In-Network Live Stream Processing with Named Functions

Christopher Scherb
University of Basel, Switzerland
christopher.scherb@unibas.ch

Claudio Marxer
University of Basel, Switzerland
claudio.marxer@unibas.ch

Urs Schnurrenberger
University of Basel, Switzerland
urs.schnurrenberger@unibas.ch

Christian Tschudin
University of Basel, Switzerland
christian.tschudin@unibas.ch

***Abstract*—Information Centric Networking (ICN) is designed for content distribution. Therefore, it is a solid basis for data streaming applications. In general, data streaming applications have become very popular in networks where many sensors provide their data. Thereby, users often want to fetch the information of several data streams and aggregate or analyze them. Normally, such results are smaller than the input data. Therefore, it can be beneficial to compute the analysis on a data source. Especially for sensor devices this may not be possible due to their limited computation power. Additionally, sensor networks can be widely distributed, meaning that to compute the analysis on one sensor device requires to aggregate all streaming data on it. Hence, it makes sense to aggregate or analyze the streaming data inside the network. Here Named Function Networking (NFN) can help. It is an ICN generalization which enables users to request results of in-network computations. We show how user defined in-network data analysis can look like in several scenarios such as analysis of live streaming GPS tracking data. Thereby, we present how the NFN naming scheme can be used to name data streams. Furthermore, we address the problem how to enter an existing data stream.**

I. Introduction

The Web was conceived as a hypertext infrastructure for rather small documents like Web pages which would be transferred by the underlying Internet in one go. In the past ten years, however, streaming has become the dominant mode of content delivery for two major reasons (beside the classic streaming applications like telephony): First, documents have become very large (e.g. movies), mandating a transfer mode where content is delivered piecewise, and second, an increasingly large number of sensors is attached to the Internet which generates a continuous flow of items towards the servers that process these streams. In this paper, we explore how an Information Centric Network, which by design is specialized in delivering individually-named small data items, can optimally support streaming of data as well as their processing, leading to a "stream-aware network".

Processing data streams inside the network has many potential benefits and has been proposed since long: In-network data fusion, for example, suggests that combining the information from neighboring data sources avoids transferring multiple streams to maybe several receivers which all would have to merge the individual streams in a redundant fashion. The problem, though, is that unlike universal bit transfer, processing is highly application-specific and potentially computationally expensive. In our solution, we expose to the network the operations on the streams and let the network find out whether and where in-net processing is feasible, and whether processing results can be cached instead of being recomputed. We demonstrate such a system by an number of scenarios of increasing complexity:

- Data filtering: Transform GPS coordinates of a body sensor data stream "on the fly", for privacy reasons.
- Data aggregation: Derive from one data stream another stream that combines multiple values (e.g. average) at another rate.
- Data correlation: Detect spatio-temporal events like e.g. deciding whether two runners are running together.

As we will show, these examples lead to complex implementation issues like namespace design for the data samples, lookup services to "connect" to an already ongoing stream, or the naming of operations on multiple streams. Our system makes use of named functions, which is a generalization of the named-data approach of ICN. Our contributions are:

- A methodology for naming streams and their data samples, amendable to in-network processing and recursive treatment as yet-another-stream.
- Study on common problems to be addressed, including
  - how to identify a position in a data stream
  - how to join a live stream
  - how to process a data stream
  - how to match data parts from different streams

The paper is organized as follows: In Section III we describe how we apply data streaming in ICN and how to handle computations on data streams. As a test case we used live GPS data tracking and perform tests with several applications inside the NDN testbed. We describe the test scenarios as well as their behavior in Section IV and the test environment in Section V. In Section VI we introduce a solution to secure data streams.

ISBN 978-3-901882-94-4

II. Background

A. Information Centric Networking

All current derivatives of the ICN paradigm according to Van Jacobson [1] have in common that they address content directly by name. This is a paradigm shift since users have simply to ask for a content's name to get the desired data instead of going first to a specific network location and searching there for the data. This shift is mainly legitimized by changed user demands but also by the wish of overcoming limitations of the original networking paradigm. Supporting the understanding of core aspects of this paper, this is shown for example by three comparisons:

1) Compared to the limited IP address space cardinality (2^{32} for IPv4 and 2^{128} for IPv6) there is potentially an inexhaustible amount of names because they can be of arbitrary length. Moreover, they are extendible and thereby can fulfill additional functions than just *naming* or *addressing*.

2) IP addresses identify network locations whereas names identify content/data. Amongst others, this has direct implications on the forwarding process. Normally, there exists just one *best* path to a network location (IP address). In comparison, identical names - or identical data respectively - can be available in different network locations, meaning that there are multiple *best* paths to the data. Accordingly, the IP Forwarding Information Base (FIB) contains just one entry for a certain IP address or prefix while an ICN FIB can have several entries for one name or name prefix. In doing so, the problem of multi-path routing is solved very elegantly.

3) A further important reform are standard in-network short-term caches, referred to as Content Stores (CS). Content Stores can keep recently seen data for a shorter or longer period according to an implemented strategy or policy. This is in strong contrast to IP network which are generally stateless. The main potential benefits of Content Stores are shorter latency and less congested channels because data may be available closer to the consumer thereof. This capability is a requirement to perform in-network computations because it facilitates the storage of interim and final results of computations. Note that results are nothing else than yet another chunk of data.

The following subsection briefly introduces our approach on how to express network computations.

B. Named Function Networking

In an ICN network, a user can request data by using an identifier, usually a data name. But this is only a special case of requesting a result of a computation [2]. The computation itself is encoded in the name of the Interest message. We use λ-calculus as encoding since it is a compact way to represent computations and can be represented as string. The encoded computations inside the ICN name define a workflow, while it is possible to call high level functions from within the λ-calculus. These high level functions are stored in ICN content objects. Thus, they are seamlessly transferable over the network.

By manipulating computations inside the network, NFN is enabled to optimize the location where a computation is executed by using metrics such as computational load on a node or channel capacity utilization between nodes [3]. Optimally, a NFN tries to compute a result on the path to the data sources to reduce the network traffic and to distribute the computational load. This makes NFN to a good example for fog computing. But it is possible to *pin* functions on nodes so that they cannot be transferred over the network. This might be required for security or copyright reasons. In this case, the NFN has to transfer the data to the pinned function.

NFN is compatible to the ICN implementations CCNx and NDN. Since a λ-expression has no meaning to the routing system of CCNx or NDN, one of the names of the Interest message is prepended.

This works since both CCNx and NDN use longest prefix matching for routing. The first components containing the prepended data name are valid for the CCNx/NDN routing system, and all other components are ignored because of the longest prefix matching. Thus, it is possible to use NFN in a heterogeneous network where CCNx or NDN nodes store and forward data and NFN nodes perform computations. In this paper we represent NFN-functions as pseudo code like:

```
func <functionname> (<params>):
  <functioncode>
  return value <....>
```

In the following, we simplify NFN-names carried by an Interest to common function call notation

```
f(param1, param2)
```

instead of explicitly writing out the λ expressions with a prepended parameter:

```
param1 λ x call f x param2
```

III. Data Streaming and Stream Processing

In this paper we distinguish between live and on-demand streaming. While for live streaming there are only the latest data elements available, for on-demand streaming any data element is accessible at any time. Live streaming perfectly fits to ICN, since it offers a efficient way for distributing content to multiple receivers[4]. Since a data stream is a sequence of data elements which are accessible over time, stream processing strongly depends on the representation of the data elements [5]. In the following we provide a data elements representation for ICN streams and explain how to use them in NFN for processing. Thereby, we focus on live streams. A live stream is a stream which provides the latest data when they are published. The beginning and the ending of the stream are unknown. Moreover, an on-demand stream is a special case of a live stream where each position can be requested anytime. An on-demand stream can be handled the same way as a live stream.

A. Streaming in ICN

For data streams it is required that even a single data element contains certain meaningful information. Thus, each data element has to be addressable since ICN communication

is receiver driven. In our case, we use a sequence number as the last part of the name for distinction.

Because ICN is receiver driven and has a symmetric Interest/Data packet flow, it is required to request each data element individually by expressing an Interest. The data stream with the prefix `/provider/streams/stream1` can be requested with the Interest message:

```
I: /provider/streams/stream1/s4
```

The component `s4` refers to the fourth data element of the stream. Some ICN implementation use longest prefix matching for searching the content. In this case, a user can express a shorter version of the Interest message by removing the sequence number. In this case, the user would receive an arbitrary data element of the stream. After receiving one data element of a data stream, a user has to increase the sequence number to receive further elements since sequence numbers are sorted in ascending order. The naming scheme for data streams is quiet the same as used for VoCCN[6]. We explicitly choose a common name scheme to stay compatible with existing work.

B. Joining a Stream

The current position of a live stream is unknown in advance, but it is required to be aware of it in order to join. For the special case of on-demand streaming a user always starts a stream by requesting the first part since each part is available. For live streaming only the contemporary data elements are available, and maybe some previous data elements, stored in the network's content stores. For that reason it is not possible to start with the first part. Thus, joining a live stream is more complicated. We propose two ways to join a data stream: Increasing sequence numbers and manifest files.

1) Increasing Sequence Numbers: For ICN networks using longest prefix matching for searching the content, it is possible to join a data stream without adding a sequence or chunk number. Thereby, the caller will receive an arbitrary part available in the network. The received content object will contain a sequence number. By increasing the sequence number until there is no content object with a higher sequence number available, it is possible to reach the latest position of the stream. For live streams which may produce a lot of parts in a short time period, it may take some time until the user reaches the latest part. To reduce this effort, the sequence number can be increased exponentially. Thereby, it is possible to miss the latest sequence number. If a requested sequence number is not available, the sequence number can be halved and increased again. Still, this requires some network communication and in ICN networks using exact prefix matching this does not work.

2) Stream Manifest Files: An alternative method to join a live stream is manifest-based. A manifest file contains meta information about the data stream, e.g. the start time and the sampling interval. With these information it is possible to approximate the latest chunk number. Since a manifest file is usually stored in a data object, it is possible to join the data stream by only requesting the manifest file. Therefore, it is required that the manifest file is available for the entire duration of the stream.

C. Stream Processing with NFN

A stream processing environment computes new streams from existing ones. The challenge is to setup an environment where users can deploy their own processing functions and are not restricted to the reuse of existing functions. This is where NFN can demonstrate its strengths. A user defines his own computations and NFN will find a location to execute them. For stream processing, the result of a NFN computation will be cached and can be reused. Because a new stream is created this way, it is possible to chain NFN computations. To identify the data element of the stream on which the computation is defined, it is required to add the sequence number or a range of sequence numbers. The name of the new stream will be the name of the computation (a λ-expression). The data element of the stream is identified by the input sequence numbers. When a computation is defined on multiple data streams, the sampling rate can be changed. Depending on the computation itself, a new result is available every time when one of the streams publish a new element or only when multiple did.

To describe a data stream computation we use a graphical representation as shown in Figure 1 where a processing function combines several input streams to one output stream. The output stream can also be the input for another processing function. Our graphical representation focuses on input data, the processing function and the output data because NFN hides the network from a user.

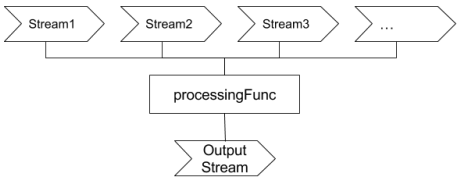

Fig. 1. Stream processing schema

Just as ICN, NFN relies on a receiver driven approach. As Figure 2 shows, each processing of a part must be triggered by a request of a client.

A client expresses an Interest message which contains a program encoded in λ-calculus. This program consists of a function call `processingFunc` on a part of a data stream `Stream1`. The part is identified by using the sequence number `seqNum`:

```
I: processingFunc(
       /provider/streams/stream1/s4,
       /provider/streams/stream2/s7,
       /provider/streams/stream3/s1, ...)
```

The Interest message is routed towards the data, the function code is fetched, and the computation is executed next to the data source (red/left part). The processing function has to request the relevant data parts from the data source (blue/right part).

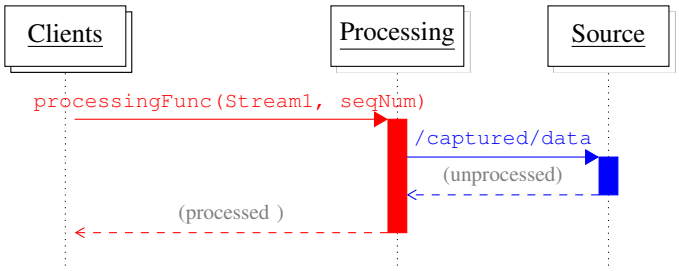

Fig. 2. Requesting a single part of a processed data stream

We assume that the source of the data stream has no computation unit. Thus, it is not required that a publisher has high computation power to publish a data stream in different formats to many users, such as it is required in the current Internet. YouTube needs high computation power to transcode a video stream to different resolutions for different devices. When processing a data stream in NFN, the goal of the network is to find a processing unit as close as possible to the data source. This way, the probability that other requests can be satisfied by a cached result is increased.

IV. GPS Stream Processing with NFN

In the following we demonstrate the behavior of NFN based on live streaming GPS tracking data. We show that NFN provides a simple possibility for personalized in-network data stream processing. With mobile data connection, it is possible to stream GPS position data live to the Internet where other people can view the current position. We choose a scenario where GPS sports watches or a smartphone app is used to track a running workout. We have three processing functions. We start with a simple example which is used to filter personal information out of a GPS stream. We continue with computing the running distance where data has to be aggregated over a time period. Lastly, we detect if two runners are running close to each other. In that case, data of multiple sensors have to be combined. We assume that a data element contains exactly one GPS tracking data point.

A. Filtering

Activity tracking produces data which most people do not want to be released to the public. Nevertheless, users might want to share the workouts they performed. For example, personal time-location data might reveal individual habits and indicate a person's place of residence, work or recreational activities. On the other side, refusing to contribute personal data often means to forgo useful applications. We think that for many users providing filtered data is a viable middle course. Filtering means to reduce the information content of certain data by still preserving enough information to run certain applications. By translating the workout in a way that the starting point is on the origin of the geographical coordinate system, all personal location information is removed. The sole track information is much more difficult to misuse for unintended purposes. The `gps_origin_filter` take one input data stream and the result will be a second stream. This is exactly the scenario we described in Figure 2. The filter requests the first data point and subtracts it from the data point to be filtered.

B. Aggregation

In order to analyze data of a sensor, the challenge is to collect the data in an efficient way. We demo data aggregation in our streaming based NFN scenario by computing the distance completed by a runner. Figure 3 shows the processing function receiving multiple data elements over time and combines them to an aggregated result.

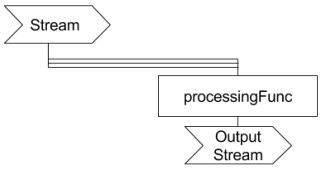

Fig. 3. Time wise data aggregation from streaming data elements

This is done by adding up the differences between the single GPS data points. Therefore, it is required to know the current and the last data points as well as the already covered distance. We propose a NFN function that collects a certain number of data points and computes the distance between them:

```
aggregate(stream, s4, s321)
```

where `stream` is the data stream, `s4` is the start position and `s321` is the end position. The function `aggregate` has to request all parts between the start and the end position. For a live stream, it is required that the starting point is available in the moment the computation starts. The NFN function can aggregate the data points of the available parts while waiting for new data points to be published. This behavior can be achieved by splitting the aggregation to a sub problem:

```
aggregate(stream, s4, s321) =
    aggregate(stream,s4, s320) +
    aggregate(stream, s320, 321))
```

This way, it is not required to store every data point locally, but only the intermediate result. The output of the data stream `aggregate(stream,s4, s320)` become the input for computing the next element of the data stream `aggregate(stream, s4, s321)`.

C. Simple Event Detection

Event Detection names the task to supervise data stream(s) in order to notify if certain pattern(s) occur [7]. Such a scenario is described in Section III where multiple input data streams are combined. During a marathon race, two persons may run together unless one becomes faster. If the distance between two runners is smaller than a threshold, they are still running next to each other (Algorithm 1, line 9). The input for this computation are two data streams and the

position (sequence number) `pos1` in the data stream `s1`. It is important to choose the right sequence number for the other data stream `s2`. The timestamp of the GPS data points should be as close as possible, otherwise we would not detect if both runners are running together. Therefore, the function `detect_together` has to find a matching data point given the position in one stream. By requesting the last few data points (line 6) of the second runner, it is possible to find the closest data point (line 7). The function `request_latest` (line 3) performs an operation to find the latest data element in a stream (see III-B) while the function `request` just requests a data element given a certain sequence number.

Algorithm 1 Event Detection: Are two athletes running close at a certain point in time?

```
1  function detect_together(s1, s2, pos1):
2   d1 = request(s1, pos1)
3   pos2 = request_latest(s2).pos
4   closestPt = request(s1, pos2)
5   for i in 0..5:
6     d2 = request(s2, pos2-i)
7     if d2.ts-d1.ts < closestPt.ts-d1.ts:
8       closestPt = d2
9     if d1.gps - closestPt.gps < thrshld:
10      return true
11  end for
12  return false
13 end function
```

V. Deployment

Since ICN and NFN rely on named data objects and streams, and not on nodes in the network, the topology of the underlying network is completely hidden to the user. The user expresses a computation such as on a local computer and the network resolves it. To verify our test scenarios, we made use of the NDN testbed. Thereby, we connected a NFN node to the testbed node of the University of Basel (UB). To issue our Interest messages, we connected to the node of the University of California, Los Angeles (UCLA). The testbed node of the UCLA forwards the Interest messages to the node of UB and to our NFN node. The NFN node initiates the computation by requesting the required data from the data streams. We added the data streams on the node of the UB. The computation is executed on the NFN node in the network and only the result is transferred over the testbed. Figure 4 shows the topology of the test setup.

VI. Secure Streaming

In several cases it is required to secure a data stream. For example, in the introduced use cases it makes sense to secure streams consumed by a `gps_origin_filter` since the unfiltered content (Figure 5, lower right part in green) should not be publicly accessible. To overcome this issue, the consumed and unfiltered stream delivers only parts which are

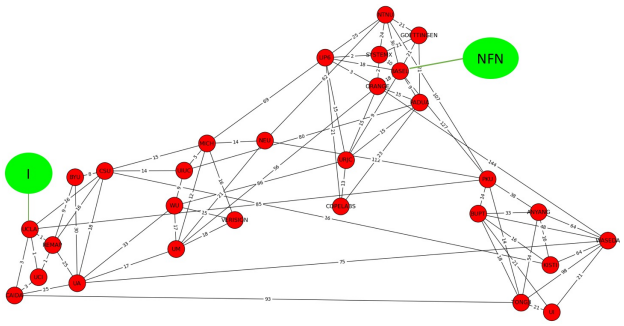

Fig. 4. NDN testbed topology with NFN node

symmetrically encrypted with a certain key (chosen by the capturing device). An instance of the `gps_origin_filter` can request that symmetric key by sending the public key of an available public/private key pair (Figure 5, upper right part in blue). If the public key is contained by a list of trusted identities, the symmetric key is encrypted therewith and returned. The requesting consumer is able to encrypt the symmetric key by making use of the locally available private key.

A function holding a trusted public/private key pair needs to be pinned down on a certain node. This avoids that privileged identities travel through the network. However it is still possible to distribute the computations in the network by setting up several instance of `gps_origin_filter`.

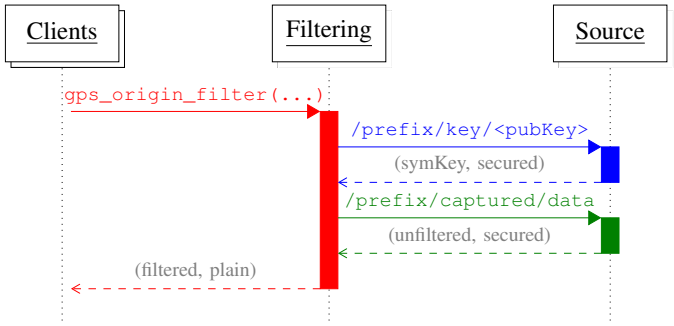

Fig. 5. Secured streaming

VII. Discussion & Future Work

We propose an approach for data stream processing which is based on NFN. Instead of loading a data stream to a client and processing it there, we perform in-network operations. There are different data stream processing models where fixed processing units are installed. In that case, the user has to choose one of the available processing units and use available functions. NFN enables users to define in which way the data stream should be processed. By optimizing the location of the data stream processing using mobile function code, NFN ensures that data is processed on the direct network path from the data source/producer to the data requester. This

reduces the network load compared with classic approaches where that data stream has to be transmitted over specific processing units. Mobile code can be distributed and different computations can be distributed to multiple nodes, similar to S4 [8]. Moreover, the computation can be changed during the data stream, since all requests are receiver driven.

Thereby, some of the key features of ICN are preserved. The data can be received from any node storing or caching the data. Furthermore, data and results of processing can be cached in the network. The disadvantage of receiver driven live stream processing is that the computation is only initialized if there is a request to do so. Hence, (intermediate) results are generally not precomputed as soon as new input data becomes available.

Since NFN relies on ICN names to express computation, a solid naming scheme for data streaming is required. Our naming scheme is very similar to those used in VoCCN [6]. Transferring the naming scheme of a static data stream to a dynamic NFN stream is straight forward. As long as we can address a position in a data stream using ICN names, we can just wrap NFN computations around it. This is one of the key properties in NFN to be compatible with existing ICN solutions. Nevertheless, it is required to define a mechanism how a live stream can be joined, since the current position is unknown. An additional challenge are multiple data streams with different sampling rates used as input of a computation. In this case, either a directory with current positions or user knowledge over the sampling rates is required.

In-network data processing reduces the load on the network and on client devices. Especially for mobile devices it is beneficial to use as less own resources as possible because computation and power capacity are limited. Thus, it is required to deliver preprocessed data in a way that as little client-side resources as possible are used. With NFN, users get the possibility to precisely express their personal needs.

Up to this point we examined only a secured transmission between the data source and the processing/filtering node. That is why we assume that filtered data can be accessible for everyone. To protect data and to restrict the access to authorized persons, it is required to encrypt the data between the processing/filtering node and the users as well. This was not the focus of this paper since there are already concepts available how manage the access to results of NFN computations [9]. Nevertheless, the example showing how to secure the communication between data source and processing node demonstrates that the NFN access control mechanism also applies to stream processing given that we use a regular NFN name scheme.

VIII. Conclusion

Flexible in-network data stream processing is a challenging task. In this paper we showed how NFN can be used for stream processing. NFN is a generalization of ICN that enables the user to request results from the network. Moreover, NFN hides the network in a way that users can express computations similar as on a local computer. Thus, the network itself decides where the computation is executed and it can optimize its load. We introduced a simple data streaming model where single data elements are identified with sequence numbers. By processing one data stream with a function, a new data stream is created. This new data stream can be used as input for another processing function. Handling stream processing becomes very similar to local workflow management. A user defines the input data stream and the computations to be applied. The network will deliver the result. It is not required to download the streaming data first or to connect to a remote machine to compute the result. To test our implementation, we used live tracking data from GPS devices. Thereby, we focused on three test applications: data filtering, data aggregation and event detection in live streams. Since ICN networks are usually receiver driven, our data filtering implementation relies on a pull based approach. The goal for the data filtering operation was to reduce the information content in a way that the user's privacy is protected. Therefore, it was required to introduce a security mechanism to prevent unauthorized users to request the original data.

References

[1] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. H. Briggs, and R. L. Braynard, "Networking Named Content," in *Proc. of the 5th Int. Conf. on Emerging Networking Experiments and Technologies*, ser. CoNEXT '09. New York, NY, USA: ACM, 2009, pp. 1–12.

[2] C. Tschudin and M. Sifalakis, "Named Functions and Cached Computations," in *CCNC, 2014 IEEE 11th*, 2014.

[3] M. Sifalakis, B. Kohler, C. Scherb, and C. Tschudin, "An Information Centric Network for Computing the Distribution of Computations," in *Proc. of the 1st Int. Conf. on ICN*, ser. ICN '14. ACM, 2014.

[4] C. Westphal, S. Lederer, D. Posch, C. Timmerer, A. Azgin, W. S. Liu, C. Müller, A. Detti, D. Corujo, J. Wang, M.-J. Montpetit, and N. Murray, "Adaptive Video Streaming over Information-Centric Networking (ICN) – RFC 7933," Internet Engineering Task Force, 5177 Brandin Court Fremont, California 94538 USA, Tech. Rep., aug 2016.

[5] H. Xu, Z. Chen, R. Chen, and J. Cao, "Live Streaming with Content Centric Networking," in *Networking and Distributed Computing (ICNDC), 2012 3. Int. Conf. on*, Oct 2012, pp. 1–5.

[6] V. Jacobson, D. K. Smetters, N. H. Briggs, M. F. Plass, P. Stewart, J. D. Thornton, and R. L. Braynard, "VoCCN: Voice-over Content-Centric Networks," in *Proc. of the 2009 workshop on Re-architecting the Internet*. ACM, 2009, pp. 1–6.

[7] L. Probst, I. Giangreco, and H. Schuldt, "PAN – Distributed Real-Time Complex Event Detection in Multiple Data Streams," in *Distributed Applications and Interoperable Systems*. Springer, 2016, pp. 189–195.

[8] L. Neumeyer, B. Robbins, A. Nair, and A. Kesari, "S4: Distributed Stream Computing Platform," in *2010 IEEE Int. Conf. on Data Mining Workshops*, Dec 2010, pp. 170–177.

[9] C. Marxer, C. Scherb, and C. Tschudin, "Access-Controlled In-Network Processing of Named Data," in *Proc. of the 2016 conference on 3rd ACM Conf. on Information-Centric Networking*. ACM, 2016, pp. 77–82.