SCANDEX: Service Centric Networking for Challenged Decentralised Networks

Arjuna Sathiaseelan
Computer Laboratory
University of Cambridge
Cambridge, UK
first.last@cl.cam.ac.uk

Liang Wang
Computer Laboratory
University of Cambridge
Cambridge, UK
first.last@cl.cam.ac.uk

Andrius Aucinas
Computer Laboratory
University of Cambridge
Cambridge, UK
first.last@cl.cam.ac.uk

Gareth Tyson
School of EECS
Queen Mary University
London, UK
first.last@eecs.qmul.ac.uk

Jon Crowcroft
Computer Laboratory
University of Cambridge
Cambridge, UK
first.last@cl.cam.ac.uk

ABSTRACT

Do-It-Yourself (DIY) networks are decentralised networks built by an (often) amateur community. As DIY networks do not rely on the need for backhaul Internet connectivity, these networks are mostly a mix of both offline and online networks. Although DIY networks have their own homegrown services, the current Internet-based cloud services are often useful, and access to some services could be beneficial to the community. Considering that most DIY networks have challenged Internet connectivity, migrating current service virtualisation instances could face great challenges. Service Centric Networking (SCN) has been recently proposed as a potential solution to managing services more efficiently using Information Centric Networking (ICN) principles. In this position paper, we present our arguments for the need for a resilient SCN architecture, propose a strawman SCN architecture that combines multiple transmission technologies for providing resilient SCN in challenged DIY networks and, finally, identify key challenges that need to be explored further to realise the full potential of our architecture.

Categories and Subject Descriptors

C.2.1 [**Network Architecture and Design**]: Network Communications

Keywords

Information Centric Networking, Service Migration

1. INTRODUCTION

Recent years have seen an increase in localised communication paradigms, e.g., community networks, do-it-yourself (DIY) networks and mobile ad hoc networks. Such networks do not rely on the need for managed backhaul Internet connectivity and, hence, they are a mix of both online and offline (i.e., they can be both connected and disconnected from the wider Internet). Although such decentralised networks can have their own homegrown services, it is fair to say that current Internet-based cloud services could be beneficial to many communities. Clearly, disconnected offline networks would find it impossible to access such remote cloud services though.

The above suggests that building technologies that support easy hosting and migration of local services could be highly beneficial in DIY networks. If remote cloud services could be migrated within individual DIY setups, then users could seamlessly use them without requiring consistent egress connectivity. This, however, is not trivial. Key questions include: (1) Which services should be migrated? (2) When should they be migrated? (3) Can these services continue to operate without remote connectivity? (4) How should state be managed? The difficulty of addressing these questions is exacerbated by the unpredictable nature of DIY networks, which may frequently move between online and offline statuses.

The advent of virtualisation technologies, especially low cost unikernels, offers an interesting avenue of exploration in this space. Unikernel architectures [1] allow very small (e.g., 2MB) virtual machine services to be migrated across a network with very little cost. Such capabilities are ideal for DIY networks, which may struggle to host and transport larger virtual machines. This paper argues that the integration of services with DIY networks, underpinned by a unikernel architecture, could open up a whole new communications paradigm, enabling low cost DIY networks to become far more powerful in their capabilities. Although attractive, the above principles bring many challenges. Whereas transporting unikernel-based services within a datacenter is trivial, transporting them across unpredictable and unstable networked environments is far more difficult. Even if the service is migrated successfully, many existing services are online-only (e.g., Google Maps) and therefore would struggle to operate once subsequent egress connectivity fails.

.

Service Centric Networking (SCN) [6] has recently been proposed as a potential solution for deploying and managing networked services. It has largely emerged from the Information Centric Networking (ICN) community [9]. SCN decouples the service from their origin location, removing the need for the current end-to-end client server model. This means that the service and/or content can be served directly by any host that currently has the service/content. Current SCN architectures do not yet offer the necessary underlying support to operate in challenged scenarios (e.g., where a DIY network loses egress connectivity). Instead, SCN solutions focus mainly on well-connected environments. Although mobility has been partially addressed by certain ICN solutions [5], they still do not offer resilience during periods of total disconnectivity [13]. As such, we argue that SCN principles should be extended to better support the unique properties of DIY networks.

This position paper proposes SCANDEX (SCNx)[1] — a strawman architecture (c.f. Fig. 1) specifically designed to support services in challenged DIY community networks. Unlike previous SCN designs, we focus on supporting situations where networks may have unpredictable and intermittent connectivity. This extends to both intra- and inter-network connectivity. We underpin SCNx with unikernel-based services that can be seamlessly migrated, cached and executed across (theoretically) all devices within the network. Rather than hosting services in fixed predetermined locations, SCNx allows them to be hosted *anywhere*. To enable this, we borrow principles from the ICN, SCN and Delay Tolerant Networking (DTN) [8] communities. A name-based routing scheme allows clients to access any nearby copy of the service, which may be cached and replicated at will. Importantly, we integrate delay tolerant techniques to allow services to be passed through the network in a store-and-forward fashion, thereby mitigating the problems brought about by communication failures. We believe that this offers a positive first step towards bringing popular services to DIY community networks.

2. BACKGROUND

An SCN is a network that is explicitly built around the concept of services [6]. Any node can offer services to the network, and any (authorised) node can consume them. Unlike more traditional service oriented architectures (SOAs), the network is inherently involved in the process by allowing services to "migrate" to any network location and exist as a self-contained unit (e.g., within a router or middlebox).

The principles of SCN have emerged largely from the Information Centric Networking (ICN) community. ICN fundamentally changes the communications API by allowing nodes to request unique content objects, identified using unique names. The network is then solely responsible for returning the requested object. By decoupling content from specific locations, it becomes possible to easily cache these atoms within the network. Several ICN projects ([9, 10, 11]) have advocated redesigning the Internet based on these principles. In essence, such projects propose name-based routing/resolution schemes that allow clients to discover (cached) copies of content, thereby returning the object in the "optimal" manner. SCN follows similar principles, but, instead, allows *services* to be discovered and accessed independent of their location. It has been noted that these properties are very promising in challenged environments [12, 13]. For example, by caching a service near to the user, it may become possible for the user to access it locally when backhaul Internet connectivity is not available (e.g., after an earthquake).

As of yet, however, mainstream SCN implementations struggle in more challenged environments (where connectivity at both the front and backhaul are intermittent or disrupted for a period of time), as global synchronous connectivity needs to be available. For example, if a service were hosted in a location where no end-to-end path exists between it and the consumer, a request would simply fail. SCN is therefore only beneficial in challenged environments when an accessible cached copy of the service exists. In contrast, DTN addresses this concern by supporting disruption and discontinuity in end-to-end paths [8]. Rather than relying on contemporaneous connectivity on all segments of an end-to-end path (as IP networks do), DTN operates in a store-and-forward fashion. Intermediate nodes assume temporary responsibility for messages and keep them until an opportunity arises to forward them to the next hop. This inherently deals with temporary disconnections or disruptions, allowing service instances to be progressively passed through the network hop-by-hop until reaching their consumers.

We have previously proposed an architecture that combines IP, ICN and DTN into a single unified architecture, allowing content to be accessed via any available technology [3]. Here, we build on this work to also support the migration of services (as well as content). Such services are modelled as unikernels: small VMs dedicated to executing a single role. Alternative methods, such as Linux containers [4], could also be used to host services. Much like ICN supports the caching of content as a independent unit, we propose to make service instances (unikernels) entirely mobile and cacheable. This is underpinned by a name-based routing strategy (similar to ICN), that allows local services to be easily discovered and accessed. Through this, we intend to allow DIY networks, that may suffer periods of offline disconnection, to utilise local services that otherwise would become unavailable. Importantly, we integrate DTN capabilities, allowing services to be opportunistically distributed through the network in a hop-by-hop fashion. This is particularly critical in DIY networks, where the assumption of constant egress connectivity is often false.

3. SCANDEX (SCNX)

3.1 Overview

We propose SCANDEX (SCNx), a strawman SCN architecture specifically designed for challenged decentralised networks. It allows services to exist in the network as self contained objects (similar to information objects in ICN). Each service is uniquely identified and can be, thus, cached (and accessed) from any location. Clients contact services by issuing a service request containing the service's unique identifier, alongside any input parameters (c.f., Section 4). Services, themselves, can be in one of two states: stored and instantiated. A stored state is not running and is, instead, within memory as a static object; an instantiated service is one that is running and able to accept service execution requests. In SCNx, individual services are actually unikernel

[1]Akin to Spandex, our architecture allows flexibility for operating across diverse challenged environments.

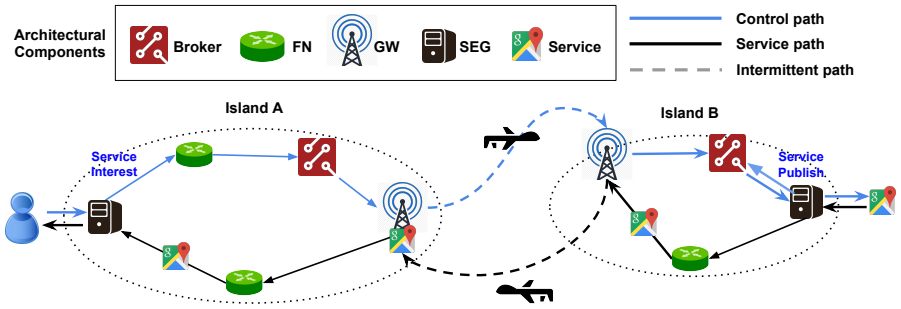

Figure 1: Example SCNx network.

VMs [1], which are produced by *Publishers* and requested by *Subscribers*. Subscriptions are performed by consumers sending *Interest* messages that request a given service.

An SCNx network consists of four key components as following. Note the components are logical hence can be multiplexed on the same physical nodes.

1. *Service Execution Gateways* (SEG): These are the points of attachment for clients and servers, and could exist as wireless access points or base stations. A SEG also hosts and executes services on behalf of its attached clients.
2. *Forwarding Nodes* (FN): These are are responsible for routing requests for services towards available copies. These nodes also cache services locally, thereby allowing local copies to be returned to the client (they do not execute services though).
3. *Edge Gateways* (GW): These are responsible for connecting different domains (two separate networks). Gateways can also temporarily function as Publishers and Subscribers of services, i.e., a proxy to request services from another network on behalf of one of its users.
4. *Brokers*: These perform the service resolution and are also responsible for performing intradomain forwarding. Every device (SEG, FN, GW) that wants to be part of the SCNx network within a domain should register with a broker and should be reachable by the broker. Reachability can be either through a direct path between the broker and the device, or through hop-by-hop store-and-forward if the path is intermittent (i.e., DTN). The brokers could use distributed routing protocols such as OSPF to discover the network topology. Brokers between different domains specifically exchange Interest messages using scoped flooding. Brokers could also exchange service resolution information. Brokers can reside in a standalone device or within a SEG, FN and/or Edge Gateways.

A unique feature of SCNx is that it integrates many different technologies together as *transmission strategies*. We define a transmission strategy as being a mechanism by which any objects containing control or data information can be transported from one location to another. For example, based on the environment of a DIY network, SCNx could transport an instance of a requested service or an Interest via IP, DTN or other source routing methods (e.g., Bloom filters), if supported. Through this, SCNx adapts the DIY network to use the most appropriate communications technologies for its specific needs (c.f., Section 4).

3.2 Example of Operation

The operation of SCNx is best explained via an example (Figure 1). Imagine a disaster area, where an earthquake has damaged local communications infrastructure. Several local DIY networks have quickly emerged, operating using ad hoc wireless links. The regional authorities have invested in a Unmanned Aerial Vehicle (UAV) that flies along predetermined paths over the disaster zone. When the UAV comes into wireless contact with a DIY network, it can temporarily communicate. The UAV may, or may not, have backhaul Internet connectivity. The UAV and the DIY networks all operate SCNx. For simplicity, all nodes on an intradomain level utilise the same communications technologies; we assume a (relatively) well connected setup that might, for example, be connected by classic ad hoc protocols such as AODV.

We highlight the key components of SCNx using an example request. A client node in Island A wishes to access a service that is located in Island B. Each client is attached to its local DIY network via a SEG. A service request is therefore initially passed from the client to the SEG (i.e., stipulating the name of the desired service). If the SEG possesses a copy of the service locally in its cache, the request is simply passed to the local service instance and it is executed immediately (note that execution occurs on the SEG). If, however, the SEG does not have a local copy, the request must be forwarded on. The SEG translates the request to a service Interest (request) using the SCNx naming format later discussed in Section 4.

Interests are then passed through the network via FNs to a Broker. Each DIY network must contain at least one Broker. These are responsible for indexing *all* services that are locally available in the DIY network (intradomain). This means all SEGs that host services must register them with the Broker. If the Broker is aware of an intradomain instance of the service, the broker contacts the device that has the instance and requests that it migrates the service to the SEG using a specific transmission module that is suitable for the underlying network topology. If the underlying network has stable connectivity, this transmission strategy could be carried out using standard IP or using a source routing method (e.g., Bloom filters). If the underlying network is challenged, then the transmission strategy would be a DTN (using the Bundle Protocol [8] or other DTN options). Once the service instance is migrated, the SEG then instantiates the migrated instance and serves it to its clients.

If the Broker does not know of any copies of the service within its own network, it is necessary to forward the Inter-

est on an interdomain level via the GW. Before forwarding the Interest, the broker records the Interest as pending (denoting this is yet unresolved). Interdomain communications are far more challenging than intradomain communications because different network islands may utilise different technologies that may even vary over time based on conditions. Most problematic is the intermittent connectivity that may make the distribution of state information difficult. To address this, the Broker floods its neighbouring networks with the Interest. The exact mechanism by which the neighbours are contacted will vary based on their capabilities. We envisage two key possibilities:

- Synchronous: If the Broker has full synchronous connectivity (e.g., IP) to other network islands and/or the UAV, it will directly contact them. It will forward the Interest and wait for the response.
- DTN: If the Broker only has intermittent connectivity to other network islands and/or the UAV, it will temporarily store the Interest until another network/UAV comes into communications range. Based on the TTL, these networks/UAVs may forward the Interest further to other parties. Most notably, the UAV would carry the Interest and forward it when it comes into contact with Island B.

Either of the above transmission strategies will result in the Interest being passed into Island B. It will enter via Island's GW; this node will then take responsibility for trying to obtain the service from its local area. To achieve this, the GW will pass the Interest to its network's Broker, requesting the service. As the service is located in Island B, the Broker discovers the service and instructs the host to pass it to the GW. The GW in B now has the migrated instance locally in its cache. The Broker in Island B instructs the GW to forward the service instance using a DTN transmission strategy to the UAV. The GW now forwards the service instance using a DTN transmission strategy via the UAV reaching the GW in island A. When the GW in Island A receives the service instance, it caches it and then publishes this to its own broker. The broker upon receiving the Publish message, recognises there is a pending Interest, and instructs the GW to forward the service instance to the actual subscriber using intradomain forwarding. The SEG on receipt of the service instance, instantiates the service and serves the clients locally over standard IP.

4. KEY CHALLENGES

In this section, we briefly summarise a list of key challenges that needs to be explored further to realise the full potential of our architecture.

4.1 Service Caching and Synchronization

SCNx caches services within the network. By deploying the services to SEGs as close to clients as possible, the services are made available during periods of disconnection from the wide area network. Although service popularity follows a highly skewed distribution, it is unlikely that we can run all the services simultaneously at the SEG given the limits of physical resources. To assist in this, the forwarding nodes (FN) become storage nodes that cache the small unikernels (yet do no execute them).

Various caching algorithms can be implemented to achieve different goals. For example, the most popular services could be moved to the edge to reduce service latency and network

```
GET /api/0.6/map?bbox=11.54,48.14,
    11.543,48.145 HTTP/1.1
Host: api.openstreetmap.org
```

Figure 2: An example RESTful API request for a map within a bounding box.

use. If the SEG does not have sufficient capacity for the service, we can use LRU/LFU to evict the less popular ones. If there are other copies in the network, the SEG can simply drop the evicted service. If the evicted service is the only copy registered in the Broker (i.e., the only copy in the network), instead of dropping the service completely, we can push the evicted services to an upstream FN. The SEG can then inform the Broker about the eviction and the location of the evicted service.

Multiple instances of a service can also exist within the network, which introduces complexity but improves service availability. If multiple instances exist within a SCNx network, the Broker maintains a list of all available copies. If a service is evicted out of the cache, the Broker unregisters the service and updates the information by flooding. The broker should pick which service instance it should forward the request to avoid duplicate responses. By allowing stateful services and mutable content, however, we are confronted with the state synchronisation problem of the services, i.e., it is necessary to provide services with the means of merging their state. The merge semantics are service-dependent and should not be implemented within the native SCNx architecture. Instead, the services themselves need to be designed to incorporate state updates from each other. One option of doing so is to provide a version control mechanism for state synchronisation [14], where each application decides how to resolve state conflicts.

For stateless services, multiple copies usually only introduce negligible storage overhead. Comparing the significant improvement on the system robustness, service availability and latency reduction, the overall cost is marginal. However, for stateful services, proper synchronization mechanisms must be implemented to guarantee the different service copies have a consistent view of the service state. Obviously, we need to balance the trade-off between synchronization overhead and latency reduction.

4.2 Service Representation and Registration

One of the key strengths of SCNx is the interoperability with legacy clients and services through the use of the SEG. A client can simply interact with the SEG via a well known protocol (HTTP), leaving the SEG to translate wider interactions into SCNx. This is possible because there is a direct mapping between the hierarchical RESTful API style addresses (Fig. 2) and SCNx service names. As RESTful API calls can contain parameters both within the URL and the body of the request, we only map the URL onto the SCNx service name and put the request content into the body of the Interest message. The representation does not distinguish static content from services since everything is considered as a service. For example, raw static content is wrapped into a service publishing files in response to subscriptions. This generalisation therefore incorporates the efficient large content distribution as that in various ICN proposals. If a user requests a map within a certain bounding box from the Open Street Map, the coordinates of the location (*bbox*

parameter) are embedded into a request which is used by a SEG to construct a service Interest.

Service registration is managed by the SEG. As it controls when to instantiate or evict a service, it announces to the Brokers every service it has instantiated by flooding an Interest message to all Brokers within its own domain. The registration information is a tuple $< name, loc >$ which contains both the identifier and locator(s) of a service. The *name* of a service is simply its SCNx name, whereas the *loc* is a list of locators for every potential underlying transmission strategy. The reason is that each transmission strategy has a well-defined locator scheme. In order to incorporate the flexibility in routing and guarantee the reachability of the service, the broker needs to know how to describe the service location in every transmission strategy. For instance, the locator in IP transmission strategy can be defined as $IP : PORT$ combination; the one in LIPSIN or Bundle Protocol can be defined as node's *id*; the one in DHT can be the hash value of *id*; etc. A key challenge is therefore managing this complexity, which is non-trivial. In a challenged and dynamic environment, state management can be costly; it is therefore important to develop techniques that can manage this large body of dynamic state in a low overhead and reliable manner.

Service discovery itself is also an important topic that needs further investigation: users could potentially utilise conventional methods such as search engines to discover various services. However, such services themselves may become inaccessible when the network is isolated. In such cases, in-network service discovery can be implemented; by querying the available Brokers, a list of available services could be returned using a well-defined format, e.g., Web Service Definition Language (WSDL). A key challenge is that this might violate the need to interoperate with legacy clients. That said, some lightweight service directories can be implemented as default services constantly residing within the network to provide context-based content provision and recommendation.

4.3 Distributed Authentication

The authentication requirements of SCNx are reduced by using self-contained service VMs (unikernels) as the basic representation. Implementation details together with any security credentials stored within the VM are not directly accessible to any other nodes or services due to the strong isolation offered by the VM hypervisor running on a SEG. Authentication between the user and the service is therefore specific to each service and can use a variety of current methods, such as password or cookies.

Authentication of individual requests and responses becomes more challenging when they traverse the boundaries between the legacy and the SCNx networks. Subscribe messages must not be encrypted for correct routing to publishers, and, of course messages may traverse multiple hops in the network which may not be trusted. End-to-end encryption commonly used today (e.g., TLS) is therefore not viable.

One possible approach (due to the small service footprint and strong isolation) is to require each SEG to run a corresponding authentication *sub-service* for each service it serves or forwards requests for. This sub-service can then be used for RESTful to SCNx request translation: it would terminate the encrypted connection using service keys, sign and potentially encrypt the contents of the newly generated subscription message, but provide the hierarchical service name for the SEG to use for resolving the service. This, although viable, does, however, introduce extra overheads, while also assuming the authentication sub-service is entirely trusted.

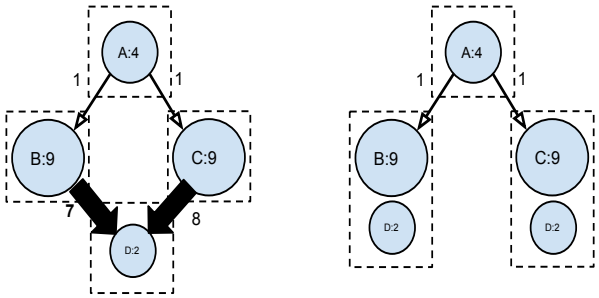

Figure 3: Service dependency and migration strategies

4.4 Managing Service Dependencies

Authentication is just one example where a service is a combination of multiple, smaller sub-services. This introduces new challenges, as it becomes necessary for service dependencies to be resolved. Such dependencies can be described using a directed acyclic graph (DAG), used commonly in Service-Oriented Architectures (SOA). In existing proposals [6], the service dependency is described as an aggregation of multiple independent chains, which are required to be explicitly embedded in the header of an Interest. In SCNx, a user only needs to interact with the root of a service DAG. The services themselves initiate communication with other sub-services they depend on.

Figure 3 presents a simple example. Service A is the requested root service, which depends on the outputs from B and C. Meanwhile, both B and C rely on service D. Four dashed squares represent four SEG nodes. The weight of a node represents the cost of running it and the weight of an arrow represents traffic cost between communicating services. In this example, the amount of traffic between A and B (also C and D) is small, but the cost of running both of these services may be too high for the same SEG. At the same time, there is a considerable amount of traffic from both B and C to D. By duplicating service D at both nodes where B and C reside, we can significantly reduce traffic footprint. However, if there is negligible traffic between B and D (also C and D), the weight of the corresponding edges are small. As such, the service will be executed remotely. In this case, duplicating the service D only introduces more traffic overhead comparing to transmitting the service results.

Of course, this is only a toy example and real-world scenarios would be far more challenging; managing the placement and execution of these dependent services will be a key line of future work. For example, measuring, modelling and predicting network cost is not a trivial operation. It could be addressed in a centralised way at the Broker, however, this requires the broker to not only have the knowledge of all service dependencies, but also their costs. Though such information can be obtained by sampling and sharing their footprint, it adds further overheads and additional state synchronisation. Another option would be for each node to independently estimate the costs of its service dependencies and manage them autonomously without assistance from the Broker. As described above, the SEG makes the decision to

run a service locally or to forward each service request to a corresponding node elsewhere in the network. The explicitly recorded cost together with the DAG of dependent sub-services forms the basis for a SEG to decide whether to migrate the service and if yes – which parts of it.

4.5 Transmission Strategy Selection

SCNx incorporates multiple transmission strategies into the architecture, hence the Broker needs to select the best strategy for both delivering the control and data information objects. From the users' perspective, such decisions should be completely transparent. Although the underlying topology (whether it's fixed or mobile) does not change often, intelligent in-network decision making requires the SCNx network to be context-aware and event-driven so that it can respond quickly to the changes in network conditions, and adjust the strategy adaptively. Such a goal is achieved by exchanging the information among the Brokers in the control plane. Whenever a network event happens, e.g., node failure, link failure or network partition, the event will be detected by the nearby nodes and the update will be propagated to every broker within the network domain. Therefore, the brokers can have a consistent view of the network condition to help them in deciding proper transmission strategies. Although seemingly straightforward, this is actually quite challenging, particularly in highly dynamic environments where changes may occur often. In the worst-case scenario, the optimal transmission strategy may change moments after one has already been initiated. Therefore, the stability of the protocol should also be taken into account to prevent the system from oscillating between different transmission strategies quickly.

5. CONCLUSION

Migrating current service virtualisation instances over decentralised networks is a challenge. In this paper, we have discussed this problem, and considered SCN as a solution. Although SCN, as a concept, has existed for years, we extend it to explicit consider the needs of challenged DIY networks. We have proposed SCANDEX (SCNx), a strawman SCN architecture that combines multiple transmission technologies such as IP and DTN for providing resilient SCN in challenged DIY networks.

Intentionally, we have given a broad overview of the topic. Our future work centres on building up these principles with the goal of deploying SCNx. To this end, we have explored several key challenges that we will have to face. A common thread running through these is that of *service management*. In essence, SCNx wishes to integrate the management of services into the network as an explicit responsibility (much like packet forwarding is today). A key question, however, is whether or not network architectures are ready for such an expansion in their role. For example, delivering, storing and executing services can have many wider implications for network design, particularly given the extra overheads involved. A key line of future work is therefore understanding and controlling these implications. We aim to begin by building models that can capture the needs of services, such that decision making entities (e.g., Brokers and SEGs) can properly utilise this contextual data. Most prominently, it becomes necessary to decide when (and where) a service should be stored, and when it should be discarded. This is a non-trivial optimisation that will involve the conflicting needs of many parties. Another key challenge we face is design of data synchronisation mechanisms that can allow stateful services to operate in environments where disconnections are the norm. Of course, in all cases this dynamism has an impact on the underlying service routing and resolution protocols as well: control messages must be distributed, just as routing control messages are disseminated in IP. Potentially, SCNx faces a much greater burden though, as the number of services is likely to far exceed the number of nodes IP deals with. We also have a long way to go before we can fully exploit the architectural potential of things like dynamic transmission strategy selection. Although highly attractive, it remains to be seen how effectively a network can manage its own context information to inform these decisions. Despite these challenges, we argue that the advantages of using SCN principles in DIY networks is huge. Consequently, we feel our approach is well founded and is worthwhile pursuing.

Acknowledgements

The work has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 645124.

6. REFERENCES

[1] A. Madhavapeddy, D. Scott, Unikernels: The Rise of the Virtual Library Operating System, Communications of the ACM, January 2014.

[2] G. Xylomenos et al,, A Survey of Information-Centric Networking Research, IEEE Communications Surveys and Tutorials, May 2014.

[3] A. Sathiaseelan et al, An Internet Architecture for the Challenged, IAB ITAT Workshop, December 2013.

[4] J. Hadley, Y. Elkhatib et al, MultiBox: Lightweight Containers for Multi-cloud Deployments, EGC Workshop, November 2015.

[5] G.Tyson et al. A survey of mobility in information-centric networks, Communications of the ACM, December 2013.

[6] T. Braun et al, Service-Centric Networking, IEEE ICC, December 2011.

[7] B.H. Bloom, Space/time trade-offs in hash coding with allowable errors, Communications of the ACM, July 1970.

[8] K. Fall, A Delay-Tolerant Network Architecture for Challenged Internets, IRB-TR-03-003, February 2003

[9] V. Jacobson et al, Networking Named Content, ACM CoNEXT, December 2009.

[10] D. Trossen, G. Parisis, Designing and Realizing an Information-Centric Internet, IEEE Communications Magazine, July 2012.

[11] T. Koponen et al, A Data-Oriented (and Beyond) Network Architecture, ACM SIGCOMM, October 2007.

[12] G Tyson et al, Towards an information-centric delay-tolerant network, IEEE INFOCOM NOMEN, April 2013.

[13] G. Tyson et al, Beyond Content Delivery: Can ICNs Help Emergency Scenarios?, IEEE Network, June 2014.

[14] B. Farinie et al, Mergeable Persistent Data Structures, JFLA 2015.