Persistent Interests in Named Data Networking

Philipp Moll
Institute of Information Technology
Alpen-Adria-Universität Klagenfurt
philipp.moll@aau.at

Sebastian Theuermann
Institute of Information Technology
Alpen-Adria-Universität Klagenfurt
sebastian.theuermann@aau.at

Hermann Hellwagner
Institute of Information Technology
Alpen-Adria-Universität Klagenfurt
hermann.hellwagner@aau.at

***Abstract*—Recent research in the field of Information-Centric Networking (ICN) shows the need for push-based data transfer, which is not supported in current pull-based ICN architectures, such as Named Data Networking (NDN). IoT deployments as well as emergency notifications and real-time multimedia communication are well suited to be realized using the ICN principles, but experience challenges in pull-based environments. Persistent Interests (PIs) are a promising approach to introduce push-like traffic in Interest-based ICN architectures such as NDN. In this paper, we explore the characteristics of PIs and discuss advantages and disadvantages of using them. We provide an efficient solution for preventing so-called Data loops, which are introduced by giving up NDN's one-request-per-packet principle. Furthermore, we investigate the performance of PIs compared to classical Interests in terms of the computational complexity of forwarding and discuss possible applications of PIs.**

***Index Terms*—Information-centric networking, Named Data Networking, Persistent Interests, push-based traffic**

I. Introduction

In Information-Centric Networking (ICN) research, a trend to strictly pull-based architectures is noticeable. In Named Data Networking (NDN) [1], a promising Future Internet architecture, clients are forced to request every single Data packet by a so-called Interest packet. The desired Data is specified in the Interest by a system-wide unique name. Requested Data always follows the reverse path of the requesting Interest back to the client, enabled by the Interest laying breadcrumbs on each forwarding network node. These breadcrumbs are realized as entries in the Pending Interest Table (PIT), which stores incoming and outgoing interfaces of Interest packets.

The one-request-per-packet principle used in NDN leads to a strict pull-based architecture. When looking at current ICN research, we see a need for push-based traffic for certain tasks, such as for pushing sensor data from low-power IoT devices to a gateway [2] or for sending emergency notifications in delay tolerant networking [3]. Besides those applications, multimedia streaming could benefit from push-based traffic as well. Due to low latency requirements and variable bitrate codecs, it is challenging to use NDN for conversational services.

In the first pull-based implementation of a conversational service over NDN [4], Interests pre-requested Data packets to be produced in the future, in order to keep the delay between the generation of data at the producer and playback of the data at the client as low as possible. With modern variable-bitrate audio and video codecs, the number of produced Data packets varies each second, which makes pre-requesting Data packets difficult.

NDN-RTC, a library for real-time streaming over NDN [5], tackles this issue by estimating the number of future Data packets. The challenge for this approach is to find the correct number of Interests to issue. If too few Interests are issued, additional Interests have to be sent in order to fetch missing Data packets, which increases the latency. Too many issued Interests increase the network overhead and may mislead adaptive forwarding strategies which rely on metrics such as the Interest satisfaction ratio, which is untruly decreased by Interests which are sent due to overestimation of the number of Data packets to be produced in the future.

Another approach is to send traffic in a push-like manner. Amadeo et al. [6] propose methods for pushing IoT traffic reliably, mostly considering small traffic volumes and local area networks. In addition, the focus on fully reliable delivery introduces overhead, which some applications such as conversational services would not require. Investigations of Persistent Interests (PIs) [7] showed that PIs are well suited to enable push traffic for conversational services in NDN. A PI is a modified Interest which does not only request one Data packet, but subscribes to a stream of Data packets and enables the producer to send as many Data packets as it produces during a predefined time interval and thereby enables push-based traffic in NDN. This keeps the latency low and decreases overhead. Nevertheless, PIs bear some challenges, such as increased forwarding complexity [8] and security issues.

This paper presents an analysis of PIs including their advantages and shortcomings compared to the classical Interest emitting approach. Moreover, we present an efficient solution to prevent Data loops which can arise when using PIs. We introduce several ways of implementing push-like traffic including a discussion of the pros and cons of the variants. Beyond that, we evaluate our algorithm for Data loop prevention by means of network simulations and we present a comparison of the forwarding effort of PIs and the forwarding effort of classical Interests on real hardware. Finally, we discuss the applicability of PIs for different applications. In addition, we contribute the implementation of PIs including our presented Data loop prevention algorithm as open-source software to allow other researchers to build on our work.

II. Persistent Interests

The strict pull-based nature of NDN is not perfectly suited for all types of applications. Tsilopoulos et al. [9] first asserted that different traffic types need to be supported in order to fulfill the demands of all types of applications. Therefore, the idea of Persistent Interests (PIs) was proposed. This concept was implemented and investigated in [7]. In this implementation, PIs carry a type field, which can be used to distinguish PIs from classical Interests, which is required because a PI must not be deleted from the PIT when a single Data packet is received. It stays active in the PIT until its PI-defined lifetime times out. In order to prevent a PI from timing out, the PIT entries have to be refreshed in regular intervals. This is achieved by periodically sending a new PI carrying the same name as the old one to the producer; this is referred to as a refresh PI. Analyses in [7] showed the suitability of PIs for conversational services over NDN. Besides increasing the estimated user satisfaction, PIs lead to a significant decrease of traffic due to the reduced number of sent Interests.

The authors of [7] further discovered that forwarding strategies intended for classical Interests are not applicable for PIs, because metrics such as the Interest satisfaction ratio can not be calculated when using PIs. Therefore, an adapted version of the best route forwarding strategy [10] was developed. A first adaptive forwarding strategy for push-based traffic in NDN was published in [8], which uses probing results in order to rate the quality of different paths through the network.

In the following, we present a deeper analysis of PIs in order to assess whether they are promising for delivering data of multi-party real-time communication, produced during video conferencing or in multi-player on-line games, in large scale networks.

A. Mitigating Data Loops

Flow balance, which means that each emitted Interest expects exactly one returning Data packet, is one characteristic of NDN. This balance is disrupted when using PIs because PIT entries, which are linked to a PI, do not only stay valid for one Data packet, but for all Data packets which are produced during the lifetime of the PI. In classical NDN where flow balance is active, each Data packet should arrive only once at a node. If it arrives twice, which is possible when Interests are sent redundantly on multiple faces, the redundantly arrived Data packet is dropped as an unsolicited packet, because the corresponding PIT entry is already satisfied.

PIT entries of PIs behave differently. A PI-PIT entry is not satisfied when a matching Data packet is received. Because of this, a duplicated received Data packet is not recognized as such and is forwarded redundantly towards the client.

Another related, but more severe problem arises when using adaptive forwarding strategies. When a forwarding strategy decides to redirect the data flow of a PI to a new path because the current path's performance is too low, the strategy can only do this by redirecting refresh PIs to the new path. The lifetimes of two consecutive refresh PIs have to overlap in order to prevent timeouts. When redirecting the refresh PI, this fact leads to a short timespan with multiple active PIs for the same data on different paths. This can cause so-called *Data loops* in certain scenarios, such as in the demonstration topology visualized in Figure 1. The client first sends PI 1 (blue) over the upper link. A later sent refresh PI (PI 2, green) for the same data is redirected to the lower link. The lifetimes of the two PIs are overlapping. As visualized, we assume that the forwarding strategy duplicates the PIs in the network. When looking at all active PIs between the center nodes, we recognize that they form a loop in the shape of an "8". Incoming Data packets always follow the reverse path of the PIs, which leads to looping Data packets because the forwarded PIs already collectively form a loop. We now call the loop of Data packets a Data loop, which is visualized in Figure 1 by means of red arrows. Once a Data loop is established, more and more Data packets start looping until all links are congested. Even if this scenario seems to be artificially created, Data loops could be observed in simulations of larger randomly generated networks, such as used for evaluation in Section III-A.

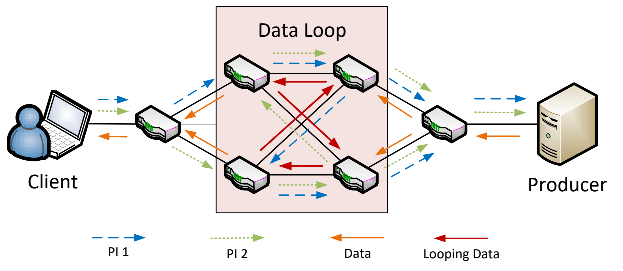

Fig. 1. Looping Data packets in demonstration topology

The naive solution to overcome this issue – remembering all Data packets which were forwarded on a node and checking if an incoming packet is a duplicate – would be effective but computationally expensive and would cause huge performance costs for all incoming Data packets on a node. We further assume that the number of packets which has to be remembered, varies from application to application. In cases where a low number of packets is sent, only a few packets have to be remembered in order to effectively prevent duplicate transmissions. In cases where many packets are delivered, it is important to remember a larger number of packets, which causes memory and processing overhead. Furthermore, packets have to be remembered per PI-PIT entry, because it has to still be possible to re-request a Data packet via a regular Interest. When thinking of core routers, which forward millions of packets each second, we quickly see that the overhead introduced by the naive solution is too high.

The method we developed detects duplicated packets by an algorithm which is inexpensive in terms of processing and in terms of memory costs. The last part of each NDN name, which uniquely identifies a Data packet, is an ascending integer number, referred to as *sequence number*. In a stream of Data packets, such as produced during Internet telephony sessions or during video conferencing, the sequence number of the first Data packet starts at zero and is strictly increasing until the end of the call. The idea of our approach is to use these sequence numbers to remember already transmitted packets in a space-efficient manner. Our adapted PI-PIT entries store the highest transmitted sequence number and a bitvector of predefined length, where each entry represents a transmitted or not transmitted packet. Algorithm 1 shows the detection of duplicated Data packets using these data structures.

Algorithm 1: Check if Data packet is duplicate

```
Data: currentSeqNo, highestSeqNo, seqNoVector
Result: duplicate packet or not
1  diff = currentSeqNo - highestSeqNo;
2  if diff > 0 then
       /* New packet, shift bitvector     */
3      seqNoVector << diff;
4      seqNoVector[0] = True;
5      highestSeqNo = currentSeqNo;
6      return No duplicate;
7  else if -diff >= len(seqNoVector) then
       /* SeqNo too old, default allow    */
8      return No duplicate;
9  else if seqNoVector[-diff] == True then
10     return Duplicate;
11 else
       /* Unseen packet, set bit          */
12     seqNoVector[-diff] = True;
13     return No duplicate;
```

The *currentSeqNo* variable represents the sequence number of the currently received packet, *highestSeqNo* represents the highest received sequence number matching the PI-PIT entry, *seqNoVector* is a bitvector which is used to remember transmitted packets. When a packet with a sequence number higher than the currently highest sequence number arrives, the packet is new and cannot be a duplicate (lines 2-6). If *currentSeqNo* is smaller than the range of the bitvector, we assume that the packet is no duplicate (default allow policy, lines 7-8). If this is not the case, we check if the corresponding position in the bitvector is set to *true*. In this case, the packet was already seen due to duplication or a Data loop. The packet gets dropped, which prevents duplicates and Data loops (lines 9-10). If the corresponding entry is set to *false*, the packet was not seen before and is classified as not duplicated.

In the case that the received Data packet is no duplicate, the packet can be transmitted according to records in the PIT entry. When the *currentSeqNo* is higher than the *highestSeqNumber*, the highest sequence number is set to the new value and the bitvector is shifted by $currentSeqNo - highestSeqNo$ positions (lines 3-4). In the other case, the corresponding bit in the bitvector is set to *true* (lines 12-13).

Performance-wise, the overhead introduced by this algorithm is negligible. Checking for a duplicate requires a lookup in a fixed size vector. Adding a sequence number to the vector means to update one bit in a fixed size vector, or to shift the vector for at most n positions, where n is the length of the vector. Regarding memory overhead, an additional bitvector of predefined size, and an integer remembering the highest sequence number is required for each PI-PIT entry. We assume that the size of the bitvector depends on the number of produced packets per second. In Section III-A, we perform a study in which we evaluate the required bitvector size for Internet telephony.

B. Gaining Flow-Control – Modifications of PIs

In classical NDN, NDN's flow balance allows the consuming application to control what and how much Data it gets. If the consumer senses a congestion, it can decrease the amount of sent Interests and thereby unload the network. As already discussed, the use of PIs disables NDN's flow balance, which means that the consumer loses control over the data flow. In this section, two possible modifications of PIs are discussed, which partially re-enable the client's flow control ability.

1) Next-N-Packets Approach: Instead of requesting all Data packets which are produced in the lifetime of a PI, the Next-N-Packets approach requests a specific number of Data packets, which is defined in the Interest. This number can vary from 1 to N. This allows the consumer to use congestion control approaches similar to the additive-increase/multiplicative-decrease (AIMD) algorithm used in TCP. At the beginning of a connection, no data about the path quality is available. Thus, clients start issuing Interest packets which only request one Data packet. This is comparable to emitting a classical Interest. When no congestion occurs, the number of requested Data packets per Interest is increased by one, whereby the Interest becomes more like a PI. When clients recognize decreasing connection quality, which is likely due to a congestion, the number of requested Data packets is divided by two, whereby the clients gain more control over the data flow.

Not only clients, but also network nodes gain flow control when using this approach. When a congestion occurs in the network, the network node with queues prone to overflow is the first that recognizes the congestion. In order to prevent further congestion, the node is able to regulate the amount of traffic requested by the clients by simply dropping Interests which request a high number of Data packets and notifying the clients by sending a negative acknowledgement. Thereby, the clients get informed about the congestion in the network and are forced to issue Interests requesting fewer Data packets.

2) Range-Interest Approach: A Range-Interest (RI) can be seen as a large Interest. Instead of requesting only one Data packet, a RI specifies a range of sequence numbers which are fetched by one Interest. By allowing varying range sizes, the consumer keeps flow control, because it can specify how many Data packets it wants to get. Similar to the Next-N-Packets Approach, a consumer can adapt the range size according to the current connection quality.

As simple as this approach seems on the first sight, forwarding RIs can be complex. Imagine the following situation: A user requests a range from sequence number 0 to 19. For some reason, a forwarding node has cached the Data for sequence numbers 10 to 14 and directly sends back these five Data packets. When forwarding the RI, the forwarding node now has to split up the range from the original RI into two separate ranges and forward two RIs with smaller ranges, one requesting sequence numbers 0 to 9, the second requesting sequence numbers 15 to 19.

III. Evaluation

In the previous section, we introduced PIs and discussed the possibility of Data loops. Furthermore, we introduced a simple mechanism to overcome this issue in Section II-A. In this section, we first evaluate the efficiency of the proposed Data loop detection mechanism and study the bitvector length for preventing Data loops. Then, we evaluate the performance of PIs by comparing the maximum forwarding capacity when using PIs and classical Interests.

A. Prevention of Data Loops

In this section, we investigate if bitvectors are capable of preventing Data loops originating from path switching in combination with PIs in NDN. Further, we want to find the optimal bitvector length, in order to prevent Data loops while keeping the possible memory overhead low. Therefore, we simulate Internet telephony calls in randomly generated networks using the ns-3/ndnSIM simulation environment [10].

Our network topology consists of five randomly generated, interconnected autonomous systems (AS), each having 20 nodes acting as NDN routers. Link speeds are uniformly distributed between 500 and 1500 kbps between nodes in one AS, and between 3000 and 5000 kbps for links interconnecting multiple ASs respectively. We decided to use such low capacity links in order to allow congestion, which would not occur when using higher capacities.

In the network, we simulate 20 Internet telephony calls between randomly placed clients using the G.711 PCM audio codec [11], which produces 100 packets per second, each carrying 80 bytes of payload. Voice data is requested by using PIs. In order to produce Data loops, the Multicast Forwarding Strategy [10], which duplicates Interests to all available interfaces, is used to forward PIs. In addition, we simulate cross-traffic by randomly placing four client-server pairs, each requesting 800 kbps using classical Interests.

To assess the effectivity of the bitvector mechanism, we count a Data packet which arrives more than once, independent of the incoming interface, as a *duplicate receipt*. A *duplicate transmission* is a Data packet which is sent more than once over the same interface. Duplicate receipts as well as duplicate transmissions can both be seen as overhead and have to be avoided.

When looking at the results in Figure 2, we can see that even very small bitvectors effectively prevent duplicated Data. A bitvector of size two, which means that only the packet with the highest sequence number and its predecessor are remembered, reduces the amount of duplicated received packets by over 60 %.

As we can further see, an increasing bitvector length leads to a decreasing number of duplicates. At a bitvector length of only 100 bits, almost no duplicated transmissions occur. This means that the bitvector recognized almost all incoming duplicates. Doubling the length of the bitvector only leads to minor improvements.

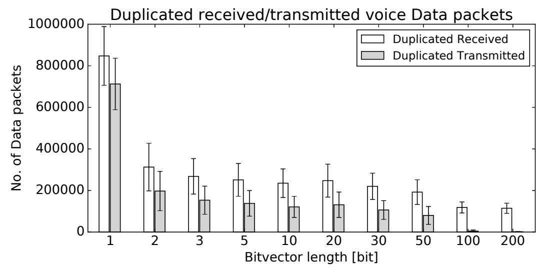

Fig. 2. Total number of duplicated received voice Data packets in the network (received twice or more often on a node) and total number of duplicated transmissions (transmitted twice or more often via the same face). Error bars depict 95 % confidence intervals.

B. Performance of Persistent Interests

After evaluating bitvectors for Data loop prevention, we now compare the performance of PIs to that of classical Interests with respect to the achievable throughput. To be more precise, we investigate how many parallel Internet telephony calls can be forwarded on low-performance network nodes, such as the nodes from the low-cost NDN testbed [12], without a significant decrease in transmission quality. Therefore, we set up a simple network, consisting of a BananaPi router (BPI-R1), which is connecting two more powerful machines that produce the actual voice traffic. One of the more powerful machines hosts the producer applications, while the other one hosts an equal number of consumer applications. The router's only task is to forward the produced Interest and Data packets. This way, we can observe how many packets the router can forward without negative influence on the quality of the voice streams. The emulated voice streams show the same characteristics as in the previous evaluation. The size of the bitvector for Data loop prevention is set to 200 bits. One emulation run takes 1 minute, the number of requested voice streams stays stable during a run and is step-wise increased in subsequent runs. In order to reduce the processing overhead introduced by content store lookups, the content store is disabled in all emulations. When evaluating the PI approach, we configure a PI lifetime of 5 seconds; refresh PIs are sent every 2 seconds.

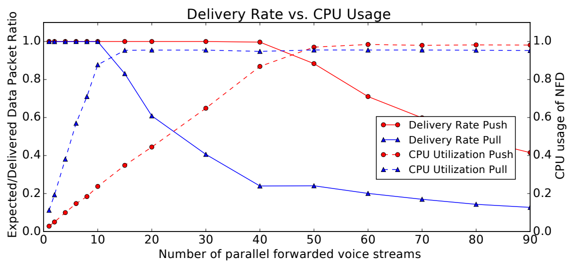

Fig. 3. Comparison of the forwarding performance when using PIs (Push) and classical Interests (Pull) on a low-performance network node.

Figure 3 visualizes the ratio between expected and delivered voice data packets on the client node, as well as the CPU usage of the NDN Forwarding Daemon (NFD) on the low-performance router. These results indicate that our initial assumption that the forwarding effort when using PIs is lower compared to issuing classical Interests holds true. When using classical Interests, the CPU usage is already over 80 % when 10 parallel voice calls are requested. At the same time, the expected/delivered ratio is drastically decreasing when the number of parallel streams goes beyond 10. When using PIs, the expected/delivered ratio stays stable until 40 parallel voice streams, where the CPU usage hits 80 %. From this point on, the expected/delivered ratio is decreasing. We presume that the good performance of PI forwarding results from the fact that fewer Interests are sent when using PIs. The total number of sent packets is reduced by 50 %, which is why a performance increase by the factor of 2 is obvious. In fact, when switching to PIs, we can quadruple the number of voice streams without increasing packet loss. This is not only because of fewer sent Interests, but also because the processing overhead when forwarding Interests is higher compared to the processing overhead when forwarding Data packets. In addition, the memory usage of the NFD on the low-performance router was monitored. We observed a consistently lower memory consumption when using PIs, which most likely results from fewer, albeit longer PIT entries.

IV. Conclusion and Future Work

In this paper, we investigated Persistent Interests (PIs) as an enabler for push-like traffic in NDN. We discussed general characteristics of PIs and introduced several implementation alternatives. Furthermore, we discussed the most important challenges in using PIs and introduced bitvectors as a means of Data loop prevention. Using simulation, we showed that even very short bitvectors (2 bits and larger) reduce the amount of redundantly sent packets by over 50 %. In our scenario, a 100 bit long vector almost completely eliminates duplicated transmissions and Data loops. Because bitvectors for Data loop detection are located in every PI-PIT entry, memory overhead increases with increasing traffic. Nevertheless, the overhead of 100 bits per PI-PIT entry is almost negligible, compared to other mechanisms discussed in Section II-A.

Furthermore, we investigated the computational complexity of PI forwarding by network emulations. To this end, we used a low-performance networking node as an NDN router and compared its forwarding capabilities when using classical Interests and PIs to request Data. We saw that PIs outperform classical Interests, which most likely is caused by the reduction of sent Interests. As the processing of incoming Interests requires high computational effort, the reduction of sent Interests frees available resources, usable to support more concurrent voice streams. We observed that, by using PIs instead of classical Interests, the number of concurrent forwarded voice streams can be quadrupled from 10 to 40 without a decrease in forwarding quality.

Besides the performance benefits of PIs, we also discussed drawbacks in Section II. One drawback of using PIs is the client's reduced ability to exercise flow control. This not only means that the client's ability to react to congestion is reduced, but it also allows for Denial of Service (DoS) attacks by adversaries. Issuing a single PI could request a large data stream, which may already cause congestion in the network. The ability to control the volume of incoming data can be introduced by alternative ways to implement PIs, such as discussed in Section II-B, but it cannot be as fine-grained as in the classical Interest approach without losing its benefits. The alternative ways to implement PIs also can serve as a way to mitigate the potential of misuse of PIs for DoS attacks. Generally we identify the introduction of tried and true ways of flow control and DoS prevention as the most pressing topics for future work. For now, we recommend using PIs rather for small amounts of data in controlled environments, such as for pushing sensor data from devices with energy constraints, than for large amounts of multimedia data on the public Internet. Pushing data from IoT devices can increase battery lifetime because sensors do not have to stay awake until an Interest arrives, but can send the Data as soon as they wake up and immediately re-enter a sleep-state afterwards. Another possible application for PIs is sending emergency notifications. Such notifications have to be sent as soon as they arise in order to react as fast as possible to an alert; polling possible emergency notifications on all possible producer devices using classical Interests would be infeasible.

Software artifacts resulting from this paper are available on GitHub (https://github.com/phylib/PersistentInterest).

References

[1] L. Zhang, A. Afanasyev, J. Burke, V. Jacobson, kc Claffy, P. Crowley, C. Papadopoulos, L. Wang, and B. Zhang, "Named Data Networking," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 44, pp. 66–73, July 2014.

[2] O. Hahm, E. Baccelli, T. C. Schmidt, M. Wählisch, and C. Adjih, "A Named Data Network Approach to Energy Efficiency in IoT," in *Proc. IEEE Globecom Workshops*, 2016, pp. 1–6.

[3] S. Dynerowicz and P. Mendes, "Demo: Named-Data Networking in Opportunistic Network," in *Proc. 4th ACM Conference on Information-Centric Networking*, 2017, pp. 220–221.

[4] 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. ACM Workshop on Re-architecting the Internet*, 2009, pp. 1–6.

[5] P. Gusev and J. Burke, "NDN-RTC: Real-time Videoconferencing over Named Data Networking," *Proc. 2nd ACM Conference on Information-Centric Networking*, pp. 117–126, 2015.

[6] M. Amadeo, C. Campolo, and A. Molinaro, "Internet of Things via Named Data Networking: The Support of Push Traffic," in *Proc. Network of the Future Conference*, 2014, pp. 1–5.

[7] P. Moll, D. Posch, and H. Hellwagner, "Investigation of Push-Based Traffic for Conversational Services in Named Data Networking," in *Proc. IEEE Int. Conference on Multimedia and Expo Workshops*, 2017.

[8] P. Moll, J. Janda, and H. Hellwagner, "Adaptive Forwarding of Persistent Interests in Named Data Networking," in *Proc. 4th ACM Conference on Information-Centric Networking*, 2017, pp. 180–181.

[9] C. Tsilopoulos and G. Xylomenos, "Supporting Diverse Traffic Types in Information Centric Networks," in *Proc. ACM SIGCOMM Workshop on Information-Centric Networking*, 2011, pp. 13–18.

[10] S. Mastorakis, A. Afanasyev, I. Moiseenko, and L. Zhang, "ndnSIM 2: An updated NDN simulator for NS-3," NDN, Technical Report NDN-0028, Revision 2, Nov. 2016.

[11] *G.711: Pulse code modulation (PCM) of voice frequencies*, International Telecommunication Union Std. G.711, 1990.

[12] B. Rainer, D. Posch, A. Leibetseder, S. Theuermann, and H. Hellwagner, "A Low-Cost NDN Testbed on Banana Pi Routers," *IEEE Communications Magazine*, vol. 54, no. 9, pp. 105–111, Sept. 2016.