Into the Moana[1] — Hypergraph-based Network Layer Indirection

Yan Shvartzshnaider*†, Maximilian Ott*, Olivier Mehani*,
Guillaume Jourjon*, Thierry Rakotoarivelo*
*National ICT Australia (NICTA)
Email: first.last@nicta.com.au

David Levy†
† The University of Sydney, Australia
Email: first.last@sydney.edu.au

***Abstract*—In this paper, we introduce the Moana network infrastructure. It draws on well-adopted practices from the database and software engineering communities to provide a robust and expressive information-sharing service using hypergraph-based network indirection.**

Our proposal is twofold. First, we argue for the need for additional layers of indirection used in modern information systems to bring the network layer abstraction closer to the developer's world, allowing for expressiveness and flexibility in the creation of future services. Second, we present a modular and extensible design of the network fabric to support incremental architectural evolution and innovation, as well as its initial evaluation.

I. Introduction

While the Internet has inspired innovation and evolution of new and exciting services, its initial design, the TCP/IP model, has remained relatively the same. The TCP/IP model provides a host-driven packet delivery service to the upper application layer. This contrasts to the majority of today's services that are primarily about content dissemination and consumption. It rapidly becomes less about sending a particular file to someone and more about where to store and share it. The core Internet fabric is not designed to deal efficiently with such service abstraction and therefore often finds itself under strain to meet the overwhelming demand from content-driven applications.

Motivated by this and other challenges [1] various recent research projects (PSIRP [2], CCN [3], DONA [4]) have emerged to tackle these problems by rethinking the network design fundamentals. These efforts offer a step towards a paradigm shift in the network service by introducing the notion of retrieving location-independent named objects, whereby the network service comes closer to resembling a storage system: you put labelled content in and you expect to get it out when you need it. The "fetch content by its label" service is a much more useful abstraction for most applications developers than one based on "sending packets from A to B."

Nevertheless, the rich and heterogeneous nature of today's Internet services calls for additional higher layers of data indirection to allow for the needed expressiveness in the communication between endpoints and the network. The networking community can "benefit from adopting the database community's perspective on reusable infrastructures for data independence" [5]. Data independence is a data-management method that allows for innovation at each of the levels of the system, without affecting other levels, by abstracting the way data is stored and retrieved.

The network efforts so far [6], [7] have focused on a single level of indirection that abstracts the notion of routing named content. Such a simple level of indirection is not sufficient for a network service that loosely couples a large number of information-driven applications with varied data formats and communication protocols.

In this paper, we propose Moana, a general-purpose hypergraph-based network indirection that merges various concepts from the network and database fields to realise a distributed and decentralised information network. Moana leverages the principle of data independence to introduce support for a (hyper)graph abstraction at the network layer. The graph abstraction maps well to modern practices in designing applications such as the use of the Unified Modelling Language (UML) or Entity-Relationship (E-R) diagrams, and also resonates with newly emerging trends in data-centric application development (*e.g.,* the semantic web's Linked Data initiative [8] and information-rich applications [9]). Moana thus allows for a more natural and richer communication between the network and applications using it, while making a clean-slate modular and extensible network fabric possible.

The remainder of the paper is organised as follows. We start with an overview of recent related work to see how Moana fits in. Section III discusses Moana's service model, data independence approach and network fabric. The implementation of Moana's service primitives is detailed in Section IV. The main concepts behind the Moana proposal are summarised in Section V. We evaluate the benefits of using Moana over CCN in Section VI and conclude the paper with a look at future work in Section VII.

II. Related Work

Moana's design draws on well-adopted practices from the database and software engineering communities to provide a robust and expressive network abstraction as well as an evolutionary design of the network fabric. In this section we summarise the highlights of the related work in that space.

[1]Pronounced "moh-ah-nah," meaning "ocean" in Hawaiian.

Service indirection. The fundamental gap between the network service abstraction and that used by top level applications, which are primarily about content, has inspired the discussion on content-centric networks. Although many of the proposals [2], [3], [4] have different technical implementations, they all argue for a common set of design goals for Information Centric Networking[1] (ICN) [7]: 1) Introduce a persistent content naming scheme. 2) Do away with a host-driven architecture and let the network route named-content instead. 3) Shift to a publish/subscribe communication paradigm. 4) Leverage in-network storage for content caching. More recently, however, questions have been raised on the ability of such an architecture to evolve and foster future innovation of new applications and services. Some cite the lack of expressiveness in communication between the end-host applications and the network as one of the main factors, as it is not trivial for individual applications to express their intent to the network service [10]. Others note the scarcity of architectural modularity, indirection and extensibility in the design to cater for future incremental changes [11], [12].

Service expressiveness. The eXpressive Internet Architecture (XIA) [10] addresses the "expressiveness problem" through the introduction of *principal types* and *fallbacks* for different communication styles between the network and the end-host applications. Principals are a way for an end-host application to communicate the type of service they require from the network, while fallbacks specify alternative services should the requested one not be available.

While motivated by the same subset of problems, the Moana approach is different in that it focuses on defining a generic way to encode and exchange information between the service and the network rather than standardising a particular service API. XIA's principal types and fallback mechanisms constitute information, and can therefore be transposed into an ontology on top of the graph of the "Moana world."

Network modularity. Modularity and support for extensibility in architectural design is a well-known practice in software engineering. A modular system comprises a set of individual components that are loosely coupled to execute a task. In a large and composite design, modularity makes for a robust and manageable system [13]. These characteristics are particularly vital to support incremental evolution of any system.

Ghodsi *et al.,* propose a new design for Ongoing and Pervasive Architectural Evolution (OPAE) in [11], based on additional indirection for flexibility and modularity to limit the scope of the change and extensibility to facilitate new functionality.

The Moana architecture follows the same design principles as OPAE. However while our design goals and principles overlap, the Moana architecture provides different mechanisms to facilitate indirection and modularity. In Moana all the components are loosely coupled through an abstract notion of a port. The basic functionality of a port is to receive and send out content fragments and tuples. We discuss these in detail in the next section.

Data Independence. Classical database systems are designed to provide three layers of data abstraction [14]. First, and the lowest, is the *physical layer* that defines how to store data. Next, comes the *logical layer* that describes the data and its relationships to other data. The third and highest is the *view layer* which is used to show parts of the entire system.

In [5], Hellerstein makes a case for leveraging data independence and its importance in designing systems that are robust to changes. He examines various large-scale content distribution networks through the "database lens" and finds that they have more in common than might appear at a first glance. Hellerstein calls upon the networking community to draw on some of the practices used by the database community when designing large-scale network applications.

Moana takes up the concept of data independence in its design to decouple the service logic from the underlying content delivery network. We describe the role and properties of each level of indirection later on.

III. Moana Design & Architecture

We first describe the basic service model behind Moana. We then go into more detail on Moana's design, first covering the data abstraction and then focussing on the architectural realisation.

A. Service Model

As mentioned in the introduction, the majority of popular Internet services such as Google, YouTube and various social networking sites are about information dissemination. A graph abstraction plays a key role in these services that use graphs to capture the underlying relationships. Currently each service or application maintains its own graph with custom APIs. In fact, the Internet is full of diverse well-guarded graphs with partially replicated information.

Reminiscent of the pre-Internet era,[2] there are now calls for a more generic approach to provide a generic graph infrastructure that interconnects and leverages by these "information graphs" which are currently partially replicated across many applications [15], [16], [17]. We therefore propose a service model to bring these graphs into a single information network—the kind envisioned by Tim Berners Lee [15]—on top of which additional services that interact and manipulate this information space can be built.

Moana is an information dissemination network layer based on a hypergraph abstraction. This service model is designed to support applications in publishing to and requesting information from the network. It departs from the traditional model of defining network protocols for particular services, and proposes a generic mechanism to request and share information between any applications using the network. A

[1]This is in fact content-centric networking. As noted by [7], "information is an abstraction on the next higher level. In order for data to become information, it must be interpreted and must take on a meaning."

[2]Internet stands for INTERconnected NETworks and was designed to bridge a variety of different data networks.

publication makes the content available in the network with a description of meta-data as a set of n-tuples. Conversely, a request is transformed into a standing graph query on desired content attributes and results in the delivery of the content to the application.

B. Data Independence

The support for data independence in a system limits the impact of changes at each level. While developers can offer different functionalities at higher levels of data abstraction, the services operating at the lowest layer remain unchanged [14]. Moana realises these layers as follows.

The View Layer. The view abstraction provides applications with a uniform functionality to view, access and add new data. It defines the language primitives used by applications to interact with the network using the Publish/Subscribe API.

The Logical layer. In the Moana design the logical layer employs a directed hypergraph abstraction to describe entities and their relationships to other entities. A hypergraph is defined by a set of n-tuples [18] (we refer to them as tuples for remainder of the paper). In Moana, tuples contain six elements:

```
< subject:GUID, predicate:GUID,
object:Any, context:GUID,
publisher:GUID, createAt:timestamp >
```

Nodes in the hypergraph representing entities, as well as the edges are identified by a Global Unique Identifier (GUID). Entity attributes hold their respective values; the types of these values are restricted to standard types like *string*, *numeric* or *blob* for images or videos. Due to their arbitrary size and the different ways with which content is often produced and consumed, blobs are labelled with a GUID and split into a sequence of *content fragments* (CF). The `context` element can be used to capture additional provenance details.

The Physical layer. Moana's physical data layer serialises tuples into blocks suitable for transmission on the transport substrate. To transfer content blobs (*i.e.,* CFs) it borrows the ICN's physical data model for storage and retrieval of named-chunks.

C. Network Fabric

Having described the three layers of the data abstraction supported by Moana we move on to give an overview of the proposed network fabric that supports their existence.

The Moana network supports two main functionality primitives: 1) hop-by-hop forwarding, and 2) reasoning over tuples. It comprises a physical layer of forwarding engines linked through ports to applications, routing engines, fragment stores and channels, as shown in Figure 1.

Forwarding Engine (FE). The main task of the FE is to forward tuples and CFs by consulting the forwarding table,

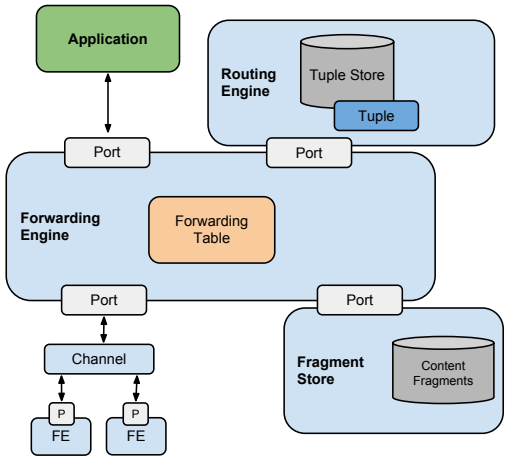

Fig. 1. The Moana network fabric

```
POLICY Flood
FORWARD [FE1_guid, FE2_guid, ...]
WHERE
 ?content type_guid TypeA_guid
 ?content publishedBy_guid ServiceB_guid
```

Listing 1. Simple policy instructing the FE to forward any content published by a particular application

which maps GUIDs to Forwarding Engine Instructions (FEI). An FEI contains the port interface on which to forward, plus additional information such as the designated neighbours' GUIDs, in case the FE is connected to a larger network.

Routing Engine (RE). The RE keeps the state of the policies that the FE follows. The policies are defined by graph queries on the tuple store attached to the RE, with a corresponding action, similar to that illustrated in Listing 1, where the notation $element_{guid}$ denotes the GUID for that element. The implementation for the tuple store may vary.

- **Local tuple store.** The local tuple store contains only a partial view of the information space. For example, an RE can store every tuple that it receives.
- **Centralised Tuple store.** A centralised tuple store acts as a global information space. The RE passes the query to a globally known tuple store. The local tuple stores at each FE may then act as caches.
- **Distributed tuple store.** A global information space consists of the distributed local tuple stores at each RE. This effectively allows the RE to query locally to access global content.
  One way to implement this is to adapt the Rete algorithm [19], an efficient pattern matching algorithm frequently used in production rule systems, onto a distributed storage environment [20].

Content Fragment Store (CFS). The CFS is attached to an FE to allow caching of content fragments. The CFS also provides an API that an RE can use to trigger the forwarding of particular content fragments.

Port. Ports provide an abstract interface used by all the network components to interconnect. FEs do not distinguish between forwarding an incoming CF to an outgoing (network) port, a storage port, or a routing engine for that matter. What to store and where to forward is left to the RE and is achieved by manipulating the FE's Forwarding Table (FT).

Channel. Channels allow FEs attached to them to exchange messages. The most basic channel provides a simple broadcast mechanism, while most practical channels will also provide unicast with optional multicast capabilities. Channels also provide a mechanism for all attached elements to eventually discover other attached entities. Additional mechanisms may exist for elements to learn about channel properties and be notified of changes to these properties, *e.g.,* the quality and availability of a wireless channel.

IV. Moana Service Primitives

The service provides two main service primitives: *Publish* and *Request*. We refer to any content produced by an application as a content unit (CU), an abstract concept understood only at the view layer.

Publication. A publication starts with an application connecting to an FE through a port and publishing a CU with a set of tuples describing it. The lower layers break the CU's blob down into a set of CFs. They are stored in the local CFS. The meta-data tuples are forwarded through a local port to the RE for processing. The RE can reason over these to take an action governed by the policy. For example, a policy can instruct the RE to populate the FE's forwarding table with FEIs that forward any CF related to a particular CU to the immediate neighbours, or forward any arriving tuples to its attached tuple store.

Request. A request consists of the following five phases: 1) An application connects to an FE through a port. 2) The service sends through a request as a graph query. 3) The graph query is decomposed into a set of tuples bound by the binding variables on the way to the FE. 4) The FE passes the tuples to an RE attached to it. 5) The RE maintains the state of the query, while fetching tuples matching the query from its attached tuple store.

Returning and processing results. Query-matching tuples returned by the RE identify relevant CFs by their GUID, as well as any additional information on how to reach the FEs containing the CFs. We assume that FEs can obtain CFs once their GUIDs are known. This is not an unreasonable assumption, as the majority of ICN architectures are designed for that. The application-hosting FE receives the CFs and forwards them to the application.

The additional returned information depends on the underlying content delivery service. For an HTTP-like protocol it can be a URL, for CCN [3], a hierarchical name that is used for routing it. Also, since each FE is able to reason over tuples (with help of their respective REs), the additional information can serve as a "hint" as to what direction to route the "fetching" request to, in case a direct path wasn't specified. For example, an FE can use the CU context (specified as part of a tuple) rather than directly using the GUID of the CU to route it. In case there is more than one possible route, the meta-data tuples might include alternative routes, similarly to Slick Packets [21]. However with Moana these routes can include a possible combination of networks.

V. Putting it all together

So far we have described the main concepts behind the Moana design and architecture. We now discuss their roles with a conceptual example that illustrates how everything comes together.

A. Main actors

The Moana service model abstraction facilitates building large network applications in a more abstract way. We choose to describe a "content sharing" functionality from a view of various stakeholders: end-user, application developer, and service provider.

End-user. Moana's design does not directly target the end-user. It deals with the interaction between the application and the network. The end-users will use the application through an interface familiar to them.

Application developer. Every application is driven by their application logic. To build an app to share content, a developer needs to design an information model that captures this logic. For the purpose of our example, we present two of the simplest models:

```
model User {
 id: GUID
 name: String
 friends: Set<User>
}
```

```
model Content {
 id: GUID
 type: GUID
 publishedBy: User
 tags: Set<GUID>
 relatedTo: Set<GUID>
}
```

The above models are self-explanatory and represents a very simplified social network interaction. The User can post Content, and be linked to other Users. These models can be defined using any of the modern programming languages, however we expect a more native solution, similar to Jenabean [22] for the Semantic Web that automatically binds Java types to the RDF[3] graph to be developed for such a hypergraph-based network service. The functionality of the application will be done by using the service primitives: *Publish* and *Request*.

Service provider. From a service provider's point of view, having intrinsic support for graphs within the network means that the application has more control on how the content is propagated through the network. Among other benefits, the

[3]Resource Description Framework

```
REQUEST ?content
WHERE
  ?content type_guid Movie_guid
  ?content publishedBy_guid FriendA_guid
  ?content tags_guid [Tag1_guid, Tag2_guid, ...]
```

Listing 2. Request graph query for a content of type "Movie" that was published by a Friend and matches a specific set of tags

content service provider can take full advantage of the small-world phenomena often found in social driven networks [23]. In YouTube, for example, the content is more likely to be fetched next if it has a relation to the currently watched video [24]. This can lead to more efficient policies for storage and caching; more so, much of it can be done on the fly in an automatic fashion as a result of reasoning over tuples accompanying the content. The service provider might have different policies to ensure a more effective, secure, or even possibly "greener" content delivery.

B. Illustrative example

Once the model has been implemented, the developer can build more complex functionality around it, while presenting the user with a friendly interface.

On the application side, for instance, the "Wall" feature that lists recently posted content is common in social-driven content sharing services. This feature comprises many requests to the network for different types of content. A lot of them will rely on the functionalities such as "*PostContent (content: Content)*" and "*FetchUserContent(user_id: GUID)*," where the developer will use the primitives described in Section IV*, namely,* the *Publish* and *Request* methods.

The actual content (*e.g.,* image or video), is split into a set of CFs. The meta-data describing it is encoded into a collection of tuples. The CFs and tuples are passed to the CFS and RE, respectively. The tuples can remain stored in a local tuple store or be further disseminated based on the RE policy, enacted by FEIs.

For the request, a standing graph query, similar to Listing 2 is injected into the network. As mentioned in Section IV, the query is decomposed into a set of tuples. The FE forwards the tuples to the RE that, in turn, queries the tuple store attached to it. As a standing query, the request will be monitored by the RE for any CFs or tuples matching it. The request facilitation depends on the implementation of the attached tuple store—local, centralised or distributed—and the policies for the request dissemination set by the network administrator. Once there is a match, the RE will instruct that the CFs and tuples be forwarded to the application through a relevant port. This will trigger callback functions such as "display it on the Wall".

The functionality described above is very simple, but serves to illustrate the ease with which the logic of an application is mapped into a network service that understands hypergraphs.

VI. Evaluation

We now evaluate Moana's dissemination policy. We would like to test the hypothesis that the Moana network service brings a reduction in the effort required to fetch content.

We choose to conduct our measurement campaign on top of ccnSim [25], a chunk-level CCN simulator based on OMNeT++ [26]. In this framework, we have implemented two additional modules `application` and `moana` wrapped in a compound `node` module together with `ccn_node`. The `application` modules operate using the Moana API. They can publish and request information, while the `moana` module in turn uses CCN to deliver the content.

Scenario. We simulate a simple social network scenario where a user is followed by a number of friends who are interested in receiving notifications on any content published by the user. We also assume that the path to all the friends is known.

Experiment. We use a network based on a one hundred strongly-connected grid-nodes topology (while not quite realistic, this still allows us to conduct various performance and scale comparisons; more realistic topologies are kept for future work). The CCN network was configured to use a Least Recently Used (LRU) cache replacement policy. At every simulation run, we elect a node P in the network to act as a publisher. This specific node will create and publish a content C as well as its meta-data containing: content ID, size and other attributes. Meta-data is then disseminated to the publishers' friends. On receiving meta-data, every friend requests P for the content C. In the remainder of this section, let $\mathbb{F}$ be nodes that are friends with P and $f \in \mathbb{F}$ be a friend node of the publisher P.

We run two modes for content retrieval, *CCN-only* and *Moana/CCN*. In the *CCN-only* mode, friend-nodes request content using its ID while in the *Moana/CCN* mode friend-nodes take advantage of the social graph with the Moana service layer to request content from other friends of P. We can note that, as mentioned throughout this article, with Moana the requesting node does not need to know the exact content ID, and can base its requests solely on the attributes of the content C such as publisher, type or context.

We conducted a series of simulations over the same topology with the same publisher node P. We randomised the content size as well as the selection of subset $\mathbb{F}$. Once these parameters are selected we run the simulation once for each mode. We also selected the friend group size as another independent variable of our experimentation and chose two levels of 5 and 10 friends.

Metric: We use the downloading effort from [25] as our metric. The *downloading effort* is denoted as:

$$\eta_C = \frac{\sum_{i=1}^{\text{size}(C)} d_i}{|D|\,\text{size}(C)}, C = \{c_1, c_2, ...c_n\}$$

where $\text{size}(C)$ denotes the size of the content C in chunks, d_i is the distance travelled by chunk c_i in terms of hops and $|D|$ is the shortest distance to the publisher. η_C ranges from

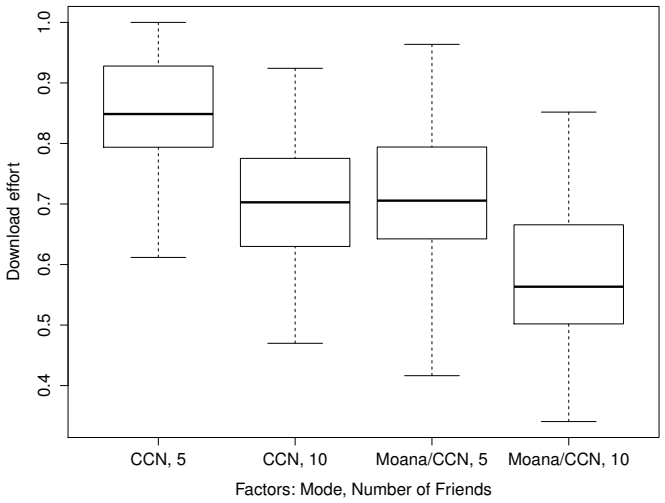

Fig. 2. Simulation results: Moana/CCN reduces the effort it takes for content to reach interested parties as compared to CCN alone.

0—no effort—to a maximum of 1—the effort it takes to get the entire content from the original publisher.

Results. Figure 2, shows box plots of effort results from 100 runs for the *CCN* and *Moana/CCN* modes with different friend set $\mathbb{F}$ sizes, first with 5 and then 10. It suggests the *Moana/CCN* mode has a significant impact in reducing the effort it takes to get a chunk. Moreover, increasing the number of friends in the set appears to further reduce the overall effort for the chunks in both modes.

To further confirm that the null hypothesis that *there is no difference between the populations of means under different treatments of the* mode *and* number of friends *factors* can be rejected, we performed ANOVA tests with significance level $\alpha = 0.05$ (for a 95% confidence level when rejecting the null). We found that the null hypothesis could be rejected for both factors ($p < 0.05$), and that their interaction was not significant ($p > 0.05$). Changing the mode and the number of friends does have a significant effect on the average effort. Tukey HSD tests identified a reduction in the effort mean of approximately 13% (1.28×10^{-1}) when using Moana.

It is worth noting that we had to use a non-parametric multivariate version of the ANOVA test as some of our result data sets did not match the normality assumption. In our experiment, when using *CCN* with only 5 friends, the effort was skewed towards 1, which we hypothetise is due to the limited number of friends in this scenario not allowing the capabilities of the mechanism to be exploited fully. This did not appear with CCN/Moana and the same number of friends. We plan to investigate this in detail in future work.

VII. Final remarks

In this paper we have described Moana, a new a hypergraph-based network service abstraction. It is a step closer to the developers' world, and allows for easier application development and deployment by hiding the particulars of the underlying content delivery infrastructure. Furthermore, the support for tuples opens new avenues in policy-based routing and network management. It can help the network core exchange and reason over information at the lowest levels using high level terms. In our future work, we plan to examine more information dissemination policies, use more realistic topologies and evaluate the performance with other transport protocols in addition to CCN.

References

[1] M. Conti, S. Chong *et al.*, "Research challenges towards the Future Internet," *Computer Communications*, vol. 34, no. 18, 2011.

[2] D. Lagutin, K. Visala, and S. Tarkoma, "Publish/subscribe for internet: Psirp perspective," *Towards the Future Internet Emerging Trends from European Research*, vol. 4, pp. 75–84, 2010.

[3] V. Jacobson, D. Smetters *et al.*, "Networking named content," in *Proc. of the 5th Inter. Conf. on Emerging networking experiments and technologies*. ACM, 2009, pp. 1–12.

[4] T. Koponen, M. Chawla *et al.*, "A data-oriented (and beyond) network architecture," in *ACM SIGCOMM CCR*, vol. 37, no. 4. ACM, 2007.

[5] J. Hellerstein, "Toward network data independence," *SIGMOD Rec*, vol. 32, pp. 200–3, 2004.

[6] I. Stoica *et al.*, "Internet indirection infrastructure," in *ACM SIGCOMM CCR*, vol. 32, no. 4. ACM, 2002, pp. 73–86.

[7] B. Ahlgren, C. Dannewitz *et al.*, "A Survey of Information-Centric Networking (Draft)," http://drops.dagstuhl.de/opus/volltexte/2011/2941, Feb. 2011.

[8] " Linked Data - Connect Distributed Data across the Web ," http://linkeddata.org/.

[9] "What's New for Visual F-Sharp in Visual Studio 2012 RC," http://msdn.microsoft.com/en-us/library/hh370982%28v=vs.110%29.

[10] A. Anand, F. Dogar *et al.*, "XIA: an architecture for an evolvable and trustworthy internet," in *Proc. of the 10th ACM HotNets Workshop*. ACM, 2011, p. 2.

[11] A. Ghodsi, S. Shenker *et al.*, "Intelligent design enables architectural evolution," in *Proc. of the 10th ACM HotNets Workshop*. ACM, 2011.

[12] T. Koponen, S. Shenker *et al.*, "Architecting for innovation," *ACM SIGCOMM CCR*, vol. 41, no. 3, pp. 24–36, 2011.

[13] G. Myers, *Composite/Structured Design. 1978*. New York, NY: Van Nostrand Reinhold.

[14] S. Kedar, *Database Management Systems*. Technical Publications, 2007.

[15] T. Berners-Lee, "Giant global graph - decentralized information group (DIG) blog post," http://dig.csail.mit.edu/breadcrumbs/node/215, Nov. 2007.

[16] B. Fitzpatrick, "Thoughts on the Social Graph," http://bradfitz.com/social-graph-problem/, Aug. 2007.

[17] A. Iskold, "Social Graph: Concepts and Issues," http://www.readwriteweb.com/archives/social_graph_concepts_and_issues.php, 2007.

[18] J. Sowa, *Conceptual structures: information processing in mind and machine*. Addison-Wesley Pub., Reading, MA, 1983.

[19] C. Forgy, "Rete: A fast algorithm for the many patterns/many objects match problem," *Artificial Intelligence*, vol. 19, no. 1, pp. 17–37, 1982.

[20] Y. Shvartzshnaider, M. Ott, and D. Levy, "Publish/subscribe on top of DHT using RETE algorithm," *Future Internet-FIS 2010*, 2010.

[21] G. Nguyen, R. Agarwal *et al.*, "Slick packets," *ACM SIGMETRICS Performance Evaluation Review*, vol. 39, no. 1, pp. 205–216, 2011.

[22] "Jenabean project home," https://code.google.com/p/jenabean/.

[23] A. Mislove, M. Marcon, K. Gummadi *et al.*, "Measurement and analysis of online social networks," in *Proceedings of the 7th ACM SIGCOMM conference on Internet measurement*. ACM, 2007, pp. 29–42.

[24] X. Cheng, C. Dale, and J. Liu, "Statistics and social network of youtube videos," in *Quality of Service, 2008. IWQoS 2008. 16th inter. Workshop on*. Ieee, 2008, pp. 229–238.

[25] D. Rossini and D. Rossi, "A dive into the caching performance of content centric networking," Technical report, Telecom ParisTech, Tech. Rep., 2011.

[26] A. Varga *et al.*, "The OMNeT++ discrete event simulation system," in *Proceedings of the European Simulation Multiconference (ESM2001)*, vol. 9. sn, 2001.