Design of a Protocol to Enable Economic Transactions for Network Services

Xinming Chen*, Tilman Wolf*, Jim Griffioen†, Onur Ascigil†, Rudra Dutta‡, George Rouskas‡, Shireesh Bhat‡, Ilya Baldin§, and Ken Calvert†
*Department of Electrical and Computer Engineering, University of Massachusetts, Amherst, MA, USA
†Computer Science Department, University of Kentucky, Lexington, KY, USA
‡Department of Computer Science, North Carolina State University, Raleigh, NC, USA
§RENCI, University of North Carolina, Chapel Hill, NC, USA

Abstract—**Deployment of innovative new networking services requires support by network providers. Since economic motivation plays an important role for network providers, it is critical that a network architecture intrinsically considers economic relationships. We present the design of a protocol that associates access to network services with economic contracts. We show how this protocol can be realized in fundamentally different ways, using out-of-band signaling and in-band signaling, based on two different prototype implementations. We present results that show the effectiveness of the proposed protocol and thus demonstrate a first step toward realizing an economy plane for the Internet.**

I. Introduction

A key problem in deploying innovative features in the network core is that many protocols and services need support by providers throughout the network. Since network operators are justifiably driven by business goals, there need to be clear incentives to support new network features. The need for the *network architecture* to associate innovation with economic motivation is reflected in the work by Clark et al. [7] that emphasizes the importance of tying real-world tussles to the network architecture. An example of misalignment between protocol design and economic motivation is multicast, which has experienced limited deployment in the current Internet [8].

To address these challenges and expose economic tussles within the architecture, an "economy plane", complementing the data and control planes, has been proposed for the Internet [16], [17]. This economy plane, called ChoiceNet, enables entities (e.g., users or their applications, providers, etc.) to dynamically set up fine-grained, short-term economic contracts for network services. These network services are offered and sold through marketplaces and can range from simple connectivity (à la pathlets [10]) to complex processing and storage services (e.g., caching for NDN [12]).

While the principles and architecture of an economy plane for the Internet have been described in [16], [17], there has been no implementation and evaluation of protocols and prototypes to enable such functionality. In this paper, we provide these insights. In particular, we present a protocol design that considers economic relationships as an integral part of communication. The contributions of our paper are:

- Design of a service access protocol based on the establishment of economic relationships between entities.
- Design of economy plane functionality using two fundamentally different approaches, one based on out-of-band signaling and one based on in-band signaling, highlighting the versatility of the proposed protocol.
- Results from prototype implementations of both types of approaches.

The remainder of the paper describes the ChoiceNet architecture, the design of a protocol to relate economy plane interactions with data and control plane operations, and two implementations on the GENI platform.

II. ChoiceNet Architecture

The principle idea of ChoiceNet [16], [17] is to enable an explicit representation of economic relationships between entities in the network. In the current Internet, these relationships are based on long-term, "paper-based" service agreements. In ChoiceNet, the economy plane enables automated contracts for network services at various time scales.

The goal of dynamic contracts is to enable market-based competition among providers of network services, which improves quality of offerings and reduces cost to customers. To enable a dynamic and competitive market, ChoiceNet is based on three components:

Services. Network services represent any functionality that can be provided in a network ranging from connectivity between two end-points (e.g., pathlet) to complex data storage (e.g., caching) or lookup (e.g., DNS) services. In order to create a competitive market for services, it is necessary to specify the semantics of services such that service offerings can be compared.

Contracts. Contracts relate economic exchanges (e.g., payments) with operations within the network (e.g., access to a service). To be effective, contracts require enforcement. Thus, a customer needs to be able to verify that a service has been rendered to specification (e.g., as discussed in [3]) and a provider needs to be able to perform access control to limit services to those customers who have established economic relationships. The latter is one aspect of this paper.

Marketplaces. Marketplaces provide functionality to match provider offerings with customer requests. These marketplaces also may act as trusted intermediaries for economic transactions and support service composition (e.g., as discussed in [9]).

The steps taken to set up connections (or more complex service offerings) in ChoiceNet are: (1) Providers advertise their services in one or more marketplaces. (2) An end-system application (e.g., movie streaming app) queries the marketplace for available service offerings (e.g., QoS pipes, cached content). (3) The user (or a delegated entity, such as the operating system) makes a decision on which service to "purchase". (4) The providers involved in the service offerings set up their services in return for "consideration". (5) The end-system application uses the provided service.

A key challenge in this context is to connect the economic relationship among entities to the network services offered/purchased. In this paper, we describe a protocol that establishes this connection, and we describe two prototype system implementations that illustrate specific instantiations of these mechanisms.

III. Network Services

At the heart of the ChoiceNet architecture is the concept of a *network layer service*, the ability to compose services together, and to choose among available services. Although composed services have been explored in other contexts before [6], [11], past work has focused on the problem of integrating functionality, rather than that of compensating the operators of those services. A ChoiceNet network layer service not only needs to define "what the service does" so that it can be used/composed, but it must also specify "what a user of the service must do to compensate the provider of the service."

Here, we present our network layer service abstraction for ChoiceNet, describe how it can be composed to form complex/tailored services, and how we use our ChoiceNet protocol for implementing the selection mechanism.

A. Consideration

In ChoiceNet, all network layer services require some form of *consideration* along with each service request. Consideration is the medium of exchange of value; that is, consideration is used by one party to convince another to provide a good or service. For practical reasons, the system must admit a variety of forms of consideration.[1] Some connection to a system for transferring money may be required (e.g., a credit card number or bitcoin transaction [1]); in other cases a user may simply need to prove membership in some group (e.g., being a faculty member at a particular univeristy). A receipt (proof of purchase) might also be accepted as consideration. In short, consideration in ChoiceNet can be any form that the customer and provider agree on for exchanging value.

B. Service Description and Composition

A network layer service description contains information about a service's characteristics. It is used to advertise the service in the marketplace, and is also used by *planning* services to compose services together. There are six parts to an network layer service description: (1) the data *transformation/operation*, (2) the *type of input* require, (3) the *type of output* generated, (4) the *input location*, (5) the *output location*, and (6) the *consideration* required. The first three components—the operation, input specification, and output specification—are similar to other interface description languages, web service definition languages, remote procedure calls, etc. The other three components are needed by the economy plane to sell/purchase services and compose them together.

[1]In some cases a network layer service might be offered for free and not require any particular consideration.

One can think of a network layer service as a channel with one or more input endpoints and one or more output endpoints. When the specified *consideration* is given along with request for service, the channel performs the specified operation, (possibly) transforming data arriving on the input endpoint(s) into data leaving on the output endpoint(s). The operation may also have side-effects (e.g., changing the state of the channel).

Composition is achieved by connecting the output from one channel to the input of another channel. However, it is not sufficient to know that a service's output type matches another service's input type. Channel endpoints need to be in the same *location* so that they can be connected. Locations are simply identifiers (names) selected from some namespace (i.e., *scope*) meaningful to the network layer service (e.g., ID of a switch, a port on switch, an AS number, an IP address, an ISP provider name, a geo-location, etc.). Endpoints sharing a location are composable, with ChoiceNet providing the functionality to connect output to input.

Network layer service descriptions are "advertised" by the network layer service to the marketplace, where the marketplace itself is a set of *marketplace services* that allow applications to browse or search the set of available services. Like all services, access to marketplace services requires consideration. Given the ability to discover available services (in the marketplace), one can implement *planning services*, which, given a particular request for service, identify (plan) a composed service that will meet the requirement. The planning service might then invoke *provisioning services* that "purchase" the planned set of composed services (i.e., providing the necessary consideration to each service), or it might return the plan to the user who would invoke a provisioning service to "purchase" the composed service. This ability to hierarchically compose services enables a variety of different business models including resellers, aggregators, brokers, etc.

C. ChoiceNet Protocol for Network Services

Conceptually, ChoiceNet services are "purchased" in the *economy plane* and "used" in the *use plane* (i.e., control and/or data plane). One of the challenges is to develop suitable protocols that enable both invocation of economy plane services and use plane services. For example, communication in the economy plane is likely to resemble conventional request/reply, client-server communication. Communication in the use plane, on the other hand, may take various forms, such as a client pushing data through a series of transformation services. Thus, it might seem that ChoiceNet should support two distinct communication protocols: one for customers purchasing services from providers, and another for applications using services.

While the economy plane/use plane distinction is conceptually useful, the services that are implemented in practice

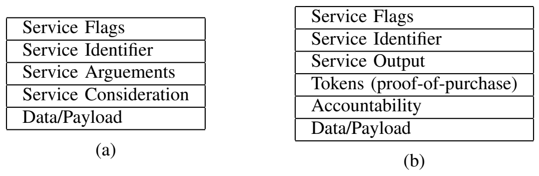

Fig. 1. Basic components of a ChoiceNet (a) Request and (b) Output message.

often cannot be easily classified as economy plane or use plane services. A path service, for example, may collect information from forwarding services to construct and sell paths and thus be considered a marketplace (economy plane) service, but at the same time be considered a use plane service because it computes and returns a set paths along with the "proof of purchase" needed to use those paths. In other words, it both *sells* forwarding service and *computes* paths, and this combination may be necessary to dynamically determine/set prices.

To embrace ChoiceNet's conceptual distinction between the economy plane and the use plane, but allow services to play both roles at the same time, we designed a *single* ChoiceNet communication protocol that is usable by services regardless of the plane to which they belong (or fall between).

D. ChoiceNet Protocol Messages

In the ChoiceNet protocol, services are invoked with a service *request* and may produce an *output*. Figure 1 shows the general structure of a *request* and *output*.

The *request* message is similar to a remote procedure call, indicating which service should be invoked at the server and a list of arguements to be passed to the server. Unlike remote procedure calls, a ChoiceNet request also carries consideration. A generic service flags field carries flags understood by all services (e.g., a "price check" flag that allows a customer to learn the precise cost of performing tasks with a certain set of parameters). The flags field can also be used to indicate that certain fields will be carried in the payload, rather than the header; this allows larger values to be conveyed. Like the request message, the *output* message indicates for which service it is providing results. The message may also carry the output from the service (e.g., a list of "proof-of-purchase" tokens for use with a forwarding service).

E. ChoiceNet Protocol Interactions

There are a number of ways of how this simple ChoiceNet protocol can be used to create interactions that match realistic networking scenarios:

- Iterative use to enable choice: Choice is critical to enable service competition. To let users choose among multiple services, repeated ChoiceNet protocol operations can be used: First, a marketplace is queried to obtain a list of available services. The ChoiceNet protocol is used to send the query to the marketplace (and possibly provide consideration in case the search needs to be paid for). The marketplace returns the available services. In a second protocol exchange, the user then contacts the provider of choice to purchase the actual service.
- Recursive use for composed services: Network services may consist of several pathlets and potential processing and storage services. A provider can hide this complexity to a user by offering a single service. However, when a user purchases this service, multiple subservices need to be instantiated for use. In this case, the single ChoiceNet protocol interaction by the user may trigger multiple, recursive ChoiceNet protocol interactions.
- "One-shot" use for speed: In cases where the user has already made the choice of service, our ChoiceNet protocol can be used very efficiently since all necessary information (service selection and consideration) is included in a single protocol message. Thus, this information can be included in-band with data transmission and does not require additional messages between user and provider.

Note that all three scenarios use exactly the same ChoiceNet protocol, but can achieve different goals.

F. Specifying Service Semantics

Having defined a common message structure for messages in both the economy and use planes, we ultimately need to define precisely what goes into each field of the messages shown in Figure 1. Depending on the target service, the information exchanged in these messages may range from simple flags and identifiers (similar to fields in an IP header) for forwarding services to complex XML structures for services that process packet payloads. Clearly, the customers and providers must agree on the meaning/semantics of the data carried in these fields. Much like there exist protocol standards for the network and transport layers of the current Internet, we expect similar standard will be defined for use/data plane services in ChoiceNet. However, services in the economy plane may rely instead on agreed upon *vocabularies* to define the semantics of messages.

To support a variety of different (extensible) vocabularies, we adopted a triple $\{$ *Attribute Name, Attribute Value, Vocabulary URL* $\}$ as the general structure for information being exchanged in the economy plane. *Attribute Name* identifies the import of the field, and is a literal that must be interpreted the same way by entities that exchange messages containing this attribute. That is, such entities must share a common *vocabulary*. A vocabulary, in this context, may be a simple *dictionary* of literals; the meaning or import of such literals is embedded in the logic of the entities exchanging the message. More generally, it is an *ontology*, where some of the rules for manipulation of such literals is embedded in the vocabulary itself. Examples of *Attribute Name* values are `ChoiceNet Version` or `Message Type`.

Attribute Value is a literal that provides the value of the attribute named by the *Attribute Name*. It may be a number, a string, a list, or it may nest a single, or multiple, other fields (whose values, in turn, may nest others). This allows ChoiceNet entities to ignore entire hierarchies of fields if they are not relevant to the entity's current role or interaction. In other words, an entity may understand the import of a message completely at the top level, without understanding all of the detail structure (but being able to pass them on, say, to another entity). For example, the concept of *consideration* can simply be represented by an attribute field with *Attribute Name* set to `Consideration`. Its value can be a nested structure, representing many different methods of transferring consideration such as mechanisms like PayPal, or previously established contexts like an account number to charge, or credit mechanisms like credit card numbers. Similarly, complex concepts like tokens can be encapsulated in single attribute fields with internal structure that can vary from use to use.

Finally, *Vocabulary URL* provides the basis for an extensible vocabulary, by allowing the sender of the message to indicate where the vocabulary being used for the value of the *Attribute Value* is available. It may well be that this vocabulary is the same as that needed to understand this field's *Attribute Name* itself, but the ability to specify a different vocabulary for any field's *Attribute Value* allows providers of innovative services to immediately start using existing ChoiceNet marketplaces and other mechanisms, and incrementally build an ecosystem of other entities who understand the new custom vocabulary.

From the above, it is clear that services that rely on vocabularies must *a priori* understand all top-level attribute field *Attribute Name* values – this represents the bootstrapping vocabulary, and can be considered the common core vocabulary. This common core can be minimal. Further, we reasonably expect that the core vocabulary will grow over time, as practice makes it clear what vocabularies are most helpful to the ChoiceNet user community.

IV. Prototype Implementation

To evaluate the ChoiceNet protocol, we developed two GENI [5] prototypes. Both prototypes add new ChoiceNet services to the IP protocol to leverage and maintain compatibility with existing applications. A fundamental requirement of ChoiceNet is to support alternative paths. In particular, ChoiceNet needs to support per-flow dynamic routing based on the user/application's requirements. Legacy static routing and adaptive routing does not meet these requirements, so we solved the dynamic routing problem using two approaches. The first one uses SDN-based pathlets to compose end-to-end paths, while the second one uses a set of link-to-link packet forwarding services to compose end-to-end paths.

A. SDN-Based Prototype

Software Defined Networking (SDN) is an approach that decouples network systems into the control plane and the data plane [14]. Using SDN, we can allocate the path for each flow by installing flow entries on switches along designated path. This approach allows providers in ChoiceNet to provision path services to users.

We build a minimal SDN-based IPv4 prototype of ChoiceNet on ExoGENI [4] as shown in Figure 2. The two ASes and the marketplace are in three different locations. The two links between the two ASes are throttled to 1Mbps and 10Mbps, respectively, and unlabeled links are 1Gbps. In our prototype, we use pox[2] as the SDN controller, and Open-

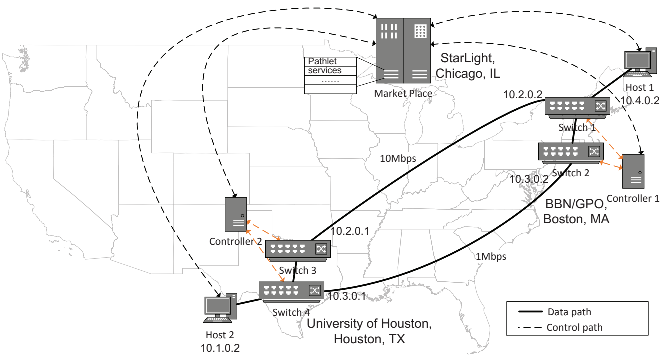

Fig. 2. SDN-based prototype topology.

vSwitch[3] as the SDN switch. The SDN protocol is OpenFlow 1.0. The hosts are running 64-bit Ubuntu 12.04.

1) Marketplace Design: In this prototype, the marketplace is a server written in Python. It uses standard sockets to communicate with the controllers and user apps. The requests and responses are encoded in JSON[4] text format, and follows the semantics defined in Section III. The marketplace currently only offers one type of service—the pathlet service. The pathlet service is a directional path defined by the *location* of source and destination, which are IP addresses. An example of pathlet service is shown in Table I. Note that the description can be different for each service types, and the JSON representation ensures the extendibility in the future.

The marketplace is responsible for handling users' service requests and notifying the control plane to do the provisioning. It also serves as an intermediate of payment between the users and providers. The marketplace constructs a directional multigraph from the pathlet services advertised by the controllers. When the marketplace receives a service planning request, it takes the source IP and the destination IP, and starts a modified Breadth First Search on the graph. The search attempts to find multiple Pareto-optimal solutions using a branch and bound method as it traverses the network graph. Finally, the Pareto-optimal subset of paths is presented to users for selection.

To handle payments, there is a web server co-located with the marketplace to perform authentication with PayPal. This web server interacts with the marketplace by sharing its database and exposes a HTTP-based JSON API to user applications for PayPal payments.

2) Use Plane Design: The use plane consists of the controllers and the switches. The controller is an SDN controller with customized control logic. It detects the topology of switches with LLDP packets and detects hosts by their DHCP, ARP, and IP packets. When a new host or a new link has been detected, the controller updates the new pathlet service to the marketplace, thus the marketplace knows all the services in all ASes. Another task of the controller is provisioning: once a provisioning request is received, the controller installs flow entries on the switches along the designated path.

The main difference in paradigm of a controller in ChoiceNet and a standard OpenFlow controller is: the instal-

[2]http://www.noxrepo.org/pox/about-pox/

[3]http://openvswitch.org/

[4]http://json.org/

TABLE I. EXAMPLE PATHLET SERVICE.

Attribute Name	Attribute Value
Service ID	10.1.0.2_0_10.3.0.2
Service Type	NetworkLink1000
Controller ID	192.168.0.15
Controller IP	192.168.0.15
Endpoint 1 ID	10.1.0.2
Endpoint 1 IP	10.1.0.2
Endpoint 2 ID	10.3.0.2
Endpoint 2 IP	10.3.0.2
Service Bandwidth	1Mbps
Service Latency	15 ms
Service Cost	0.001 USD

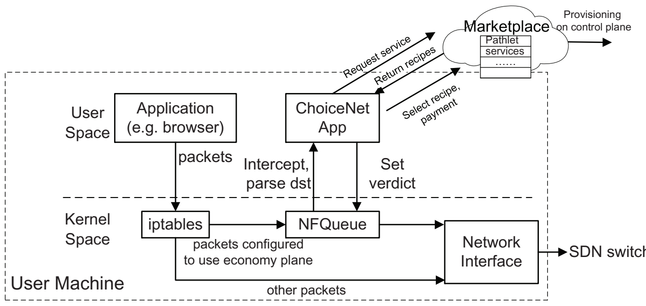

Fig. 3. ChoiceNet App interactions on end-system.

lation of flow entries is not triggered by the first packet of each flow. Instead, flow entry installations are triggered by the provisioning command from the marketplace–after the users have requested and paid for the service.

3) User App Design: To allow the user to make choices, a program (ChoiceNet App) on end-systems is used. The function of this app is shown in Figure 3. It uses NetFilter Queue[5] to intercept the initial packet of each connection (except the connections that goes to the marketplace). The app then contacts the marketplace, asking for a path service to the destination IP. After the marketplace returns a list of available service combinations, the app prompts the user to select one service. After the selection, the user is redirected to a PayPal payment page. After receiving the payment notification from PayPal, the marketplace transfers the money to the account of the controller(s) and notifies the latter to provision the services. After the provisioning, the app releases the intercepted packet and traffic will traverse along the assigned path.

It may be impractical for the user to select and pay for each network connection. Instead, network services can be made more granular (e.g., encompassing all connections to a video service provider for a week) and preferences can be specified in the ChoiceNet app to automate the service payment process.

B. A Packet Forwarding Service

Our second prototype introduced a per-node *packet forwarding service (PFS)* to support user-selected end-to-end paths. The topology is similar to that of Figure 2, except that SDN switches were replaced with programmable Click [13] routers, each programmed to offer a PFS that advertises the ability to relay packets between every possible ingress-to-egress combination of links attached to the router. Each advertisement includes information about the most recent performance (latency and bandwidth) of the links so users/applications can select the path that best meets their needs. We used IPv6 extension headers to encode the series of forwarding services a packet should traverse end-to-end—similar to carrying a source route in the packet, but with the addition of the ChoiceNet header components (notably *consideration*).

The PFS's on the Click routers understand IPv6 extension headers and use the next forwarding specification (i.e., the next ingress/egress pair) in the packet to determine the outgoing link. Prior to forwarding the packet, the service verifies, via the consideration, that forwarding has been paid for in the economy plane. Because this check is performed on every packet, it must be efficient. We used *delegated capabilities* similar to those used in Platypus [15], which simply involves a cryptographic comparison (i.e., HMAC operation) as opposed to a complex transfer of consideration in the use plane.

1) The Marketplace: To help find appropriate end-to-end paths, we developed a *path service* that applications can contact to request paths that satisfy their requirements. The path service registers itself with a marketplace listing service where users/applications can go to learn about, and purchase access to, the path service. Having purchased the ability to use the path service (i.e., sending it the appropriate consideration), the sender's machine is given a token (proof-of-purchase) that is used as consideration when contacting the path service.

To allow legacy applications to select paths, we developed a "wrapper library" that is loaded at runtime along with an application (via the LD_PRELOAD environment variable in Unix) to intercept all networking calls made to the OS such as socket(), bind(), connect(), send(), recv(), etc. The wrapper library uses the previously acquired token to request a set of paths from the path service. The path service then returns a set of "paths" (i.e., a series of packet forwarding services) to the wrapper library along with the consideration (delegated capabilities) needed to use the forwarding services. The set of paths that are returned also include information about the performance of the path that can be used by the wrapper library to select the most appropriate path. To determine which path is the most appropriate, the wrapper library consults a local policy file that contains a policy entry for each wrapped applications (e.g., "ssh: low latency" or "scp: high bandwidth"). The wrapper selects the best path and includes it in the IPv6 extension headers for packets originating from the application.

To compute the best routes, the path service collects the routing advertisements issued periodically by the packet forwarding services. A single service collecting performance information and computing paths may not appear to scale well, but our recent analysis of current Internet paths shows that the processing can be easily parallelized to produce a scalable service [2]. To enable efficient consideration checking in the use plane, the path service periodically "purchases" the right to use the forwarding services by providing consideration to the forwarding service and in return receives a (time-limited) delegated capability that it can further delegate to customers of the path service.

[5]http://www.netfilter.org/projects/libnetfilter_queue/

TABLE II. BREAKDOWN OF CONNECTION SETUP TIMES.

Task	SDN	PFS
Resolve DNS name to IP address	1.46 ms	1.46 ms
Contact marketplace/path service	17.93 ms	0.25 ms
Plan (SDN only) and select paths	24.59 ms	0.00 ms
Provision the path	44.67 ms	0.00 ms
Total connection setup time	88.65 ms	1.71 ms

C. Performance Results

Although the ChoiceNet marketplace enables choice, it also introduces additional overhead when originating a flow. To quantify the additional overhead, we measure the average time it takes to perform each step of setting up a flow (see Table II).

The planning (path computation) and purchasing overhead is included in the cost of accessing the path service for PFS, while planning and purchasing is performed by the ChoiceNet client machine for SDN (our results here assume the choice is made programmatically—not by a human). The two approaches represent different business models. PFS precomputes paths (in parallel) and repeatedly pre-purchases paths for short periods of time so most paths queries can be answered immediately. The SDN service, on the other hand, includes the application in this planning/selection process giving the user/application greater control of what is purchased/used, but also increasing the setup overhead. So the numbers in Table II's SDN implementation are affected by the round trip time from the user/provider to the marketplace. The provisioning step for SDN also increases delay, but packets flow through the use plane without any added overhead. The PFS has no provisioning costs, but requires a consideration check (an HMAC operation, which takes on average 11 usecs on a Xen Click router) at every router along the path. Also note that when compared with the time to do the DNS lookup (which IP already requires), consulting the path service adds relatively little to the overall connection setup time.

V. SUMMARY AND CONCLUSIONS

Economic relationships between entities in the network are a critical driver for operation of the Internet. In this paper, we have presented the design of a protocol that can associate economic contracts with network layer services. We have shown that this general approach can be instantiated in two fundamentally different prototypes, one using out-of-band signaling and one using in-band signaling. We have presented results from implementations on GENI to illustrate the effectiveness of our protocol design.

ACKNOWLEDGEMENTS

This material is based upon work supported by the National Science Foundation under Grant Nos. 1111040, 1111088, 1111256, 1111276.

REFERENCES

[1] Bitcoin. https://bitcoin.org/.

[2] O. Ascigil, K. L. Calvert, and J. N. Griffioen. On the scalability of interdomain path computations. In *IFIP Networking 2014*, June 2014.

[3] A. C. Babaoglu and R. Dutta. A verification service architecture for the future internet. In *Proc. of the 22nd IEEE International Conference on Computer Communications and Networks (ICCCN)*, Nassau, Bahamas, Aug. 2013.

[4] I. Baldine, Y. Xin, A. Mandal, P. Ruth, C. Heerman, and J. Chase. Exogeni: A multi-domain infrastructure-as-a-service testbed. In *Testbeds and Research Infrastructure. Development of Networks and Communities*, pages 97–113. Springer, 2012.

[5] M. Berman, J. S. Chase, L. Landweber, A. Nakao, M. Ott, D. Raychaudhuri, R. Ricci, and I. Seskar. Geni: A federated testbed for innovative network experiments. *Computer Networks*, 2014.

[6] K. L. Calvert and E. W. Zegura. Composable active network elements. http://www.cc.gatech.edu/projects/canes/.

[7] D. D. Clark, J. Wroclawski, K. R. Sollins, and R. Braden. Tussle in cyberspace: defining tomorrow's Internet. *SIGCOMM Computer Communication Review*, 32(4):347–356, Oct. 2002.

[8] C. Diot, B. N. Levine, B. Lyles, H. Kassem, and D. Balensiefen. Deployment issues for the IP multicast service and architecture. *IEEE Network*, 14(1):78–88, Jan. 2000.

[9] A. Dwaraki and T. Wolf. Service instantiation in an Internet with choices. In *Proc. of the 22nd IEEE International Conference on Computer Communications and Networks (ICCCN)*, Nassau, Bahamas, Aug. 2013.

[10] P. B. Godfrey, I. Ganichev, S. Shenker, and I. Stoica. Pathlet routing. In *Proc. of the ACM SIGCOMM Conference on Data Communication*, pages 111–122, Barcelona, Spain, Aug. 2009.

[11] X. Huang, S. Shanbhag, and T. Wolf. Automated service composition and routing in networks with data-path services. In *Proceedings of the IEEE ICCCN 2010 Conference*, 2010.

[12] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. H. Briggs, and R. L. Braynard. Networking named content. In *Proceedings of the 5th international conference on Emerging networking experiments and technologies (CoNEXT)*, pages 1–12, Rome, Italy, Dec. 2009.

[13] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek. The Click modular router. *ACM Transactions on Computer Systems*, 18(3):263–297, Aug. 2000.

[14] Open Networking Foundation. Software-Defined Networking: The New Norm for Networks. White paper, Open Networking Foundation, Palo Alto, CA, USA, Apr. 2012.

[15] B. Raghavan, P. Verkaik, and A. C. Snoeren. Secure and policy-compliant source routing. *IEEE/ACM Trans. Netw.*, 17(3):764–777, 2009.

[16] T. Wolf, J. Griffioen, K. L. Calvert, R. Dutta, G. N. Rouskas, I. Baldine, and A. Nagurney. Choice as a principle in network architecture. In *Proc. of ACM Annual Conference of the Special Interest Group on Data Communication (SIGCOMM)*, pages 105–106, Helsinki, Finland, Aug. 2012. (Poster).

[17] T. Wolf, J. Griffioen, K. L. Calvert, R. Dutta, G. N. Rouskas, I. Baldine, and A. Nagurney. ChoiceNet: toward an economy plane for the Internet. *ACM SIGCOMM Computer Communication Review*, 44(3):58–65, July 2014.