639

[14] P. Z. Ingerman. 1961. Thunks: A Way of Compiling Procedure Statements with Some Comments on Procedure Declarations. *Commun. ACM* 4, 1 (Jan. 1961), 55–58. https://doi.org/10.1145/366062.366084

[15] Mihaela Ion, Jianqing Zhang, and Eve M Schooler. 2013. Toward content-centric privacy in ICN: Attribute-based encryption and routing. In *Proceedings of the 3rd ACM SIGCOMM workshop on Information-centric networking*. ACM, 39–40.

[16] Van Jacobson, Diana K Smetters, James D Thornton, Michael F Plass, Nicholas H Briggs, and Rebecca L Braynard. 2009. Networking named content. In *Proceedings of the 5th international conference on Emerging networking experiments and technologies*. ACM, 1–12.

[17] Michał Król and Ioannis Psaras. 2017. NFaaS: named function as a service. In *Proceedings of the 4th ACM Conference on Information-Centric Networking*. ACM, 134–144.

[18] Michał Król and Ioannis Psaras. 2018. SPOC: Secure Payments for Outsourced Computations. In *NDSS’18 Workshop on Decentralised IoT Security and Standards (DISS)*. https://www.ee.ucl.ac.uk/~ipsaras/files/spoc-payments.pdf

[19] Anil Madhavapeddy and David J. Scott. 2013. Unikernels: Rise of the Virtual Library Operating System. *Queue* 11, 11 (2013).

[20] Priya Mahadevan, Ersin Uzun, Spencer Sevilla, and J.J. Garcia-Luna-Aceves. 2014. CCN-KRS. *Proceedings of the 1st international conference on Information-centric networking - INC ’14* (2014). https://doi.org/10.1145/2660129.2660154

[21] Dima Mansour, Torsten Braun, and Carlos Anastasiades. 2014. Nextserve framework: Supporting services over content-centric networking. In *International Conference on Wired/Wireless Internet Communications*. Springer, 189–199.

[22] Spyridon Mastorakis, Alexander Afanasyev, Ilya Moiseenko, and Lixia Zhang. 2015. ndnSIM 2.0: A new version of the NDN simulator for NS-3. *NDN, Technical*

Report NDN-0028 (2015).

[23] Satyajayant Misra, Reza Tourani, and Nahid Ebrahimi Majd. 2013. Secure content delivery in information-centric networks. *Proceedings of the 3rd ACM SIGCOMM workshop on Information-centric networking - ICN ’13* (2013). https://doi.org/10.1145/2491224.2491228

[24] Ilya Moiseenko, Mark Stapp, and David Oran. 2014. Communication patterns for web interaction in named data networking. In *Proceedings of the 1st ACM Conference on Information-Centric Networking*. ACM, 87–96.

[25] Ilya Moiseenko, Lijing Wang, and Lixia Zhang. 2015. Consumer/producer communication with application level framing in named data networking. In *Proceedings of the 2nd ACM Conference on Information-Centric Networking*. ACM, 99–108.

[26] Shashank Shanbhag, Nico Schwan, Ivica Rimac, and Matteo Varvello. 2011. SoCCeR: Services over content-centric routing. In *Proceedings of the ACM SIGCOMM workshop on Information-centric networking*. ACM, 62–67.

[27] Neil Spring, Ratul Mahajan, and David Wetherall. 2002. Measuring ISP topologies with Rocketfuel. *ACM SIGCOMM Computer Communication Review* 32, 4 (2002), 133–145.

[28] Suman Srinivasan, Amandeep Singh, Dhruva Batni, Jae Woo Lee, Henning Schulzrinne, Volker Hilt, and Gerald Kunzmann. 2012. Ccnxserv: Dynamic service scalability in information-centric networks. In *Communications (ICC), 2012 IEEE International Conference on*. IEEE, 2617–2622.

[29] Christian Tschudin and Manolis Sifalakis. 2014. Named functions and cached computations. In *Consumer Communications and Networking Conference (CCNC), 2014 IEEE 11th*. IEEE, 851–857.

[30] Lixia Zhang, Deborah Estrin, Jeffrey Burke, Van Jacobson, James D Thornton, Diana K Smetters, Beichuan Zhang, Gene Tsudik, Dan Massey, Christos Papadopoulos, et al. 2010. Named data networking (ndn) project. *Relatório Técnico NDN-0001, Xerox Palo Alto Research Center-PARC* (2010).