Charting an Intent Driven Network

Yehia Elkhatib
School of Computing and Communications
Lancaster University, UK
{*i.lastname*}@lancaster.ac.uk

Gareth Tyson
EECS
Queen Mary, University of London
London, UK

Geoff Coulson
School of Computing and Communications
Lancaster University, UK

***Abstract*—The strong divide between applications and the network control plane is desirable for many reasons; but a downside is that the network is kept in the dark regarding the ultimate purpose(s) of applications and, as a result, is unable to optimize for these. An alternative approach is for applications to declare to the network their abstract intents and assumptions; *e.g.* "this application requires group multicast", or "this application will run within a local domain and is latency sensitive". Such an enriched semantic has the potential to enable the network better to fulfil application intent, while also helping optimize network resource usage across applications. We refer to this approach as *intent driven networking* (IDN). We sketch an incrementally-deployable design to serve as a stepping stone towards a practical realization of IDN within today's Internet.**

I. Introduction

A key principle of the early Internet was the provision of a very simple send/receive interface for applications. As the complexity of the Internet has grown and new capabilities been added (multicast, streaming, mobility, etc.), we have attempted to continue to live with this very simple interface. Although the approach has served us well so far, clear downsides are emerging. Essentially, applications operate in the dark with respect to the capabilities and functioning of the underlying network and are therefore obliged to include (potentially rather complex) logic to handle network related events such as faults, performance fluctuations, service changes (*e.g.* mobility), etc. Similarly, network operators do not understand application needs beyond the issuance of seemingly isolated *micro-transactions* (send and receive calls), and is thus unable to conserve resources or optimize performance for applications.

At the heart of the matter is the inability of the network to see the underlying *intent* of the application. Instead, the network only sees a series of micro-transactions. One solution is to extend the network API to give direct access to all the individual network elements such as caches, middleboxes, routers, etc. But this is clearly problematic for many reasons: application developers would find it too hard to understand, the API would regularly mutate in line with corresponding changes in the technologies, and it would promote unforeseen interactions between per-technology API elements, with unpredictable and probably undesirable consequences.

In this paper, we propose *Intent Driven Networking (IDN)* as an alternative approach. It enables the formulation of an application's "intents" as high level statements of its predicted macro-level behaviour, *i.e.* an abstract formulation of what it desires from the network, while remaining agnostic about the underlying means used to satisfy them (protocols, etc.). For example, an intent might be to communicate with a particular group of users (*e.g.* collaborative document editing); another might be to stream a video uninterruptedly while switching between using a laptop and a smartphone as well as from an 802.11 network to a cellular one. Whereas the current Internet sees sets of independent micro-transactions, an intent driven Internet would understand the aims and optimize accordingly.

IDN also allows us to simplify application development by removing the need to provide "cover all cases" logic. Instead, user application requirements are fed down to the network thus providing flexibility in how different requirements are met without predefined restrictions. For instance, one intent might implicitly ensure the availability of a certain service despite the failure of a remote server; another would ensure a certain level of Quality of Experience (QoE) even if the application is not designed to seek alternative potentially better routes; etc. As a consequence, IDN facilitates more fluid development of end user applications and is conducive to better alignment of the network to application needs.

Another good example arises in the context of mobility support. Currently, host mobility is an extremely complicated process, typically managed in the lower layers of the network stack by Mobile IP. In essence, these lower layers attempt to hide the effects of host mobility (*e.g.* changes in IP addresses) from the higher layers (*e.g.* applications) using costly mechanisms such as tunnelling. This measure is necessary as nearly all applications assume stable, globally reachable addressing, as well as consistent connectivity, none of which are valid in the case of mobility. These needs, however, often only emerge because the lower layers have no understanding of the *intents* that exist at the higher layers. Imagine, for example, that an application has no need for consistent addressing or, alternatively, only requires access to content within the local domain (*i.e.* no need for global addressing). In such circumstances, the constraints are relaxed and Mobile IP's blanket-style behaviour becomes unnecessary. However, due to the network's inability to see the application's intent, there is no way to decide when such principles should or should not be applied. Consequently, the lowest common denominator must always prevail.

In order to attain the IDN vision, we need means by which application intents are *formulated*, *compiled*, and ultimately *reified*, *i.e.* acted upon, in the network (Fig. 1).

We present the concept of an intent driven network using

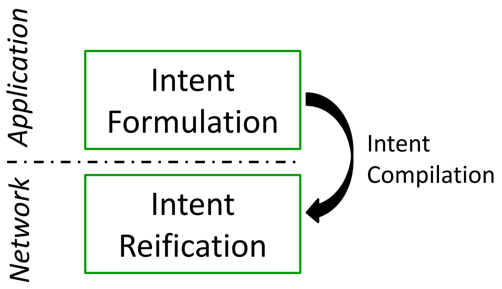

Fig. 1. High level view of an intent driven network.

both a straw man design (§II) and illustrative examples (§III). We focus particularly on the practical concern of how IDN might be incrementally and partially deployed in the existing Internet without restarting at year zero. We comment on the feasibility (§IV) and implications (§V) of this design, and then on related work (§VI), and finally lay out a roadmap for the incremental realization of IDN (§VII).

II. Intents

This section defines what an intent is (§II-A), proposes an approach for the formulation of intents based on compositions of primitive verbs (§II-B), and discusses the mechanics of reifying intents using in-network *mediators* (§II-C).

A. What is an Intent?

An *intent* is an abstract declaration of what the application desires from the network on behalf of the user. It is a composition of a set of primitive "verbs", each describing a specific but high-level operation. For example, an intent to update an Instagram feed might be composed of primitive verbs to reconfigure the application topology (connect to a service and to peers), exchange data (update the content), and uphold a certain QoE level (allocate sufficient network resources). In response to this, the network carries out the necessary configuration to best serve such an intent. This could entail setting up direct connections between users, and allocating fair shares of router queues considering other network services.

Essentially, intent expression is based on the *verb-object-subject* sentence structure used in linguistics, supplemented by *modifiers* as an additional set of words. Primitive intents expressed using such sentences are then composed using recursive encapsulation to form a full intent.

In more detail, the primitive elements that comprise intents are expressed using <*verb, object, modifiers, subject*> tuples. A *verb* is an operation that describes the intent based on an ontology (described next in §II-B). *Object* identifies a service, process or item that is the objective of the verb. *Modifiers* are then used to specialize or parameterize this; each modifier can be tagged as either 'essential' or 'desirable', indicating prioritization preference. *Subject* is an (optional) identifier of another service/process/item to be linked to the *object*.

Intents are not limited to only user applications; they extend to those operated by other network players (*e.g.* ISPs, cloud service providers, content providers) to express their own intents.

B. Formulating Intents

An intent verb is expressed using one of the ontology entries in Fig. 2. This is not a comprehensive ontology; modification and expansion is possible through collaboration with the wider research community. The ontology is divided into three operation categories: *Construct*, *Transfer*, and *Regulate*. Each of these categories has a number of sub-operations from which the verb is chosen. Categories are just logical groupings; it is the verbs that signify the primitive intent.

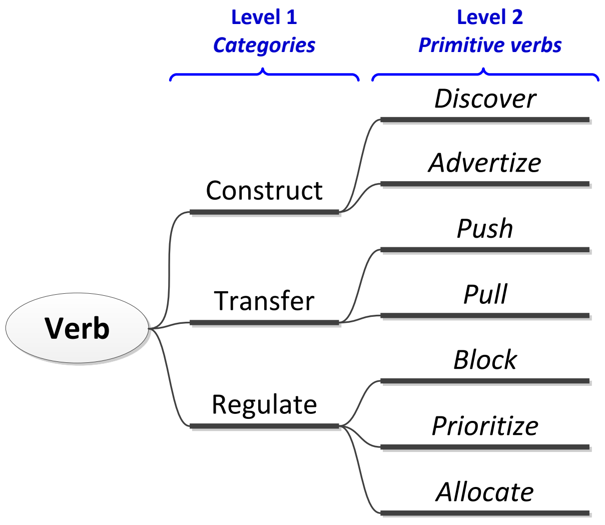

Fig. 2. A basic ontology of primitive verbs.

Construct is used when an application needs to form connections to another application (the *object*) in a peer-to-peer fashion, either locally over a broadcast address or remotely. An example is a request of intent for a VoIP client to connect to another VoIP client. *Discover* is issued to look for certain applications, whilst *Advertize* allows an application to announce a new service that is able to serve the intents of other applications. Examples include nodes spawning a caching or load balancing service. Announcing various applications to be discovered or advertized enables an application to dynamically employ external modules without the latter being a component of the native application code. This is of particular use to applications running with scarce resources, *e.g.* a mobile gaming application offloading its transcoding processes.

Transfer intents allow applications to pull and push content (the *object*). An item of content could be referred to in any of a number of different ways, as illustrated in Table I. In this sense, a *Transfer* intent is analogous to an ICN abstraction, where the *Push* verb corresponds to a prefix announcement whilst *Pull* corresponds to an interest packet.

Finally, *Regulate* intents capture the desire of an application to have traffic handled in a certain way in the network rather than locally. This is helpful for propagating traffic management logic higher up the network closer to the source,

TABLE I
EXAMPLES OF DIFFERENT WAYS TO REFER TO CONTENT IN A *Transfer* INTENT.

URL	`http://releases.ubuntu.com/16.04/ubuntu-16.04-server-amd64.iso`
CCN	`ubuntu.com/torrent/ubuntu-16.04-server-amd64.iso`
BitTorrent	`699cda895af6fbd5a817fff4fe6fa8ab87e36f48`

which facilitates better network management and aggregation of interests. An example is an intent to block ssh login attempts from a certain address block, or to prioritize traffic from a service like `hulu.com`.

C. Reifying Intents

Our conceptual architecture relies on a hierarchical structure of mediators deployed in the network. These are middleboxes that arbitrate between user intents, network and service operator policies, and the current state of the network. We refer to this mediation presence in the network as *Maat* and each of the middleboxes as a *Maat* agent, in reference to the ancient Egyptian concept of conflict resolution to achieve harmonious equilibrium and order.

Given this, user intents are initially sent on a specific broadcast address to be picked up by a local Maat agent. If a Maat agent is not available as signified by the expiry of a timer since issuing the intent, the application can widen the address scope to seek another agent in the parent subnetwork (and recursively so), or alternatively it could choose to fall back to non-IDN behaviour.

If a Maat agent is available and able to satisfy an intent, its job is to "reify" this intent by deploying or activating the required mechanisms (such as an in-network function) or identifying candidate services (a nearby deployment), and consequently sending the relevant information back to the user application to realign itself accordingly. The Maat agent is also required to create a session to keep track of how the intent was met. This is important for auditing mediation efficiency. We will discuss this further in §V-B.

III. EXAMPLES

We now further illustrate the formulation of intent and its corresponding reification through a set of use cases.

A. Use case #1: Clustering

Alice is editing a document with her colleagues Bob and Charlie on the Google Docs cloud service. Currently, this is handled in a manner that is analogous to a chat server where the collaborators connect to a cloud-based backend to push edits and/or receive updates. The problem with this is that it involves unnecessary communication back and forth to wherever the remote service is hosted when some or all of the collaborators may be within a short network distance of each other.

Using IDN, the local application on Alice's device (Google Docs in this case) would express its intent to share updates between Alice and a set of other users. For that, it needs to communicate with the Google backend to fetch the addresses of the collaborators' devices. These addresses are then used to formulate an intent as follows:

```
<allocate,
  ip_multicast,
  (ttl=32,essential),
  <discover,
    GoogleDocs,
    (userID=92cd701c0be,essential),
    (userID=33a88280853,essential),
    NULL
  >
>
```

Note that the intent here takes the form of a composition between *discover* and *allocate* verbs. Having first discovered the various players in this scenario (the GoogleDocs application and the relevant users), the network allocates a multicast group, and Maat responds with the group address. From then onwards, local communications are exchanged over this group, and Alice's application is responsible for sending periodic updates to the Google backend for backup if it requires. If the collaborator devices move or change, Alice will issue a new intent accordingly.

B. Use case #2: Discovery

Serendipitous peer discovery is important for emerging Internet applications. A particularly important application of discovery is in Internet of Things (IoT) environments where a large number of hosts would be operating different services in any one locality. Currently, discovery relies on the presence of directory or similar services, which obviously has its limitations in terms of consistency, scalability (considering the scale of IoT swarms to come [1]).

In such a context we might signal an intent to build a new overlay structure from a set of suitable nodes. This would work in a fashion similar to ARP; a node would seek other nodes that fit certain criteria on the service(s) they operate, location, communication mode, QoS metrics, etc. The network would then propagate this intent announcement according to the criteria laid out in the intent modifiers.

Consider for instance an actuator in an IoT deployment that wants to find a nearby node capable of running a MapReduce analytics workflow over a collection of sensor data. In this case, the intent is formed by composing a *discover* primitive verb with a *push* one as follows:

```
<push,
  dataset-201507-1800,
  (net=1.2.3.0/24,essential),
  <discover,
    hadoop,
    (rtt<50ms,desirable) &
```

```
      (rtt<80ms,essential),
    hadoop-workflow.jar
  >
>
```

This would be examined and collated by Maat to allow the intent to be expanded and traverse across different networks and operational environments (if within the specified criteria). As another example, an intent could emanate from a node in a sensor network seeking secure data storage, and use IDN to explore options as diverse as local fixed-power nodes and remote data centers. Such discovery may also extend beyond IoT and include intents formed at different levels, such as the ability to choose which middlebox (intrusion detection, proxies, interceptors, anonymizers, etc.) to go through.

It is important to explicitly note that this is *not* a proposal for another discovery broadcast protocol. On the contrary, a key objective of IDN is to enable applications to express high level intents and be agnostic about the means to satisfy them.

C. Use case #3: Edge Deployment

Finally, consider an example involving different stakeholders: content and service providers. Both of these stakeholders have a lot to gain from a strong presence towards the edge of the network in areas where there is demand for their services.

Consider for instance a content provider that finds an increase in the consumption of certain content (say the feature film “A Beautiful Mind” following the death of John Nash) in a particular area (say large metropolises in the US). It is in the provider’s interest to provide good viewing QoE for its customers and at the same time manage increased load on its backend services. Accordingly, it might decide to push copies of the content to cache in different cities.

In this case, the intent will be expressed as a composition of a verb that discovers suitable caching services (the *object*) in certain locales (the *modifiers*), a verb that pushes content to the discovered caching points, and a final verb to announce the new content once cached.

A full intent follows, where `asn` represents the AS number which signifies a certain customer base. Other modifiers could be used to identify target locales at a finer grain.

```
<push,
  ABeautifulMind,
  (auth=https://universalstudios.com/oauth),
  <push,
    831FD96B0.mp4,
    NULL,
    <discover,
      cache,
      (asn=123456,essential),
      NULL
    >
  >
>
```

In a similar fashion, a service provider might deploy applications to nodes offering hosting services to balance load at the edge, mitigate flash crowds, or improve user QoE.

IV. Feasibility

We now present a preliminary investigation of the overhead associated with the straw man design we put forward. For this, we use the intent types given in the above examples and identify the costs incurred for a real campus network.

To host the Maat mediators, we use Raspberry Pi devices as they are highly affordable and have reasonable hosting capabilities [2].

We focus on assessing the ability of an in-network mediator in processing incoming application intents as the initial bottleneck in this process. This is important for *intent mediation dimensioning*, *i.e.* to identify the number of mediators required to cater to expected user application intents in a given network.

For this we develop a Python module to act as a lexer and recursive descent parser with backtracking. The module receives over the network intents that are dynamically created based on the examples given in §III. It parses these intents in order to identify possible reification paths for each intent. We run this over 2 different setups: a Linux box with an Intel i5 1.30GHz processor and 4GB of RAM and running Ubuntu version 16.04, and also on a Raspberry Pi 2 with Raspbian Jessie kernel version 4.9 We vary the number of issued intents between 10 and 1000 intent per second, repeating each input size 10 times.

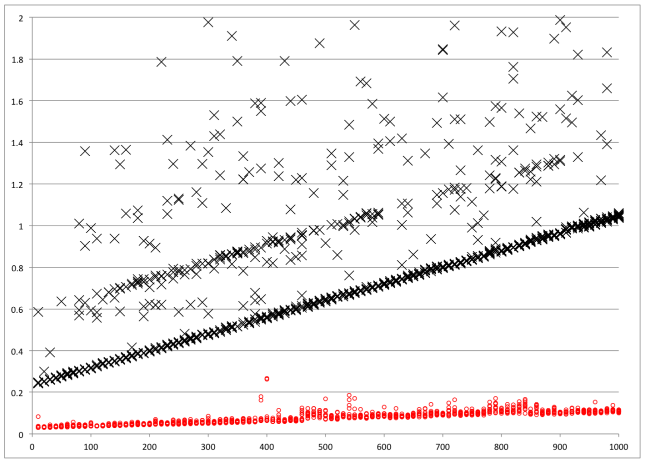

Fig. 3. The amount of time taken (vertical axis in seconds) to parse intents (horizontal axis. °for the Linux box, and X for the Raspberry Pi 2.

The results, shown in Fig. 3, demonstrate that intent parsing is significantly sustainable to varying degrees. On the Linux box, parsing is extremely quick, being able to sustain upto 1000 intent rate under 0.2 seconds, excpet in limited cases when an intent has several layers of encapsulation.

V. Implications

In realising IDN we do not propose the re-writing of the entire network stack[1]. Instead, we propose to overlay the concept of intents onto the existing Internet architecture. As such, our straw man IDN architecture has been designed to

[1]Although an IDN architecture could indeed be approached from this angle, it is likely to create excessive disincentives that limit its deployment.

support **backward compatibility** (falling back to non-IDN behaviour) and **incremental/ partial deployment**.

As already described, IDN pushes some of the meta-logic of a deployed application to the network in a form that can be reified by Maat. As such, IDN opens up a whole new set of opportunities in research and operational circles, and also creates some challenges. We now discuss some of these.

A. *Opportunities*

IDN opens up self-adaptation opportunities for all players in the network space: users, developers and service providers. Users benefit from improved QoE through service provisioning that is dynamic and adaptive to their requirements and contexts. Application developers gain access to higher programming primitives that facilitate fluid application behaviour at runtime, with less reliance on ad hoc means of connecting services and mitigating failures. Service providers are empowered to provision their services in a migration-ready form to be able to provide better QoE for their end users.

IDN also opens a market for hosting services towards the edge. This can be beneficial particularly for small and medium sized service providers who cannot afford a highly customized CDN presence like the Googles and Facebooks of the world. Instead, they would be able to bid for edge resource provisioning that in many parts of the world has a wider reach that traditional CDNs [3], [4].

B. *Challenges*

With the benefits that IDN brings, it also generates a number of challenges. The most prominent of these are *trust* – specifically: *security* and *efficiency* – and *deployment*.

The **security** challenge could be summarized by the following question: *Could the application trust the network to interfere with its communications, potentially redirecting it to an unintended destination?* This is indeed a major challenge that we recognize. We should first clarify that the in-network Maat agents receive and compile intents, not the subsequent communication which is more likely to contain sensitive information. Based on this, Maat would have information about the desires of the application such as connecting with peers, advertising services or content, regulating network traffic, etc. There is potentially a lot of risk in divulging such information to outside parties. It is noteworthy that such challenges are also being faced by the current Internet architecture.

The **efficiency** concern is summarized with a subtly different question: *Would the application trust the network to potentially impede or interfere with its performance?* Maat will have significant influence on where the application is redirected to serve its intent. As far as the application is concerned, Maat mediators are black boxes that might have interests conflicting with those of the application users. They could also be misconfigured, resulting in non-optimal mediation. We perceive this challenge to in fact be an opportunity for *auditing schemes* that ensure the efficiency of mediation. For this, we envisage regular reporting of intent, and resulting "mediation logs" that could be scrutinized to ascertain efficiency. In a multi-mediator market, the mediation score resulting from such auditing mechanisms would engender competition.

Another challenge relates to **deployment** and scalability. The core IDN design lends itself to partial deployment through independent rollout of Maat agents most likely at the edge. There are different ways of doing this, one of which is to augment wireless routers with additional modules. Such devices, however, are typically resource constrained and might suffer from performance issues if a large number of services are advertized on their local address spaces. One way of avoiding this is to deploy dedicated Maat agents instead of piggybacking on existing infrastructure. This comes with its own cost, but is feasible using commodity Linux boxes.

VI. Related Work

We review work that is relevant to our IDN concept, and briefly discuss some of the technologies that will enable the reification of intents in the network.

A. *x-Centric Approaches*

Bringing application awareness to networks has been a long sought after goal, with a number of technologies and network architectures being presented.

Resource-centric. The Representational State Transfer (REST) architectural principle [5] reduces network interactions to a few *verbs* (GET, POST, DELETE, etc.). REST professes an entirely stateless, resource-oriented approach, transitioning between states using data included in the requests. This makes infrastructure scalability and manageability easier. However, REST continues to adopt the "narrow" network API approach and, thus, continues to suffer from all of the associated problems discussed in §I.

Network-centric. Information-centric networking (ICN) solutions have been proposed to convert networks into inherent content delivery systems [6]. Similarly, service-centric networking (SCN) [7], [8], [9], [10], [11] extends ICN principles to apply to services as well as content. Both ICN and SCN attempt to align the application and the network, which helps to break away from statically binding to specific network resources. However, they only partly address the problems we have outlined in the specific cases of accessing content/services: they do not naturally generalize to other scenarios, *e.g.* those involving switching of networks.

Stakeholder-centric. In the Experience-oriented network architecture (EONA) [12], application and content providers as well as infrastructure operators exchange information from their respective control loops to improve user experience. We take inspiration from EONA, but are concerned about the viability of its approach. In a world where data is the new oil, we can not imagine such cooperative exchange of information happening between parties that ordinarily have conflicting interests [13]. The authors do not provide a reasonable argument for how this would be realized.

Policy-centric. Policy-Based Management (PBM) languages and tools have been well established for defining high level policies and refining such specification into actionable and quantifiable network-level targets [14]. PBM is typically constructed around rule-based, goal-driven, or event-driven principles that are mapped to specific operations. Literature includes a host of work on specification, management, and refinement of DiffServ policy hierarchies (*e.g.* [15], [16]) and enabling autonomic networking (*e.g.* [17], [18]). Other PBM work is also emerging under the 'network synthesis' subfield. Propane [19] compiles a network-wide BGP configuration from a high-level policy of forwarding and routing constraints, and has also been extended to adapt to abstract and evolving topologies [20]. SyNet [21] extends this further through translating Datalog-defined policies into a confluent set of OSPF and BGP forwarding rules as well as static routes.

Recent extensions to this philosophy include the RFC on autonomic networking [22], which defines some high level design goals of automated implementation of abstract operational goals *i.e.* intents. The RFC does not provide any indication of hot to implement or deploy this. A very recent paper [23] provides a solution to quantify soft goals associated with such abstract intents, and to use NFV chains to implement them. The work focuses on middlebox configuration and deployment, but does not discuss service discovery. IDN has more to offer, as it covers more than just soft high-level goals.

However, this work is largely about facilitating malleable network management that is driven by QoS objectives or business goals/constraints. A such, they are geared towards network operators and those dealing with wholesale traffic. They cannot, for instance, be used for facilitating application-defined opportunistic service binding at the edge.

Application-centric. Closer to our proposal are recent efforts in the direction of enabling applications to express their requirements and allowing these to percolate down to the underlying network. Pyretic [24] is an open source Python framework that raises the level of abstraction of writing network policies, enabling the definition of sophisticated network structures through a high-level language. Merlin [25] is another declarative language that enables the specification of a global networking policy, which is expressed as a collection of logical predicates to identify traffic subsets and a set of statements indicating the action(s) to be taken on each subset. Both Pyretic and Merlin focus on issues relating to unifying network administration rather than identifying and addressing application requirements.

Other relevant efforts include: yanc [26], a Linux abstraction to facilitate writing network control logic in any language; FlowOS [27], a programming model to capture and process Internet flows; NOSIX [28], an abstraction layer to enable portable deployments; and P4 [29], a language to configure switches to process packets and match header fields.

B. Enabling Technologies

The *Active networking* (AN) paradigm enables users to modify network behaviour by sending custom code to be executed on network devices (*cf.* [30]). *Software defined networking* (SDN) removes the need for bit-wise configuration of network components, and instead allows a central policy to be applied system-wide. Both AN and SDN technologies could be used to the benefit of users, but neither helps the application signify its needs. Thus, they are complementary to IDN by providing mechanisms to manage and modify the network in order to reify intents.

VII. Summary and Roadmap

We have proposed Intent Driven Networking (IDN), a concept in which applications and other players such as content providers formulate their communication-related 'intents' in high-level terms that get transformed into network-level reifications that better support the declared intents. We put forward a straw man design that specifies how intents might be formulated (§II-A), involving an ontology of verbs to signify various application desires (§II-B) and a recursive syntax that allows encapsulation of intents. Reification relies on the Maat system that provides in-network mediation between user intents on the one hand, and policies of network and service operators on the other (§II-C).

Apart from enjoying network service levels that better match their intents, applications also benefit from IDN in that some of their logic could be pushed to the network. No longer are they expected to ship with intricate conditional logic to work around unexpected network behaviour. (They still could employ such logic, but they would thereby be limiting their ability to be deployed in foreign environments and under unforeseen conditions.)

Maat also exploits the semantic content to optimize resource usage within the network. It does this by explicitly negotiating how each intent might best be reified to produce the most "friendly" way in which each network interaction can be performed, considering the needs of all stakeholders.

There is a huge body of future work required to develop IDN into a viable implementation. We capture some of the required next steps in a roadmap (Fig. 4). This, however, is undoubtedly a non-exhaustive plan of action. Therefore, we solicit contributions from the wider systems research community, architects and developers of different disciplines.

We see the process of developing a proof-of-concept realization of our design as falling into three main workpackages. First, we will work on the formalism of intent specification and its compilation into a format that can be used by Maat. This will involve refining the ontology presented here, defining a domain-specific language for the formulation of intents on the basis of this, and also the creation of associated developer tools. It will also involve investigating the semantics of the (recursive) composition of intents in terms of existing intents and ultimately in terms of primitive verbs. Semantics must take into account the effects of composing network functions that interact in complex ways: for example how caching strategies change when associated with mobility.

The second workpackage involves the definition of the negotiation protocol employed between Maat agents. This should be independent of any particular set of verbs and rely on generic notions of utility and priority as derived from intent specification, and be capable of handling negotiations between

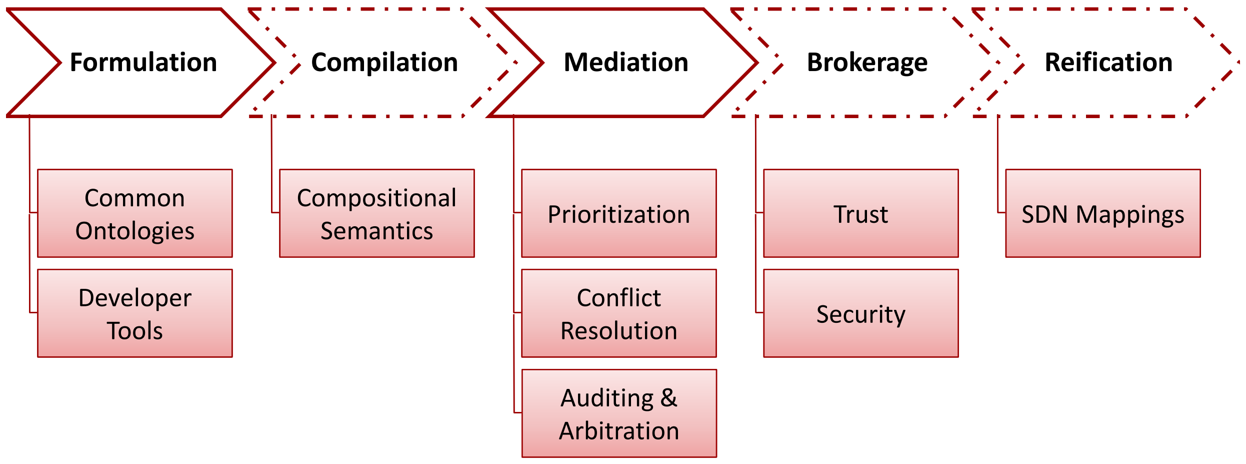

Fig. 4. A roadmap for the realization of IDN.

multiple stakeholders and converging on distributed consensus. It is clear that mediation is a highly complex task as it is likely that many conflicts will emerge. For example, a user streaming content would want high quality delivery at low cost, a publisher would wish to have their content viewed as many times as possible, and an ISP would prefer to have low-cost (locally available) content viewed. Such potentially conflicting stances will need to ensure thorough negotiation that all stakeholders are incentivized to cooperate.

The third workpackage would then focus on practical aspects of the deployment of Maat agents, as discussed in §V-B.

As a final consideration, marketplace brokerage is an area with a lot of potential for reifying spontaneous and strategic intent. Reification is likely to create the need for running in-network services towards the edge. Marketplaces of resources to host such services might benefit from the operation of brokerage and arbitrage agencies, a role which might be co-located with Maat. For this, thorough investigation is required to alleviate concerns regarding trust and security. Efforts are also sought for reifying mediation outcomes in the form of adjusting the network control plane (using SDN technologies) or providing information that could be used by applications for late-binding.

We hope that our IDN proposal will serve as an initial step towards a long-term research campaign that focuses on higher-layer needs of Internet stakeholders rather than forcing them into using fixed and constrained abstractions.

References

[1] E. Lee et al., "The swarm at the edge of the cloud," *IEEE Design & Test*, vol. 31, no. 3, 2014.
[2] Y. Elkhatib, B. F. Porter, H. B. Ribeiro, M. F. Zhani, J. Qadir, and E. Rivière, "On using micro-clouds to deliver the fog," *Internet Computing*, vol. 21, no. 2, pp. 8–15, March 2017.
[3] Y. Elkhatib, "Mapping Cross-Cloud Systems: Challenges and Opportunities," in *HotCloud*, 2016.
[4] R. Fanou, G. Tyson, P. Francois, and A. Sathiaseelan, "Pushing the frontier: Exploring the African web ecosystem," in *WWW*, 2016.
[5] R. T. Fielding, "Architectural Styles and the Design of Network-based Software Architectures," Ph.D. dissertation, University of California, Irvine, 2000.
[6] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. H. Briggs, and R. L. Braynard, "Networking named content," in *CoNEXT*, 2009.
[7] M. J. Freedman, M. Arye, P. Gopalan, S. Y. Ko, E. Nordstrom, J. Rexford, and D. Shue, "Service-centric networking with SCAFFOLD," http://www.dtic.mil/cgi-bin/GetTRDoc?Location=U2&doc=GetTRDoc.pdf&AD=ADA571380, Princeton University, Tech. Rep. 885-10, September 2010.
[8] T. Braun, A. Mauthe, and V. Siris, "Service-centric networking extensions," in *Symposium on Applied Computing*, 2013, pp. 583–590.
[9] D. Griffin, M. Rio, P. Simoens, P. Smet, F. Vandeputte, L. Vermoesen, D. Bursztynowski, and F. Schamel, "Service oriented networking," in *European Conf. on Networks and Communications*, 2014.
[10] C. Tschudin and M. Sifalakis, "Named functions and cached computations," in *CCNC*, 2014, pp. 851–857.
[11] A. Sathiaseelan, L. Wang, A. Aucinas, G. Tyson, and J. Crowcroft, "SCANDEX: Service centric networking for challenged decentralised networks," in *DIYNetworking*, 2015, pp. 15–20.
[12] J. Jiang, X. Liu, V. Sekar, I. Stoica, and H. Zhang, "Eona: Experience-oriented network architecture," in *HotNets*, 2014, pp. 11:1–11:7.
[13] D. D. Clark, S. Bauer, W. Lehr, K. C. Claffy, A. D. Dhamdhere, B. Huffaker, and M. Luckie, "Measurement and analysis of internet interconnection and congestion," in *Telecomm. Policy Research*, 2014.
[14] R. Boutaba and I. Aib, "Policy-based management: A historical perspective," *Journal of Network and Systems Management*, vol. 15, no. 4, pp. 447–480, 2007. [Online]. Available: http://dx.doi.org/10.1007/s10922-007-9083-8
[15] R. Rajan, D. Verma, S. Kamat, E. Felstaine, and S. Herzog, "A policy framework for integrated and differentiated services in the internet," *IEEE Network*, vol. 13, no. 5, pp. 36–41, Sep 1999.
[16] A. K. Bandara, E. C. Lupu, A. Russo, N. Dulay, M. Sloman, P. Flegkas, M. Charalambides, and G. Pavlou, "Policy refinement for ip differentiated services quality of service management," *IEEE Transactions on Network and Service Management*, vol. 3, no. 2, pp. 2–13, April 2006.
[17] B. Jennings, S. V. D. Meer, S. Balasubramaniam, D. Botvich, M. O. Foghlu, W. Donnelly, and J. Strassner, "Towards autonomic management of communications networks," *IEEE Communications Magazine*, vol. 45, no. 10, pp. 112–121, October 2007.
[18] F. Meyer and R. Kroeger, "A framework for autonomic, ontology-based it management," in *11th International Conference on Network and Service Management (CNSM)*, Nov 2015, pp. 78–84.
[19] R. Beckett, R. Mahajan, T. Millstein, J. Padhye, and D. Walker, "Don't mind the gap: Bridging network-wide objectives and device-level configurations," in *Proceedings of the ACM SIGCOMM Conference*. ACM, 2016, pp. 328–341.
[20] ——, "Network configuration synthesis with abstract topologies," in *Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*. ACM, June 2017, pp. 437–451.
[21] A. El-Hassany, P. Tsankov, L. Vanbever, and M. T. Vechev, "Network-wide configuration synthesis," in *Proceedings of the International Conference on Computer-Aided Verification (CAV)*, July 2017.

[22] M. H. Behringer, M. Pritikin, S. Bjarnason, A. Clemm, B. Carpenter, S. Jiang, and L. Ciavaglia, "Autonomic Networking: Definitions and Design Goals," RFC 7575 (Informational), RFC Editor, Fremont, CA, USA, pp. 1–16, June 2015. [Online]. Available: https://www.rfc-editor.org/rfc/rfc7575.txt

[23] E. J. Scheid, C. C. Machado, M. Franco, R. L. dos Santos, R. Pfitscher, A. Schaeffer-Filho, and L. Z. Granville, "INSpIRE: Integrated NFV-baSed Intent Refinement Environment," in *Proceedings of the 16th IFIP/IEEE International Symposium on Integrated Network Management (IM 2017)*. IEEE, May 2017.

[24] J. Reich, C. Monsanto, N. Foster, J. Rexford, and D. Walker, "Modular SDN programming with Pyretic," *Technical Report of USENIX*, 2013.

[25] R. Soulé, S. Basu, R. Kleinberg, E. G. Sirer, and N. Foster, "Managing the network with merlin," in *HotNets*, 2013, pp. 24:1–24:7.

[26] M. Monaco, O. Michel, and E. Keller, "Applying operating system principles to SDN controller design," in *HotNets*, 2013, pp. 2:1–2:7.

[27] M. Bezahaf, A. Alim, and L. Mathy, "FlowOS: A flow-based platform for middleboxes," in *HotMiddlebox*, 2013, pp. 19–24.

[28] M. Yu, A. Wundsam, and M. Raju, "NOSIX: A light- weight portability layer for the SDN OS," *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 2, pp. 28–35, 2014.

[29] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, and D. Walker, "P4: Programming protocol-independent packet processors," *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 3, pp. 87–95, 2014.

[30] D. Tennenhouse, J. Smith, W. Sincoskie, D. Wetherall, and G. Minden, "A survey of active network research," *IEEE Communication Magazine*, vol. 35, no. 1, pp. 80–86, 1997.