Content-Based Publish/Subscribe Networking and Information-Centric Networking

Antonio Carzaniga
University of Lugano
Lugano, Switzerland
antonio.carzaniga@usi.ch

Michele Papalini
University of Lugano
Lugano, Switzerland
michele.papalini@usi.ch

Alexander L. Wolf
Imperial College London
London, United Kingdom
a.wolf@imperial.ac.uk

ABSTRACT

On-line information comes in different forms and is accessed in different ways and for different purposes. For example, a recording of Beethoven's Ninth Symphony differs from a storm warning from the local weather service. Beethoven's Ninth is a large media file with perpetual validity that is typically accessed on demand by users. By contrast, a storm warning is a small ephemeral message typically pushed by the weather service to all users in a specific geographic area. We argue that both should and would be well supported by an information-centric network. More specifically we argue three points. First, modern applications, reflecting the nature of human communications, use and transmit large and long-lived files as well as small ephemeral messages. Second, accessing those two types of information involves significantly different operations within the network. Third, despite their differences, both types of information would benefit from an addressing scheme based on content rather than on more or less flat identifiers, which means that both should be integrated to some extent within a unified content-based routing infrastructure.

Categories and Subject Descriptors

C.2.6 [**Computer Systems Organization**]: Computer-Communication Networks—*Internetworking*

General Terms

Design

Keywords

Content-based networking, publish/subscribe, information-centric networking, content-centric networking, named-data networking

1. INTRODUCTION

The notion of content-centric networking is based on an addressing scheme wherein the send and receive communication primitives identify content rather than network locations. This addressing scheme is motivated by social, application-level considerations, as much as by technical, network-level considerations. At a high-level, communication would be more effective if consumers could simply specify *what* content they intend to receive as opposed to from *where* that content might be retrieved. At the network-level, an addressing scheme that identifies content as opposed to location would allow the network to operate more efficiently by duplicating and caching content around the network, since it is the delivery of content that matters, not where that content resides. Simply put, content-centric networking can be seen as a fresh and principled approach to the problem of content distribution, especially for the common usage pattern in which users request named content.

Delivering named content upon demand is only one important form of content-based communication. Another, equally important form is *publish/subscribe event notification.* We argue that both forms should be natively supported features of a truly *information-centric* network [1]. Going further, we give a first design for a network service supporting both forms of communication.

Publish/subscribe event notification is a form of content-based communication in the sense that the content of a message (i.e., the information concerning an event) determines where the message is delivered [2]: senders simply "publish" messages, while receivers "subscribe" for their content. That is, senders send messages without any explicit destination address, but with some structured content (possibly the entire message) visible to the network, while receivers declare a *predicate* (a kind of query) that, when applied to the structured content of a message, tells whether the message matches the receiver's interests. The network then transmits each message to any and all receivers interested in its content, that is, to all receivers whose declared predicate is matched by the content of the message. This notion of content-based communication and a corresponding network architecture [3, 5, 6] was developed independently from the notion of content-centric (or named-data) networking [8].

Apart from their separate histories, these two forms of communication differ fundamentally along three dimensions: (1) the validity (or value) over time of the content; (2) the initiator of the content flow; and (3) the expressiveness of names versus predicates. In this paper, we consider only the first two dimensions.

In terms of validity, information can be *persistent* or *transient*, where persistent means valuable for a long period of time, while transient means short-lived, or perhaps long-lived but most valuable only for a short period of time. In terms of initiator, a flow can be *consumer initiated* or *producer initiated*. In the first case, a consumer (or receiver or sink) requests some information that might be available from multiple producers (or senders or sources); in the second case, a producer "pushes" some information that might be delivered to multiple consumers.

The primary focus of research in content-centric networking has been the delivery of named content upon demand, which is characterized by persistent information and consumer-initiated flows. An example would be a request for an MPEG-formatted file containing a recording of Beethoven's Ninth Symphony. By contrast, publish/subscribe event notification is characterized by transient information and producer-initiated flows. Common examples are news announcements or weather-service alarms. In practice, of course, validity/value and flow initiator are not completely independent variables: persistent information may be pushed by producers, but is more typically requested by consumers, while transient information is typically pushed by producers, simply because consumers may not become aware of its availability within the limited period of validity.

In this paper we examine three questions concerning publish/subscribe event notification and its relationship to on-demand content delivery. First, is publish/subscribe event notification a *heavily used* form of communication? Second, if it is indeed heavily used, should it be implemented on top of an on-demand content delivery service or provided as a native feature of a more general information-centric network service? In other words, is publish/subscribe event notification *different enough* from on-demand content delivery that its implementation should be based on specialized network-level primitives? If it is the case that an information-centric network should directly support publish/subscribe event notification, then the third question is how should this implementation interact with the implementation of the on-demand content delivery service? In other words, *can the two services be synergistic* within an information-centric network? In the following sections we provide answers to these questions, presenting a high-level architecture for a common content-based layer within an information-centric network.

2. THE CASE FOR PUBLISH/SUBSCRIBE EVENT NOTIFICATION

Looking at the landscape of modern communication modalities, publish/subscribe event notification appears to have established itself as one of the principal building blocks of computer-mediated communication. Evidence for this claim can be found in the numerous applications in existence today that make extensive use of transient, producer-initiated messaging. Typical examples are news and information-sharing systems, such as those based on RSS feeds (e.g., PubSubHubbub[1]) and aggregators (e.g., FeedBurner[2]), Twitter, and their integration.[3] Various other types of messaging and notification services are used for presence, real-time communication, and workflow management [10, 11]. Examples of more structured applications include financial data processing (e.g., the NYSE Data Fabric[4]), analysis tools based on complex event streams, network management, and application management. Publish/subscribe communication is also used with specialized network technologies, such as sensors networks [7], and with mass-market applications, such as those targeting smart phones (e.g., the Deacon Project[5]).

Of course, the volume of data traffic for publish/subscribe event notification cannot possibly approach that of on-demand content delivery, simply because of the sheer size of bulk content, nor is the issue of caching and cache management of particular relevance to event notification as opposed to content delivery. Nevertheless, the critical observation made here is that the high-volume publish/subscribe applications mentioned above represent a substantial, significant, and growing amount of control traffic (to establish and spread subscriptions), as well as message processing (to determine message content and subscription matches) that is arguably comparable in load to similar functions (content advertisement and query matching) performed in support of on-demand content delivery.

3. IS ON-DEMAND CONTENT DELIVERY THE ONLY PRIMITIVE?

We argue that a communication primitive designed specifically for on-demand content delivery—something that supports consumer-initiated requests for named data followed by replies that transmit the actual data—is not an appropriate basis for publish/subscribe event notification. Furthermore, we argue that the opposite—implementing on-demand content delivery on top of publish/subscribe event notification—is not a viable solution either. The two communication functions are different enough that it makes little sense to implement one with the other and, although conceptually feasible, it is not the best technical solution. In order to establish the basis for this technical argument, we refer to the content-centric networking (CCN) architecture introduced by Jacobson et al. [8] as a reference implementation of an on-demand content delivery service.

In CCN, data transfer is initiated by a consumer who sends out an "interest" packet to request a specific block of data. Interests are forwarded hop by hop towards one or more potential sources of the data using the Forwarding Information Base (FIB) available at each router. The FIBs are compiled on the basis of the announcements made by producer nodes in which a producer declares that it is a source for some named data. Such announcements are transmitted and processed through a more-or-less standard routing protocol (e.g., IS-IS).

Along their paths towards one or more producers, interest packets leave a trail of pending-interest entries stored in a Pending Interest Table (PIT) at each visited router. These "bread crumbs" serve two purposes. First, they cut loops in the process of forwarding interests, and second, they define the return path for the replies. As soon as an interest reaches a node capable of satisfying that interest—meaning that the node is either the original producer of the data or the node has a cached copy of the data from a previous reply—the

[1]http://code.google.com/p/pubsubhubbub/
[2]http://feedburner.google.com
[3]http://twitter.com/feedburner

[4]http://www.nyse.com/pdfs/Data-Fabric-product-sheet.pdf
[5]http://deacon.daverea.com/

node transmits the data in a reply packet that follows the trail left by the interest all the way back to the consumer. Also, the first reply consumes the corresponding pending interest, which means that further (duplicate) replies are dropped. Pending interests that are not consumed by a reply expire after some time.

One might try to support event notification using the same CCN architecture. Since transient event notifications are "pushed" by producers, one way to support them within the CCN infrastructure would be for the consumer to continually issue interests at more or less regular intervals, and for the producer to reply with a null data packet when no notification is immediately pending, or with a notification packet when a notification is issued (by an application on the producer node). This method, however, is problematic.

A first problem is that polling the producer by continuously issuing interests is likely to generate a lot of traffic (interest packets) and network state (pending interests) for only a few effective transmissions. Depending on the rate of publication and on the maximum latency accepted by the consumer, which would determine the frequency at which the consumer issues interests, the overhead might simply be too high.

A second and crucial problem is that the caching semantics of interests and data packets are fundamentally at odds with the semantics of transient event notifications. In fact, in order to get the latest notification each time one is produced, the consumer cannot issue the same request over and over again, since that would retrieve the same (most probably cached) data packet. Even assuming an interest that implicitly refers to the latest notification, there would have to be some mechanism by which the data packet satisfying that interest would have to expire at just the right time (probably immediately). Other solutions are possible, for example by having the consumer issue interests for the data representing an event notification with ever-increasing sequence numbers. However, this solution would also be problematic, since it would require some level of synchronization between producer and consumer, lest the consumer be stuck in the past, always asking for an obsolete notification, or stuck in the future, always asking for notifications that are yet to be produced.

An alternative to polling is to use CCN primitives so that the transmission is initiated by the producer. This can be done by having the producer send an interest packet that is not intended to return any data, but that carries what amounts to a call-back prefix, effectively triggering a polling interaction by one or more interested consumers. This is a general protocol that can be used to implement producer-initiated exchanges in CCN, such as posting an e-mail message or sending a document to a printer.[6] Although the protocol is functional, it is not optimal in terms of delay and resource usage, especially for event notification. One problem is that the protocol requires a three-way exchange for what amounts to a one-way message. This may not be a significant problem, since short notifications could be carried entirely with a single interest packet, and also because the polling phase works well within the CCN architecture, even for multiple receivers.

The more serious problem is due to the overloaded use of interests as one-way notifications. The problem is that interests leave a trail of in-network state, so the most efficient way to use them is to forward them along a unicast path to the origin of the data or (better) to a closer node that caches the same data. However, when used as notifications, interests must be forwarded along all matching paths and all the way to the corresponding consumer applications. In the case of notifications with several interested receivers, this would create a large amount of network state for no useful purpose. Furthermore, without additional information, intermediate routers would not be able to distinguish interest packets used to transfer data from those used to trigger a transfer; to support both, they would have to forward every interest packet through all matching interfaces. The architecture we propose in this paper resolves this problem by distinguishing these two functions explicitly.

Another approach to supporting event notification on top of a CCN architecture is to augment CCN with some sort of "long-lived interests". Such a notion does not yet exist in CCN, but one can imagine implementing a form of standing interest that would be forwarded just like ordinary immediate interests but that, once they reach a producer, would stay active for some time waiting for the producer to send matching data. However, this solution would also pose a series of problems.

One obvious problem is that long-lived interests would only amplify the problem of maintaining in-network state by locking valuable PIT entries for a long period of time. Related to this, another problem is that the current CCN architecture was not designed to support partially overlapping interests—that is, interests sent by two or more consumers for prefixes that identify overlapping sets of names—so each interest would still be managed independently. However, while this design choice makes sense for immediate interests, the same design would be inefficient in the presence of long-lived interests. In practice, this means that two or more consumers with overlapping long-lived interests would use two or more entries in each PIT on the way to the corresponding producers. Also, each data packet matching multiple long-lived interests would be sent in multiple copies back to the consumers, with each copy being sent and forwarded independently by each router.

Yet another problem with this notion of long-lived interests is that they would still not be completely functional as an implementation of an event notification service. In fact, assuming that long-lived interests would be consumed by data packets like normal interests, there could be cases in which a number of events sent very quickly after the first event that consumes a standing interest would be lost before a new interest, supposedly reissued by the consumer, could eventually reach the producer.

Finally, notice that a long-lived interest is almost exactly equivalent to a subscription for publish/subscribe event notification, so a solution that relies on long-lived interests implemented within CCN would amount to implementing event notification as a native network service, which is essentially what we argue in this paper.

In summary, we conclude that the semantics of publish/subscribe event notification and on-demand content delivery are sufficiently different that each requires some level of specialized support in an underlying network fabric.

[6] http://www.named-data.net/faq.html

4. ONE CONTENT-BASED NETWORK

Although each form of communication requires some special support from the network, there exists a fundamental point of commonality that can be synergistically leveraged by both. We now sketch the architecture of a foundational content-based network layer. The main idea behind this shared layer stems from a correspondence between *interests* in on-demand content delivery (as defined, for example, in the CCN architecture [8]) and publish/subscribe *event notifications*.

On-demand content delivery (Figure 1) requires producers to register the prefixes of the names of content they intend to produce. The registered prefixes are then effectively used as addresses, or more appropriately address prefixes, with a standard routing protocol (e.g., IS-IS) to populate the FIB in each router. As with IP addresses, the routing protocol may combine individual prefixes into successively more generic prefixes. Correspondingly, in publish/subscribe event notification (Figure 2), consumers subscribe by declaring a predicate that expresses their interests, and those predicates are used as the addresses given to a routing protocol that populates the FIBs. As with IP addresses and name prefixes, predicates can be aggregated according to the topological (often hierarchical) structure of the network [2].

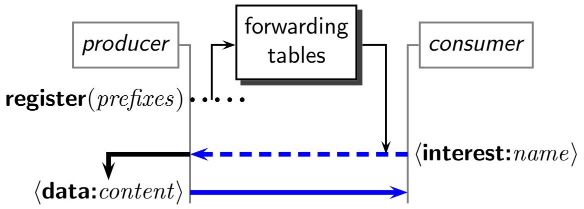


Figure 1: On-Demand Content Delivery

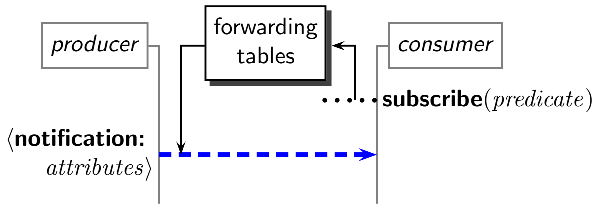


Figure 2: Publish/Subscribe Event Notification

Most publish/subscribe systems support expressive subscriptions that amount to SQL-like queries over semi-structured message content formed as a set of attributes of various data types (e.g., strings, integers, and dates). We see no conceptual obstacle in supporting this kind of useful and general content-based addressing for both on-demand content delivery and publish/subscribe event notification. However, to simplify the discussion in this short paper and more easily draw a comparison to existing naming schemes for on-demand content delivery, we consider only a limited form of publish/subscribe message content consisting of a single attribute—a character string *name*—and consider a limited form of predicate consisting of *name (i.e., string) prefixes* that have an identical semantics to the prefix-matching used in IP and CCN.

Given this starting point, the difference between on-demand content delivery and publish/subscribe event notification in terms of routing protocols and forwarding state is simply the *source* of the routing information, namely producers in on-demand content-delivery and consumers in publish/subscribe event notification, since both advertise "addresses" that are name prefixes. This suggests that the two communication primitives could use both a common routing protocol and a common FIB, as we describe below.

In terms of data traffic, the difference between the two primitives is a bit more involved. Both interests and event notifications are forwarded along paths toward matching prefixes. However, an interest is expected to generate a corresponding reply, while an event notification is a one-way message. Furthermore, the caching semantics are different. An interest that can be satisfied by cached content will not be forwarded downstream toward the original producer node, while an event notification must be forwarded all the way to interested consumers (although the event notifications might be cached for reliability purposes). This suggests that a common content-based layer must handle three separate types of messages: one-way messages, messages that expect a reply, and reply messages (Figure 3).

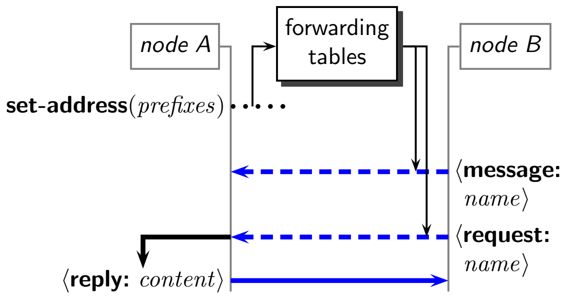


Figure 3: Unified Content-Based Network Layer

4.1 Node Interface and Packet Formats

The interface of a node in a content-based network implements the abstract service interface depicted in Figure 3. In order to avoid ambiguities between primitives in this layer and the corresponding ones in specific layers that implement on-demand content delivery or publish/subscribe event notification, we adopt a third and separate terminology. Each of the two forms of communication maps in a straightforward way onto the common content-based network layer. We discuss this mapping in Section 4.4 after first introducing the concepts underlying the common layer.

The node interface consists of a method to set its *content-based address* and methods to send *one-way messages* (or simply *messages*), *requests*, and *replies*. The content-based address of a node is defined by a set of prefixes, where a prefix might match a name contained in either a message or a request. A message behaves much like an IP multicast datagram, in the sense that it is intended to reach one or more destinations. A request, on the other hand, is intended to reach a data block or a cached copy of that block, and therefore results in a reply carrying the requested content or a segment of that content if found.

Figure 4 shows the high-level structure of each packet type used in the common content-based network layer. Address advertisements associate a predicate to a node, and are the primary source of routing information. In this paper we do not discuss routing in detail, and simply assume that address advertisements are transported and processed throughout the network by means of a standard link-state routing protocol.

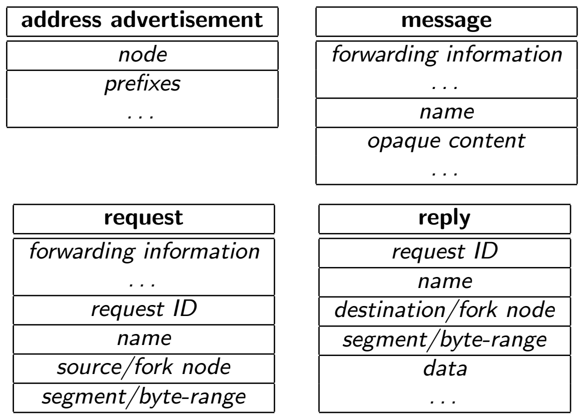


Figure 4: Packet Formats

4.2 Forwarding Messages and Requests

Both one-way messages and requests are forwarded toward nodes that advertise prefixes matching their names. Therefore, at a high level, forwarding is controlled by prefixes, and amounts to matching prefixes with names. In addition, forwarding might be controlled by other parameters, including policies of the sending node and/or of the forwarding node. For example, the sending node might want to limit fan out, since the communication is inherently multicast.

In general, the headers of messages and requests are meant to carry the information necessary to implement forwarding decisions that deliver messages and requests to nodes with matching prefixes while avoiding loops. Forwarding can be realized in a number of ways.

One strategy is to compare names against prefixes at each hop, typically with prefixes becoming progressively more specific as the message or request is forwarded downstream. Such a forwarding process can avoid loops, for example, by maintaining some soft state at each router, as in CCN [8], or by using a network-level broadcast layer [3]. In this case, the forwarding information could consist of a simple unique packet identifier used in short-term packet caches.

Another forwarding strategy is to use a form of source routing. Messages and requests can be evaluated against prefixes *once* as they enter the network. As a result, this evaluation produces a forwarding plan that is then attached to the message or request and consulted at each intermediate node. This general strategy is implemented, for example, in the B-DRP scheme, where the forwarding plan consists of a list of final destinations [4], and in the LIPSIN scheme, where the forwarding plan consists of a compact representation of the complete forwarding tree [9].

Rather than adopting a specific forwarding scheme, our primary intent in this paper is to propose an architecture in which both messages and requests can be forwarded using exactly the same scheme.

4.3 Handling Replies

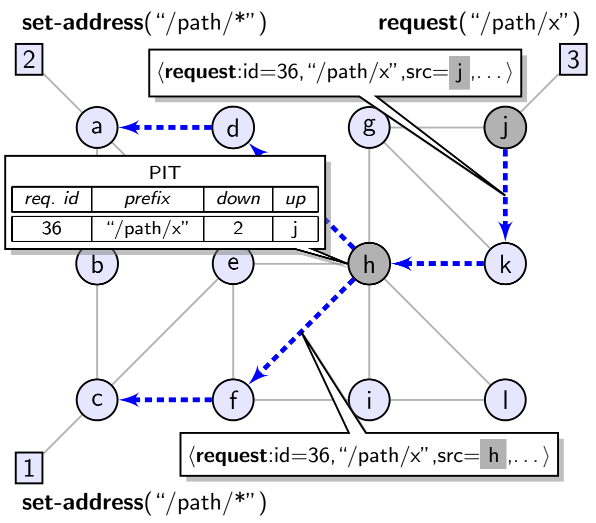
The difference between one-way messages and requests is that requests require additional processing and soft state to allow the corresponding replies to flow backward toward consumers. We now outline the process by which requests are handled at intermediate nodes.

Once again, we sketch an architecture and protocol based on CCN [8]. However, in this case we propose a different model in an attempt to reduce the space overhead incurred by CCN with its use of the pending-interest tables (PITs). CCN forwards interests (i.e., requests) by installing a PIT entry in each node through which the interest packet traverses. The advantage of this method is that the forwarding process can be simplified and effectively reduced to almost the same used in IP forwarding. This is because the forwarding can simply follow any number of matching prefixes without worrying about preventing loops, since those can be avoided when the same request is found in the PIT. The disadvantage of this method is the cost of storing (and looking up) every request in every router the request goes through.

We propose a heterogeneous forwarding scheme that combines content-based forwarding with traditional unicast IP forwarding. This scheme can cope with multiple requests and can also support negative replies (i.e., replies that carry no data, effectively stating that no such data exist). In this paper we only describe the basic behavior of a single request. This scheme is realized with four tables in each node: (1) an IP forwarding table that provides the traditional IP unicast network service; (2) a content-based forwarding table that associates prefixes to interfaces; (3) a pending-interest table conceptually similar to the PIT in CCN; and (4) a content store (i.e., a content cache) that is identical to the corresponding content store in CCN. Of these tables, the PIT is the only one that differs significantly from traditional forwarding tables or other tables in CCN, and that has a crucial role in the handling of requests and replies.

The PIT at node x associates a pending request (identified by the request id) with a number of downstream requests (forwarded by x) and a list of upstream requests. The main idea of this forwarding scheme for requests is to create a PIT entry only at the source node of the request and wherever a request is duplicated over two or more downstream paths. In particular, a request that is sent on a single (unicast) path generates only one PIT entry in its source node. The idea is that the PIT entries for a given request record the forwarding tree of that request but without superfluous intermediate nodes. Since such backward paths are very short—one or very few hops for most requests—the memory overhead is much reduced, but at the same time the path cannot be followed directly hop-by-hop. We solve this problem by sending replies upstream using standard IP forwarding. This scheme is depicted in Figure 5.

When a node v receives a request, it checks whether it can satisfy the request with a content segment cached in its content store, and if not, whether the same request is already in the PIT. If the request is already in the PIT the source address of the request is added to the upstream list of the PIT entry as in CCN. If the request cannot be satisfied and the entry is not in the PIT, v proceeds to forward the request according to the forwarding scheme. If the forwarding decision is to send the request along multiple interfaces, then v records the request in its PIT. This is what happens in node h in Figure 5. Notice that the PIT entry records the upstream link by copying the source/fork address from the request packet (labeled "src" in Figure 5) as well as the number of downstream packets (two, in the example). All



⟨**request**:id=36,"/path/x",src= j ,...⟩

PIT

req. id	prefix	down	up
36	"/path/x"	2	j

⟨**request**:id=36,"/path/x",src= h ,...⟩

Figure 5: Requests and Replies

forwarded packets then carry v's address (h in the example) in their source/fork header.

When a node v receives a request that it is able to satisfy, either because it hosts the application that advertised a matching prefix, or because a content segment with a matching prefix is cached in the content store, v produces a reply that is sent back following its branch in the tree of PIT entries towards the source of the request. The first hop of the reply is to the "source/fork node" address taken from the header of the request. Then, from there, if that is not already the source of the request, the reply follows the upstream link towards the source. After forwarding a reply upstream, v also removes the corresponding entry from the PIT so that no other (duplicate) replies will be sent that way. If the reply does not carry any content, then the reply is interpreted as signaling that no content is available on the path from which the reply comes. Upon receiving such a reply, v simply decrements the counter of pending downstream requests in the corresponding PIT entry (labeled "down" in Figure 5). When that counter drops to zero, it means that all requests forwarded downstream are negative. Therefore, v itself transmits a negative reply upstream and then removes the PIT entry.

4.4 Discussion

Both on-demand content delivery and publish/subscribe event notification map directly onto the common content-based network layer. For on-demand content delivery, a producer registration is implemented by setting the address of the producer, and interest and data packets map to requests and replies, respectively. For publish/subscribe event notification, a subscription is implemented by setting the content-based address of the subscriber, and an event notification is implemented as a one-way message.

The main benefit of this common layer is that the two forms of communication share the forwarding tables and, therefore, require only a single routing infrastructure. The architecture we propose is also modular with respect to forwarding, in the sense that it can be implemented with a variety of existing and new forwarding schemes. This modular forwarding scheme is assumed to avoid loops, which means that requests do not have to be recorded at each hop, which in turn means that the overhead for in-network state can be greatly reduced.

5. ACKNOWLEDGMENTS

This work was supported in part by the Swiss National Science Foundation under grant number 200021-132565.

6. REFERENCES

[1] B. Ahlgren, C. Dannewitz, C. Imbrenda, D. Kutscher, and B. Ohlman. A Survey of Information-Centric Networking (Draft). In B. Ahlgren, H. Karl, D. Kutscher, B. Ohlman, S. Oueslati, and I. Solis, editors, *Information-Centric Networking*, number 10492 in Dagstuhl Seminar Proceedings, Dagstuhl, Germany, 2011.

[2] A. Carzaniga, D. S. Rosenblum, and A. L. Wolf. Design and evaluation of a wide-area event notification service. *ACM Transactions on Computer Systems*, 19(3):332–383, Aug. 2001.

[3] A. Carzaniga, M. J. Rutherford, and A. L. Wolf. A routing scheme for content-based networking. In *Proceedings of IEEE INFOCOMM*, Mar. 2004.

[4] A. Carzaniga, G. Toffetti Carughi, C. Hall, and A. L. Wolf. Practical high-throughput content-based routing using unicast state and probabilistic encodings. Technical Report 2009/06, Faculty of Informatics, University of Lugano, Aug. 2009.

[5] A. Carzaniga and A. L. Wolf. Content-based networking: A new communication infrastructure. In *NSF Workshop on an Infrastructure for Mobile and Wireless Systems*, number 2538 in Lecture Notes in Computer Science, pages 59–68. Springer-Verlag, Oct. 2001.

[6] A. Carzaniga and A. L. Wolf. Forwarding in a content-based network. In *Proceedings of the ACM SIGCOMM Conference on Data Communication*, pages 163–174, Aug. 2003.

[7] C. Intanagonwiwat, R. Govindan, D. Estrin, J. Heidemann, and F. Silva. Directed diffusion for wireless sensor networking. *IEEE/ACM Transactions on Networking*, 11(1), 2003.

[8] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. H. Briggs, and R. L. Braynard. Networking named content. In *Proceedings of the 5th International Conference on Emerging Networking Experiments and Technologies*, pages 1–12, 2009.

[9] P. Jokela, A. Zahemszky, C. Esteve Rothenberg, S. Arianfar, and P. Nikander. LIPSIN: line speed publish/subscribe inter-networking. In *Proceedings of the ACM SIGCOMM Conference on Data Communication*, pages 195–206, 2009.

[10] P. Millard, P. Saint-Andre, and R. Meijer. XEP-0060: Publish-subscribe. Draft Standard of the XMPP Standards Foundation, July 2010. http://xmpp.org/extensions/xep-0060.html.

[11] P. Saint-Andre. Extensible messaging and presence protocol (XMPP): Core. RFC 3920, Oct. 2004.