Networking 2013 1569702861

Implementation and Evaluation of an Information-Centric Network

George Parisis and Dirk Trossen
Computer Laboratory, University of Cambridge
Cambridge, UK
firstname.lastname@cl.cam.ac.uk

Dimitris Syrivelis
CERTH-ITI
Volos, Greece
jsyr@inf.uth.gr

***Abstract*—Information-centric networking (ICN) has been touted as an alternative to the current Internet architecture by several research groups. So far little has been done towards the implementation of ICN network stacks and their evaluation in realistic ICN deployments. In this paper we describe Blackadder, a sophisticated prototype of our ICN architecture, and we present an extended experimental evaluation in high-performance and wide-area testbeds. Our evaluation shows the feasibility of the design and the performance of our prototype in test bed setups. In addition, we contrast our implementation against CCNx.**

***Index Terms*—Information-centric networking, Node Implementation, Click router, Performance evaluation**

I. Introduction

Information-Centric Networking (ICN) is increasingly attracting attention in the networking community. Several technological solutions within a range of architectures have been proposed, such as in [1][2][5][8], with subtle differences but also commonalities that stretch across the approaches.

However, so far, little has been shown about the feasibility and performance of the ICN paradigm. In most cases simulation results or "toy" deployments of simplistic implementations are presented to provide indications about the feasibility and performance of the proposed architectures. We argue that the ICN research field is getting more and more mature; therefore we identify the necessity to design and implement real network stacks that can run in parallel with existing host-centric communication protocols, rather than showcasing ICN characteristics with overlay implementations and deployments running as user-space processes. We strongly believe that along with the evolution of ICN technological solutions, there must exist implementations that can support real deployments and experimentation.

In this paper we present *Blackadder*, a network node implementation of an ICN network stack that leaves nothing untouched with respect to the IP legacy. Our architectural starting point is the one presented in [1] and elaborated in [2] and [3]. Based on this architectural starting point, our contribution is a network node implementation for an ICN internetworking architecture that could run in parallel with TCP/IP, support backward compatibility for legacy applications and be deployed in large networks. We show the feasibility of our approach in a high-speed network as well as in PlanetLab. Specifically, our evaluation shows our prototype performing in line-speed in a Gigabit Ethernet testbed while providing acceptable performance in a PlanetLab setup of ~100 nodes with 36.500 subscribers.

We contribute an open framework for further experimentation in real deployments as well as in simulated environments. We have released Blackadder under the GPL2 license [12] and built it on top of the Click router [7]. Blackadder supports user and kernel space deployments using the same source code. It is ported to the Android OS and OpenWrt and is integrated with the NS3 simulator. Finally, we contrast our work against a similarly ambitious implementation effort, that of the Content-Centric Networking (CCN) [5].

The work presented in [1][2] argues and lays the ground for an information-centric architecture in which *information* is the primary principle. Individual information items are identified through *labels*, which can in turn be organized through *scopes*. This allows for building directed-acyclic graphs of information, manipulated via a *publish/subscribe service model*. This service model is realized through three *core network functions*. The first one, *rendezvous*, matches supply of information to demand for it. This process results in some form of information that is used for binding the information delivery to a network location by the second function, *topology management and formation*, to determine a suitable delivery relationship for the information transfer. This transfer is finally executed by the *forwarding* function. With this separation of functions, the traditional operations of routing and forwarding are decoupled, enabling to trade off options in state management between the various network components. For example, Blackadder allows for removing flow-dependent state from forwarding nodes in favor of route computation during topology formation, inserting the forwarding state into the packet header. It also allows for scoping the realization of these core functions through the notion of dissemination strategies, embedded into the information structures over which the service model operates.

In Section II we present research related to our work. Section III describes how information is managed and disseminated in our ICN approach for different communication scenarios. In Section IV we elaborate on the implementation of Blackadder and in Section V we present an experimental

evaluation of our network node, also in comparison with CCNx. Section VI concludes our paper.

II. Related Work

In the area of content-centric architectures, Haggle [4] and CCN [5] stand out. CCN extends the IP node design with a forwarding information database for a hierarchical naming system (based on the DNS). It also introduces a forwarding function that can be configured based on some strategy for selecting particular interfaces for given named data. However, the function of routing, i.e. the population of the forwarding table based on availability of named data in different domains, is currently undefined. Only a broadcast strategy has been presented so far which is not suitable for most communication scenarios. Haggle provides manipulations of a linked information graph based on publish/subscribe operations. The Haggle component wheel logically separates core functions for information dissemination in a plug-and-play manner. Contrary to our approach, Haggle does not provide a layering structure but resides between the application and the network in a rather monolithic form. The Network of Information (NetInf) [10] operates as an overlay, using application-level event and resolution services. It provides higher-level services to application, which are built on top of IP or even HTTP. Our approach, on the other hand, envisions a parallel network stack that operates independently of IP or on top of it only when this is unavoidable. Other information-centric approaches like in [8] and [11] utilize flat labels to route information in the network. However, no implementations were ever publicly available for any of them. The authors in [9] argue for HTTP and the DNS as the effective waist of the future Internet. This lifts the current IP node design onto the level of efforts like CCN, providing manipulations of hierarchically named data.

Finally, vast research has been done in the field of publish/subscribe notification systems, such as [14] and [15]. However, all of these systems were designed to run as overlays on top of the existing IP network stack contrary to our implementation and the respective network architecture, which can run natively in the network and in parallel with the IP. For that reason we will not elaborate more on systems like these, nor will include them in our experimental evaluation.

III. Information Management and Dissemination

Information lies at the core of our network architecture. In this section, we describe major aspects related to the information-centric nature of our node implementation. We discuss how information is managed by the rendezvous network function and what information semantics are supported by our architecture. Finally, we elaborate on the currently implemented dissemination strategies; i.e. realizations of the core network functions that provide ways for disseminating information.

1) Information Management

Information management is a very important task that is undertaken by the rendezvous network function, as mentioned in Section I. Depending on the used dissemination strategy, the rendezvous function can be implemented: locally to a network node to manage information visible to a single node, centralized in a network node for small domains, or in a distributed, potentially hierarchical, fashion for managing large information spaces across one or more network domains [13]. Information items and scopes are identified using statistically unique labels, *8* bytes long. These labels carry no semantics. Any meaning can be assigned by the entities that produce them as well as by other network entities that utilize the particular information structure for their purposes. Although information can be identified in the context of a scope using a fixed size label, the absolute path from a root of the graph (more than one such path may exist) must be used when accessing the service model exported by Blackadder.

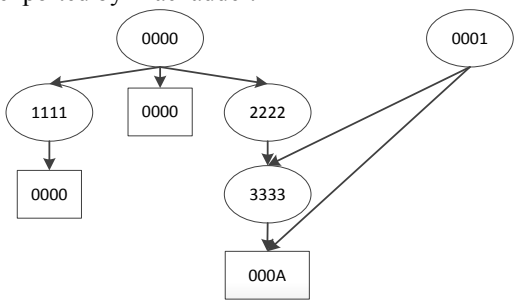

Fig. 1. An example Information Structure

Figure 1 depicts an information graph managed by the rendezvous function. Note that labels in the figure are shorter than in the actual implementation and are shown with their hexadecimal form in order to maintain text's readability. An information item can be published under multiple scopes. Scopes and information items are identified using one or more full identifiers starting from a root scope. With that, the item with label *000A* can be identified using the identifiers: */0000/2222/3333/000A*, */0001/3333/000A and /0001/000A* (slashes are added to improve readability). A publisher or subscriber can use any of these identifiers, depending on its (potentially) partial knowledge of the information structure, to advertise or subscribe to this information item. Global uniqueness of scope and information labels is not enforced. For example, the information item with identifier */0000/1111/0000* has the same label as the root scope */0000* and scope */0000/0000*. As long as the full identifiers of a scope or information item are unique, everything is legitimate.

2) Information Semantics

Blackadder is agnostic to any information semantics. *Immutable* information items can be identified with a statistically unique label under a given scope. As an example, we could assume each version of a document being labelled individually, therefore being individually identifiable within the network. For the application, there needs to be an additional information exchange that disseminates the version identifiers; this is left to the application itself. *Mutable* information items represent information that can change through time. Hence, the application needs to take care of any issues arising from this mutability. Mutable items are important when realizing, e.g. live video delivery, in which video chunks are published using the same identifier representing a unidirectional multicast channel. Finally, determining the identifiers through an

algorithmic function represents a hybrid of the previous approaches. This function forms a channel-like relation between the publisher and subscriber, with items still individually identified through the algorithmic function. Assuming that communicating parties are aware of this relation, the seemingly random identifiers can be associated with each other as they are being received.

3) Information Dissemination

A fundamental principle of our ICN approach is information scoping. Scopes can also differentiate functionally the way information is disseminated among publishers and subscribers. A dissemination strategy can be assigned to a scope and the sub-graph under that scope, defining the way the three core network functions are implemented. Below we describe the currently implemented dissemination strategies and the realization of the core functions for each one of them.

Intra-Node strategy. Here, information is disseminated within the boundaries of a single node providing an information-centric, inter-process communication mechanism. The rendezvous function maintains the information graph, which is visible and accessible only to applications running within a single node. The topology management and formation as well as the forwarding network functions are minimal since they merely dispatch information to the right applications.

Link and Broadcast strategies. These strategies allow a node to disseminate information to its physical neighbours. There is no rendezvous for this strategy; instead, subscribers implicitly subscribe to specific information items, locally to their network stack. A publisher can publish information to a specific link (link) or to all links (broadcast) whenever it wants. If a subscriber exists on the other side of a link, the information is pushed to it. The topology management and formation function is again minimal, since it only needs to find the appropriate network link. The forwarding element stores this information and forwards data to one or more links.

Intra-Domain strategy. In the intra-domain strategy, all core network functions are fully realized. One or more nodes act as rendezvous nodes of the domain and one or more nodes run the Topology Management and Formation function. Topology Managers have a centralized view of the network and create multicast forwarding trees from a set of publishers to a set of subscribers using a shortest-path algorithm (this creation is requested by a rendezvous element after a successful match of publications and subscriptions). The multicast tree is formed by computing source-based LIPSIN Bloom filters [6] from individual link identifiers in each forwarding node along the path. Based on this constant size identifier, the Forwarding element efficiently forwards each packet through a simple AND/CMP operation on the Bloom filter identifier, resulting in efficient multicast support. Flow-dependent state is not required in the Forwarding element because LIPSIN moves the state into the header of each packet; only some link information, which is assigned to each forwarding node during bootstrapping, is required for the forwarding operation.

Figure 2 depicts a simple example of the intra-domain dissemination strategy. We assume that subscribers *S2*, *S3* and *S4* have already subscribed to an information item that is later advertised by publisher *1* (message *1*). The advertisement is forwarded to one of the potentially many rendezvous (RV) nodes running in the domain. As shown in the Figure, a rendezvous node manages the information graph that is accessible by the nodes residing in the same domain. All nodes in the domain are assigned with at least one LIPSIN identifier that is used to forward pub/sub requests to a RV node during each node's bootstrapping. The RV node, then, matches the availability of information with the interest for it and publishes a topology formation request to a Topology Manager (TM) (message *2*). Note that multiple TMs may control the topology for a single domain running a link-state protocol for having a centralized view of the domain's topology. The TM creates a LIPSIN identifier from *P1* to *S2*, *S*3 and *S4* (the last two running in the same host) and publishes it to *P1* (message *3*). *P1*'s network stack maps the received forwarding identifier to the advertised information item and notifies the application about the existence of subscribers. Finally, it is up to the application to publish data with this information identifier (message *4*). For instance, in a live TV scenario, *P1* can constantly publish video chunks until no subscribers exist for the information identifier (at this point a similar message sequence causes *P1*'s network stack to notify the application that no more subscribers exist).

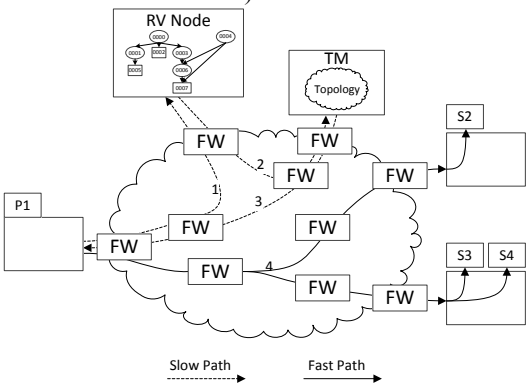

Fig. 2. Intra-domain information dissemination

In order to address scalability issues because of false positives when using LIPSIN identifiers, the intra-domain forwarding can be implemented using multi-stage Bloom filters [16]. The topology manager utilizes its domain topology information to concatenate Bloom filters, each of which encodes only the membership of the edges residing at a given hop-distance (stages) from the source.

Implicit rendezvous strategy. In this strategy, information is disseminated across a network domain but no explicit rendezvous takes place. Instead, subscribers declare their interest for information to their local network stack. Publishers publish data by also providing a LIPSIN identifier to the destination node(s) or by linking an information item to another one for which rendezvous has already taken place and a forwarding identifier is available. This way, information is directly disseminated to the network bypassing the slow-path operations (i.e. rendezvous and topology management and formation). Disseminating information without explicitly requiring rendezvous is very important in many aspects of our

network's operation. For example, all control messages presented in Figure 2 (i.e. messages *1*, *2* and *3*), are actually published as regular information, using special information identifiers assigned to network entities during bootstrapping. Nevertheless there is no explicit rendezvous that takes place for any of these items. Instead, the *implicit rendezvous strategy* is used to publish this data. For example, the initial advertisement is published using a pre-configured LIPSIN identifier that points to a RV node using an also pre-configured information identifier. Moreover, the topology manager calculates the LIPSIN identifier, which is used to publish the respective notification to *P1* (message *3*).

Another very important use case for the *implicit rendezvous strategy* is when utilizing algorithmic identifiers (see Section III.2)). As an example, we describe a simple fragmentation scenario where some static content, advertised as */A/B* needs to be published to a number of subscribers. Rendezvous takes place once for */A/B* and after the slow-path operation is over, the publisher's network stack holds a LIPSIN identifier for */A/B*. The publisher application can then publish fragments using algorithmic identifiers */A/algID$_{/A/B}$/algID$_x$* and request from the network stack to utilize the LIPSIN assigned to */A/B*. *algID$_x$* is an identifier produced for fragment *x*, while *algID$_{/A/B}$* is a scope identifier produced from the identifier */A/B*. Respectively, subscribers subscribe to the scope */A/algID$_{/A/B}$* locally to their network stack. This way a publisher can send more than one MTU-sized publications to one or more subscribers without rendezvous being required for every published item, minimizing the slow-path overhead. Note that all these items have different identifiers and, thus, can be cached in the network. The only requirement here is an agreement between the communicating entities about the utilized algorithm that produces the respective identifiers.

Inter-domain strategy. This strategy is utilized when information must be disseminated across multiple network domains. We are currently working on extending the multi-stage Bloom filter-based forwarding [16] towards an inter-domain, policy-friendly forwarding solution. Each domain is represented as a single stage, with each domain internally forwarding information according to its internal policies, which are not visible outside the domain nor represented in the inter-domain Bloom filter. Topology managers running in different domains cooperate to produce multi-stage Bloom filters by disseminate BGP-like inter-domain information to neighbouring domains. Finally, a distributed rendezvous solution, like the one presented in [13], can be used to match information demand and supply. The integration of Blackadder with the NS3 simulator makes it possible to evaluate such a strategy in a simulated multi-domain network. Such evaluation is left out of this paper.

IV. Node Implementation

Blackadder preserves the functional modularity of the underlying network architecture. This allows for separately optimizing functions throughout the lifetime of the prototype. Separate optimization is not only likely due to the specificity of each function but also due to the potential different 'ownership' of each function in certain deployment situations. Hence, the functions are separately implemented along clearly defined modular boundaries laid out by the underlying architectural core functions. This aids the extensibility and support for parallel realizations of a network core function (e.g. for different dissemination strategies). One of the major objectives of our implementation efforts is to provide a development framework for implementing and experimenting with ICN functionality. Thus, platform and deployment flexibility is crucial so that with no changes in the implemented functionality, one can experiment with high-performance and wide-area network deployments, as well as with simulated or emulated environments.

The implementation of Blackadder is based on the Click modular router [7] platform. The choice of Click as the framework to implement our node is ideal for meeting the objectives described above. Click allows for building and connecting modules providing a perfect framework for cleanly separating functionality within the network node. Each module in Blackadder (implemented as a *Click Element*) provides a clean interface to other elements. Its communication elements support a variety of transport mediums, which is ideal for experimenting in parallel to IP deployments as well as by overlaying on top of IP. The notion of Click elements enables the development of our main functions in a way that eases portability between kernel and user space as well as across operating systems. Currently, we run our prototype in Linux, FreeBSD, Mac OS X, Android, OpenWrt and integrated into NS3. A user space deployment supports quick prototyping of functionality as well as experimenting in environments where throughput is not the primary metric. A kernel space deployment is more efficient in terms of performance. The integration with NS3 allows for moving between real deployments and emulations or even simulations of the same functionality with virtually no programming overhead.

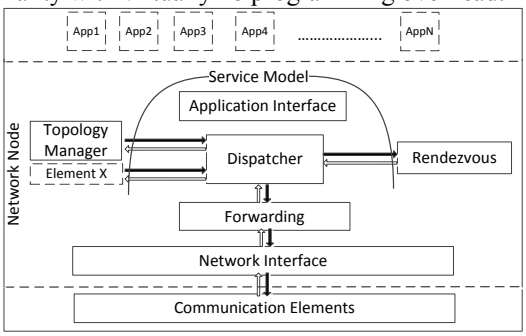

Fig. 3. Blackadder Node Overview

As shown in Figure 3, the core network functions are implemented as separate Click Elements. Blackadder exports a publish/subscribe API, described in [2], to applications as well as the topology manager and rendezvous elements. Other auxiliary, in-Click elements access the exported functionality using the same service model, as depicted in Figure 3.

A. Application Interface Element

The *Application Interface Element* interfaces our network stack to applications. Usually, applications interact with the

networking software of an operating system via system calls. In modern Linux kernels, adding system calls requires kernel recompilation, hindering quick experimentation. For that reason, we choose Netlink sockets to interact with applications. Applications can open a Netlink socket and then interact with the network stack using existing system calls. Additionally, using Netlink sockets, applications access the same API regardless of the mode in which Blackadder runs (i.e. user or kernel space). In kernel space, Netlink sockets receive socket buffers from applications that are, then, wrapped into Click packets with no extra memory cost.

B. Dispatcher Element

The *Dispatcher Element* is at the heart of Blackadder. It receives publish/subscribe requests from applications and other Click elements as well as from the network via the Forwarding element and it provides a proxy function to all publishers and subscribers. Rendezvous nodes (even the one running in the same node) are not aware of individual applications. Instead, a statistically unique label, which identifies a network node, is (self-) assigned by the Dispatcher.

Whenever the Dispatcher element receives an '*advertise'* or *'subscribe'* request from an application or a Click element that can access the exported service model (as shown in Figure 3), it creates a publication with the initial request as the payload. Then, according to the dissemination strategy, it publishes the data to a rendezvous node (e.g. locally to the rendezvous element for the intra-node strategy or to one of the domain's rendezvous nodes for the intra-domain one). As mentioned in Section III.3), it publishes the request using the implicit rendezvous strategy, a pre-configured LIPSIN identifier that points to a rendezvous node and a well-known information identifier to which rendezvous nodes are locally subscribed. Thus, all these publications finally reach the rendezvous element of the node that is a rendezvous node for this publish/subscribe request. This publication, just like all publications, is dispatched to the subscribed entity, i.e. the rendezvous element running at that node.

The Dispatcher *publishes data* for an information item if requested by an application or a Click element in one of the following cases: if rendezvous has previously taken place for this information item or if the item is published using the implicit rendezvous strategy. In the first case, there is already a LIPSIN identifier mapped to the information item. In the second case, a user-provided LIPSIN identifier or one assigned for another information item, for which rendezvous has previously taken place, will be used.

Finally, network publications that are pushed by the forwarding element are simply dispatched to applications or Click elements that previously subscribed to the identifier of the publication or its father scope. A special case is when publish/subscribe notifications are published by the topology management core function (message *4* in Figure 2). These notifications are published using a pre-configured information identifier. In their payload they contain the identifier of the information to which the notification refers. Only the dispatcher itself is interested in these notifications. Upon receiving such notifications, it dispatches a notification to start or stop publishing data for the item referred in the payload to any applications that have previously advertised the item.

C. Network Core Functions' Elements

The *Rendezvous*, *Topology Manager* and *Forwarding Elements* implement the core functions of our ICN. Their functionality depends on the dissemination strategy of the information and was described in Section III.3). The *Forwarding Element* currently implements the LIPSIN mechanism [6] for intra-domain information dissemination. For this, it maintains a forwarding table that maps link identifiers (LIDs) to Click ports that point to a Click element that can access the network. Another LID is used to "connect" the Forwarding with the Dispatcher element. If such a LID is included in a LIPSIN identifier, the forwarding element will push the packet to its Dispatcher element. This way a network node is instructed to process a network publication rather than merely forwarding it. The *Rendezvous Element* implements an in-memory index where the information graph along with the set of publishers and subscribers is stored for the intra-node and intra-domain dissemination strategies. It subscribes locally to a pre-configured scope under which all publish/subscribe requests are published. The *Topology Manager* utilizes the service model to subscribe to a scope to receive topology formation requests by rendezvous nodes and publishes the response to one or more publishers of an information item.

D. Communication elements

We utilize Click elements for communicating with other network nodes, supporting Ethernet communication as well as communication over raw IP sockets. The former can be used when experimenting in a LAN or VPN, while the latter is appropriate when overlaying on top of IP. Blackadder nodes may have multiple instantiations of the aforementioned elements even in a mixed mode where a node may bridge two or more LANs over an IP network, with individual LANs running the network stack over Ethernet, enabling complex deployments where transit links running over the Internet connect network domains that natively support our ICN.

V. Experimental Evaluation

A. Testbed Deployments

Blackadder allows for transparently supporting different operating systems as well as native, overlaid or mixed deployments in high-speed or wide-area networks. We have deployed Blackadder in three different testbeds: a high performance Gigabit Ethernet testbed consisting of 15 identical hosts [1], where network nodes run in the Linux kernel natively on top of the network, in a PlanetLab slice consisting of 106 slivers, where nodes run in user-space on top of IP, and in an international testbed that interconnects 10 major universities and institutions worldwide (~40 machines in total). All sites are connected via OpenVPN, which exports a virtual Ethernet device to all machines in the testbed. We are working on

[1] More information about the testbed can be found at http://nitlab.inf.uth.gr/

creating a multi-domain testbed where islands of native ICN networks each one running its own rendezvous and topology management nodes will be interconnected via the Internet.

B. Evaluation

An ICN network node needs to process and forward information at line-speed without requiring much state to do so. Moreover, slow-path functions must scale as the number of nodes and content increases; i.e. if the required state is easily shareable among nodes undertaking the same network function and if the processing overhead for doing so is low. The focus of the evaluation is to assess the performance of various aspects in our ICN as this is realized by Blackadder. We measure the rate at which Blackadder processes and dispatches publications to interested applications, we show the efficiency of the forwarding mechanism in an intra-domain deployment and we provide indications about the scalability of the slow-path functionality in a much larger deployment. Finally, we measure the performance of a basic socket emulation implementation on top of Blackadder that could support legacy applications requiring end-host based communication. As mentioned in Section III, ns-3 emulation based evaluations, e.g. for the inter-domain strategy, are left out of this paper for space reasons.

1) Intra-node performance

We test the implementation of our intra-node dissemination strategy when dealing with a heavy load of publications, emulating an IPC-like communication scenario. We use a single publisher that advertises an information item under a root scope. We measure the application throughput for a set of subscribers, ranging from 1 to 10, subscribing to the advertised information. After rendezvous takes place (this happens only once, minimizing the overhead posed by the slow path functions), the publisher publishes 100,000 items using the same information identifier. We repeat the experiment for different payload sizes. The upper limit of the payload size is bound by the Netlink socket buffer size (~100KBs).

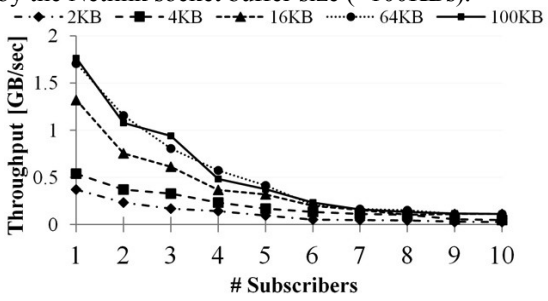

Fig. 4. Intra-node performance

In Figure 4, we observe that for large payloads and few subscribers the average throughput is more than 1 GB/sec. The performance degrades when more subscribers exist due to multiple publication copies. For 10 subscribers, the throughput is between 40-100 MB/sec. We point out that for the current node-local dissemination strategy, the Dispatcher element creates the necessary number of copies for each publication and then dispatches them to interested applications. This local memory management could be replaced with different strategies (e.g. a blackboard) that eliminate unnecessary copies. Overall the measured performance is more than acceptable since it is close to the TCP/IP based IPC which has been heavily optimized over the past decades.

2) Fast path performance

We now extend towards an intra-domain strategy with an ICN topology of 15 nodes connected in a chain, in order to measure the efficiency of the forwarding function. All published items have an MTU size of 1500 bytes, including the Ethernet header. The first node in the chain runs the RV node and TM. The second node is the publisher that behaves as in the previous experiments. The rest of the nodes run 1, 3, and 6 subscribers (depicted as (1), (3) and (6), respectively).

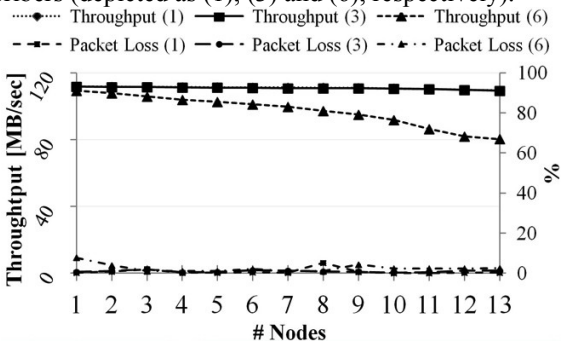

Fig. 5. Forwarding Efficiency

In Figure 5, we observe that when a single subscriber runs in each node, all subscribers receive data at line speed even when 13 subscribers exist. In this case, each forwarding node forwards all publications to its next hop and pushes the data to the local subscriber. Only for 6 subscribers per node, the performance degrades for a chain larger than 3 nodes. For all cases, the packet loss is less than 5%. Note that the only direct comparison to a TCP/IP-based performance is that of 1 subscriber in one node since all other transfers exploit the multicast support of the LIPSIN mechanism, while a TCP/IP solution would result in halving the effective throughput for each point-to-point transfer.

The presented performance is exceptional. Blackadder can forward information at Gigabit speed with minimal packet losses while the required state in each forwarding node is minimal; i.e. only a 256 bit forwarding identifier, which is known by the topology managers, per provided network link. The observed performance degradation when running 6 subscribers per node is natural and expected since 6 copies of each publication are created and dispatched as the original publication is forwarded to the next hop. Only a blackboard-based local dissemination strategy could minimize this overhead, as mentioned in the previous sub-section.

3) Slow path performance

We now turn our attention to the sequence of rendezvous as well as topology management and formation that needs to take place before executing the fast path forwarding. In the previous experiments, these functions' overhead was minimized because a single information item was used to publish data, effectively creating an information channel.

We first focus on the rendezvous process by utilizing the intra-node strategy, which requires no explicit topology

formation. A single publisher creates a scope and then advertises 100,000 information items under that scope. Then, subscribers are synchronized to start subscribing to items in the advertised range. Each subscriber iterates 500 times and each time it subscribes to an item using a randomly generated identifier, waits until it has received the data and then subscribes to another one. For each subscription, rendezvous takes place and the publisher is notified to publish the payload (the information here is a minimum Ethernet frame in order to reduce the forwarding overhead) that corresponds to this advertisement. We then measure the time between each subscription and the receipt of the data at the subscriber. We call this the *response time*. We underline that the granularity of resolved information items will usually be on a per-content (using algorithmic identifiers) or per-channel basis (like in the previous sub-section), therefore the presented experiments constitute the absolute worst case where rendezvous and topology formation must take place each time a few bytes must be transferred. Figure 6 shows the response time when up to 500 subscribers subscribe simultaneously to information items in the advertised range. In the node-local case, the average response time for 200 subscribers is 20 ms, increasing to 54 ms for 500 subscribers.

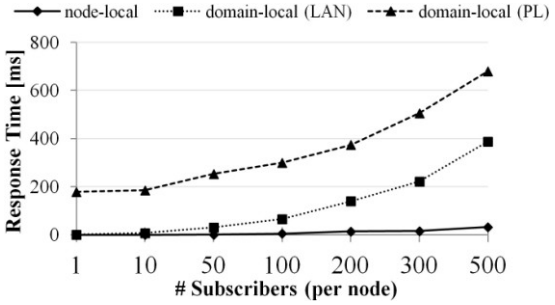

Fig. 6. Slow Path Performance: Node-Local, Domain-Local in LAN, Domain-Local in PlanetLab

To add the topology formation overhead in the response time, we extend the experiment using the intra-domain strategy in a star topology of 15 nodes in our Gigabit LAN testbed. The star topology provides a constant forwarding delay for all subscribers since the rendezvous node and TM run in the central node and the publisher runs in one of the satellite nodes. In this strategy, the entire slow path, from rendezvous to the topology management and formation, is involved. In Figure 6, we observe that the response time increases with the number of subscribers (up to 500 per node). Compared to the intra-node experiment, the response time is higher due to the network delays. Another factor is that each of the 14 nodes runs the number of subscribers depicted in the x-axis, resulting in a response time of 388 ms for a total of 7000 subscribers.

As an example of an overlay deployment, we present the slow-path performance results in a PlanetLab deployment. For this, we create a slice that consists of 106 slivers forming a (randomly generated) graph with 73 edge nodes. The rendezvous node and TM run in a dedicated machine together with the publisher. The results for this experiment are also included in Figure 6. For 200 subscribers per node (totalling 14,600 subscribers), the average response time is 373 ms, which increases to 680 ms for 500 subscribers per node (i.e. 36,500 subscribers). Note that 500 subscribers running in the same node is not a realistic application example. Nevertheless, this is the only way to scale up our evaluation to realistic conditions in terms of processing requirements.

The presented results are very promising. From the intra-node scenario we see that the processing overhead for performing rendezvous scales well as the number of publish/subscribe requests increases. For the intra-domain case where both rendezvous and topology formation take place we see that that a single RV and TM running in the same machine can cope with thousands of simultaneous requests. Although the response times, which include propagation delays, increase, we argue that the required state for performing these functions can be easily shared among multiple machines in a load-balancing or cooperative mode, achieving scalability within a single domain. For instance, multiple TMs implementing a link-state protocol can share the network load for topology formation. Respectively, distributed rendezvous solutions, such as in [13], can efficiently cope with the required load. Finally, given that the presented communication scenarios are extreme cases where measurements were taken while rendezvous and topology formation take place for each packet, we expect that slow-path functionality will be required on a much coarser granularity (per-content or per-(large) chunk of content).

4) Legacy Application Support

An open question for all ICN architectures is whether they can support legacy applications as well as what is the price to pay for supporting such applications in terms of performance. Here we evaluate a simple application that emulates a bi-directional, unreliable stream of packets, like the one provided by the UDP protocol. For doing so, the server subscribes to a well-known scope (e.g. */X*). A client then advertises an information item with a statistically unique label under the well-known scope (e.g. */X/Y*). As a result rendezvous takes place and the topology formation function creates a LIPSIN identifier that is published to the client, which, in turn, algorithmically calculates a new information label (e.g. */X/Z*), issues a subscription to the respective information item and publishes the label (as data) to the server, which advertises this item to the rendezvous node. As a result, RV takes place and, finally, the server acquires a forwarding identifier to the client. On top of these unidirectional channels one can emulate the basic socket operations; a *sendmsg* call from the client would result in publishing data to for */X/Y*, while a *recvmsg* call would wait for data for the item */X/Z*. The opposite identifiers would be used by the server's socket operations.

For this experiment we implemented a simple file transfer protocol and we measure the application throughput when multiple subscribers (clients) receive different files from the publisher (server). Although the ICN network supports multicasting using LIPSIN identifiers [6], in this example we intentionally build bi-directional pipes to support legacy functionality. In Figure 7 we show the total and average application throughput that was measured when running a simple file transfer protocol on top of Blackadder and on top of UDP, for an increasing number of clients (shown in the *x* axis)

running in different machines. We see that the total application throughput for both scenarios is bounded by the network's available bandwidth. The average application throughput decreases as the number of clients running in different machines increases because of the unicast nature of the communication.

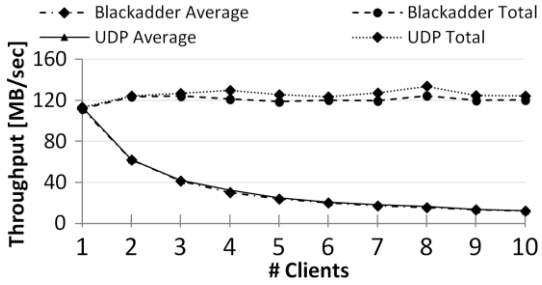

Fig. 7. Legacy Application Throughput

C. Comparison With CCNx

CCN separates its main functions into routing (building forwarding information tables in each forwarding element), forwarding (sending interest and data packets to producers and consumers) and caching (storing interest and data packets for improving delivery). Routing is currently neither fully specified nor implemented. The only effort towards populating FIB entries is OSPFN [17], an adaptation of OSPF using names instead of connectivity information. However this mechanism is not integrated with CCNx. Forwarding and caching consult the local FIB and PIT to forward data as well as interest packets for a given name, while the content store holds any previously transferred packets.

Contrary to our approach, CCN supports only immutable data; i.e. any new content carries a new name. While this is reasonable for content delivery, it poses a burden on conversational or sensor applications. Finally, CCN requires the producer to sign content on a per-content or per-packet basis, whereas data verification by forwarding nodes and consumers is optional. This ties the feasibility of the CCN node design to cryptographic advances. While mandatory signing is reasonable for videos or news, it can pose a significant overhead on end systems in sensor networks or mobile device based content scenarios. Packet signing is not always necessary when implementing simple conversational services on top of CCN (e.g. for emulating a socket interface as the one presented in the previous section). In Blackadder a similar approach [18] is only optional and can be implemented on top of the node as a separate layer. To make the comparison fairer, in our experiments we try to avoid the signing overhead whenever that is possible. Since CCNx only implements forwarding and caching (with FIB entries being manually configured), we focus our comparison on the information processing overhead and fast path performance. All applications are written using the C library provided in the latest CCNx distribution (v0.7.0). Overall, our results show that the overhead posed by the interest processing in CCN as well as the support for mutable semantics and the flexibility of using stateless forwarding mechanisms in Blackadder result in a performance advantage.

To evaluate the information processing in an IPC scenario, we emulate an IPC-like pipeline between two applications. For CCNx we implement an intra-node CCNx application that expresses 10,000 interests in content, each piece being individually labelled with its own name (e.g. */content/segment_no*).

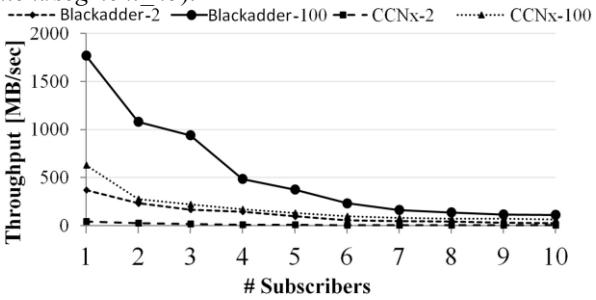

Fig. 8. Intra-Node Performance Comparison

We place the individual content pieces into the local CCNx cache to avoid the signing overhead when played out by the application. For our prototype, we run the same application as for the experiment in Section V.B.1). Figure 8 shows the results for data sizes of 2 and 100 KBs. Even though the CCNx prototype plays out data from the content store, the performance is significantly lower compared to Blackadder.

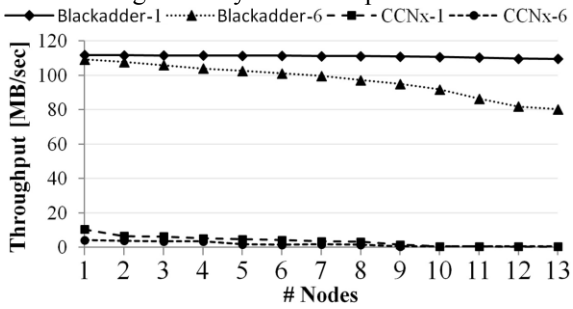

Fig. 9. Throughput Comparison: Mutable Semantics

For our second experiment, we create a chain of 14 nodes with the first one being the producer and the rest being the consumers. At each node we run up to 6 such consumers with each node being connected to our 1Gbps testbed. We implement a window-based mechanism for sending interests to the network stack. Also here, we experimented with a cached and non-cached case in order to single out the signing overhead. For our node design, we utilize the immutable as well as the mutable semantics; i.e. the first scenario incurs rendezvous overhead while the second is relevant when mutability is handled by the application. For a fairer comparison with the window mechanism used in the CCNx case, we use the same window-based mechanism in our immutable case. Figure 9 shows our measurements when comparing Blackadder in mutable mode with CCNx, each with 1 and 6 subscribers. The graph omits the non-cached case since the performance never exceeded 170kB/s. At one subscriber per node in the chain, Blackadder sustains line speed throughout the entire chain, while degrading down to 80MB/s with 6 subscribers. We can confirm the CCNx performance with one consumer being directly connected (CCNx-1 point for

1 node) with the results in [5] for a 100Mbit/s test bed. The throughput reported is similar to the one we observe in our experiments, namely ~10.4MB/s. However, the CCNx-1 performance continues to decrease along the chain of nodes, down to ~0.4MB/s for 13 nodes, mainly because a separate interest must be sent and processed by all network nodes in the path for every packet. This problem has been also identified in [19], where persistent interests are proposed. Overall, we see that our domain-local forwarding mechanism outperforms the PIB/FIT-based CCNx mechanism.

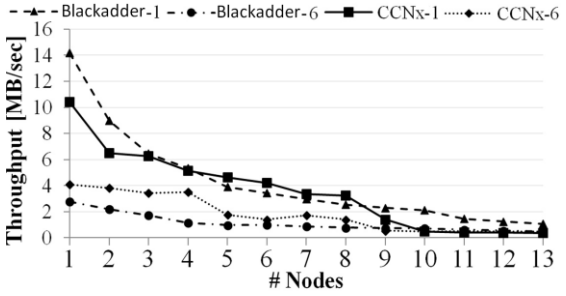

Fig. 10. Throughput Comparison: Immutable Semantics

Figure 10 compares the CCNx performance (in cached mode) to our prototype with immutable semantics. Although we see similar performance, for Blackadder the slow path functions are invoked for every single packet, while we expect rendezvous to take place on a per-content or per-chunk of data in realistic application scenarios.

We do not include a performance comparison for a conversational application, as the one presented in Section V.B.4) for space reasons. However, the expected bottleneck for CCNx is obvious from our previous experiments. Implementing such an application on top of CCNx would require signing all packets the moment they are created. In many cases (e.g. voice or telent/ssh over CCNx) it is not possible to pre-sign packets or content as a whole (e.g. by using Merkle hash trees). As a result the throughput would be limited by the signing rate, which, as mentioned above, was measured about 170KB/sec in our setup.

VI. Conclusion

Increasing interest in ICN creates the need for a flexible and extensible development platform to allow research in the area to progress. We addressed this need, and made the following contributions in this paper. We presented Blackadder, an open-source ICN node implementation that allows for continuous experimentation through its modular design than can easily accommodate future developments. It runs in parallel with TCP/IP or on top of it. It is also integrated with the NS3 simulator, easing the evaluation of our ICN in larger, simulated network topologies. We described how information is disseminated in various communication scenarios and how our core network functions are implemented within Blackadder. We also showed that this flexibility does not come at the price of performance. Our experimental evaluation showed very promising results for all three core network functions. Finally, we presented a performance comparison with CCNx and showed that our approach significantly outperforms CCNx.

Acknowledgements

The work reported in this paper was supported by the FP7 ICT project "Publish Subscribe Internet Technology" (PURSUIT), under contract ICT-2010-257217. We thank George Xylomenos for his helpful feedback to this article.

References

[1] D. Trossen, M. Sarela, K. Sollins, "Arguments for an Information-Centric Internetworking Architecture", ACM Computer Communication Review, April 2010.

[2] D. Trossen, G. Parisis, "Designing and realizing an information-centric internet", IEEE Communications Magazine 50(7): 60-67 (2012).

[3] G. Xylomenos, et al., Caching and mobility support in a publish-subscribe internet architecture. IEEE Communications Magazine 50(7): 52-58 (2012).

[4] J. Scott, J. Crowcroft, P. Hui and C. Diot, "Haggle: a networking architecture designed around mobile users", Proc. of IFIP WONS 2006.

[5] V. Jacobson, et al., "Networking named content", Communications of the ACM, Vol. 55, No. 1, 2012.

[6] P. Jokela, A. Zahemszky, C. E. Rothenberg, S. Arianfar, P. Nikander, "LIPSIN: line speed publish/subscribe inter-networking", In Proc of. SIGCOMM 2009.

[7] E. Kohler, R. Morris, B. Chen, J. Jannotti, F. Kaashoek, "The Click modular router", ACM Trans. Comput. Syst. 18, 3, 263-297.

[8] T. Koponen, M. Chawla, B. Chun, A. Ermolinskiy, K. Kim, S. Shenker, Ion Stoica, "A data-oriented (and beyond) network architecture", in Proc. of SIGCOMM 2007.

[9] L. Popa, A. Ghodsi, I. Stoica, "HTTP as the narrow waist of the future Internet", in Proc. of Hotnets 2010.

[10] T. Biermann, C. Dannewitz, H. Karl, "FIT: Future Internet Toolbox", in Proc. of TRIDENTCOM, 2010.

[11] M. Caesar et al., "ROFL: routing on flat labels", in Proc of SIGCOMM 2006.

[12] "Blackadder Node Implementation", https://github.com/fp7-pursuit/blackadder, accessed December 9. 2012.

[13] K. Katsaros et al., "On inter-domain name resolution for information-centric networks", Proc. of IFIP Networking 2012.

[14] A. Carzaniga, D. S. Rosenblum, A. L. Wolf, "Design and evaluation of a wide-area event notification service," ACM Transactions on Computer Systems, vol. 19, no. 3, 2001.

[15] A. Rowstron, A. Kermarrec, M. Castro, P. Druschel, "SCRIBE: The Design of a Large-Scale Event Notification Infrastructure", Proc. Of NGC, 2001.

[16] J. Tapolcai, et al., "Stateless Multi-Stage Dissemination of Information: Source Routing Revisited", in Proc. of Globecom 2012.

[17] "NDN-Routing/OSPFN2.0",https://github.com/NDN-Routing/OSPFN2.0, accessed January 11. 2012.

[18] Dmitrij Lagutin, "Securing the Internet with digital signatures," Doctoral dissertation, Aalto University, Finland, Dec 2010.

[19] C. Tsilopoulos, G. Xylomenos, "Supporting Diverse Traffic Types in Information Centric Networks", in Proc. of SIGCOMM workshop on ICN, 2011.