Rapid and Scalable ISP Service Delivery through a Programmable MiddleBox

Kamran Riaz Khan
Microsoft Corporation
Redmond, WA, USA
kakhan@microsoft.com

Zaafar Ahmed, Shabbir Ahmed, Affan Syed
SysNet Lab, NUCES
first.last@sysnet.org.pk

Syed Ali Khayam
PLUMgrid Inc.
Sunnyvale, CA, USA
khayam@plumgrid.com

ABSTRACT

With only access billing no longer ensuring profits, an ISP's growth now relies on rolling out new and differentiated services. However, ISPs currently do not have a well-defined architecture for rapid, cost-effective, and scalable dissemination of new services. We present iSDF, a new SDN-enabled framework that can meet an ISP's service delivery constraints concerning cost, scalability, deployment flexibility, and operational ease. We show that meeting these constraints necessitates an SDN philosophy for a centralized management plane, a decoupled (from data) control plane, and a programmable data plane at customer premises. We present an ISP service delivery framework (iSDF) that provides ISPs a domain-specific API for network function virtualization by leveraging a programmable middlebox built from commodity home-routers. It also includes an application server to disseminate, configure, and update ISP services. We develop and report results for three diverse ISP applications that demonstrate the practicality and flexibility of iSDF, namely distributed VPN (control plane decisions), pay-per-site (rapid deployment), and BitTorrent blocking (data plane processing).

1. INTRODUCTION

An explosive growth in connectivity has resulted in unprecedented increases in ISPs' CapEx (infrastructure) and OpEx (maintainence). Consequently, the original ISP business model of billing customers for access (flat-rate or metered) is not economically sustainable anymore [1]. ISPs are now exploring additional revenue lines through value-added services. Prominent examples of such services include online virus scanners, personal firewalls (e.g. NetProtect Plus by BT [14]), triple-play (internet, TV, phone), video-on-demand [11], and network-based enterprise services like VPN and VPLS [1]. Besides value-added services, ISPs have several diverse and country-specific regulatory requirements that stress their operational capabilities[1].

We held discussions with two of the biggest ISPs in Pakistan, Nayatel and PTCL [11, 8], to understand their concerns and challenges in rolling out new services. We substantiated that cost and scalability remain the biggest concerns while evaluating feasibility of new services. Besides these two major concerns, Benson et al. [1] also identified complexity, disruption, and configuration churn as significant impediments in an ISP's service delivery.

All of the above challenges are a consequence of the current service deployment mechanisms. Today, a new ISP service is either deployed centrally, within the ISP's network, or at customer premises as a *service-specific* box or software. Both deployment locations have their limitations. Services deployed at the ISP core can only *scale-up* with respect to customer-base or traffic rates. These solutions also need to be purchased from third party vendors, hence incurring significant capital (purchase) as well as operational (support and software upgrades) expenditures due to vendor lock-in. Services at customer premises also incur issues of intrusiveness, difficulty as well as cost of installation (hardware/software support provided by on-ground support staff, as most users are not technologically savvy), and security (users can unintentionally circumvent host security). These constraints seriously limit the types of services as well as the rates at which ISPs can develop and offer them to customers.

Here, we argue that a software defined networking (SDN) approach — allowing programmable redefinition, manipulation and alteration of network traffic and configuration — is perfectly suited to remove these impediments. However, we observe that current SDN frameworks, designed for data-center and enterprise networks [12, 7], are not adequate for overcoming an ISP's service delivery challenges. We therefore propose and evaluate a novel SDN-powered ISP service delivery framework (iSDF) and its prototype implementation.

2. iSDF: DESIGN GOALS AND OVERVIEW

Our primary aim is to design a framework (iSDF) that allows an ISP to easily and quickly create, deploy and manage innovative services for its customers. Such frameworks are already in use in mobile networks, like UMobile TV [23] and T-Mobile Clever Connect [21]. One reason such frameworks have not been adopted by wireline ISPs is, we believe, that their network infrastructure does not include a programmable end-device, such as smartphones. Inspired by this observation, we propose using a similar programmable middlebox at ISP customer premises. We now present the goals that drive the design and architecture of iSDF.

2.1 Design Goals for iSDF

We set three important design goals for iSDF. First, iSDF should allow ISPs to incrementally and cost-effectively *scale-out* in proportion to their customer-base and traffic demands (Goal ❶). Second, iSDF should allow ISPs to *rapidly develop*

[1]For instance, ISPs in Pakistan, KSA, and China are required to block access to specific portals; implementing such requirements can, and have, lead to unintended consequences of service disruption and revenue loss.

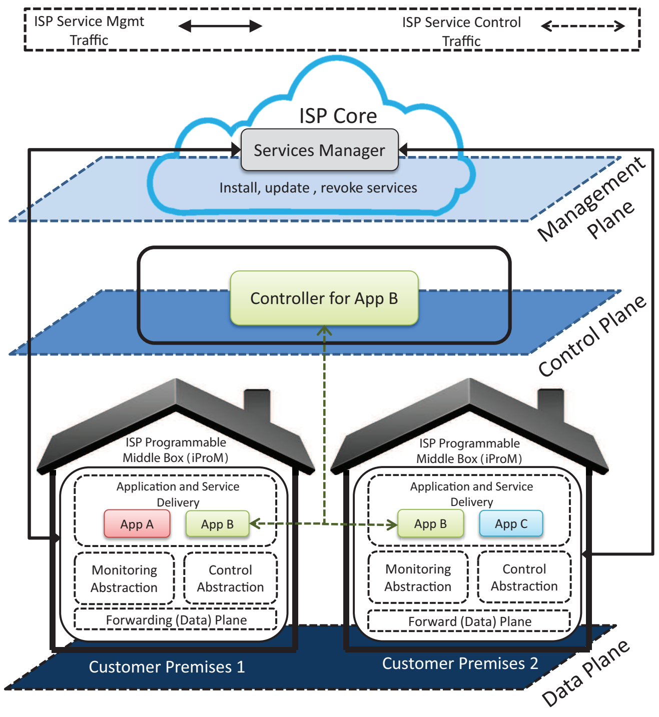

Figure 1: Overview of iSDF showing its three main components.

and deploy new services (Goal ❷). Lastly, for widespread take-up, iSDF should be *operationally simple and hot-pluggable* (Goal ❸). More specifically, pushing new functionality should happen without manual intervention or disruption of existing services.

2.2 Key Design Decisions for iSDF

With the above design goals in mind, we present a framework (iSDF) to replace the current monolithic and centralized appliances used to introduce ISP services [18]. We first introduce our key design decisions to meet the goals identified in Section 2.1. In the next section we elaborate the three core components that distribute the service logic to customer premises (Figure 1).

Goal ❶ motivates a distributed model with intelligence at customer premises, thus decoupling service delivery from the implementation at the ISP core. A corollary of this goal is to use commodity hardware to address cost concerns for real-world deployment of iSDF. The challenge in achieving Goal ❷ is that service use cases differ considerably across ISPs. Hence, to simultaneously meet the scalability (in traffic processing) and diversity (in use cases) goals, iSDF must provide a programmable middlebox at customer premises for data plane decisions. We thus propose an SDN-inspired framework that provides deep programmability of the data plane (DP) using a domain-specific API. This API is complemented with a service model that also allows the ISP to programmatically invoke control plane (CP) decisions across different customer premises[2]. We meet our third goal (Goal ❸) by designing a service delivery mechanism that allows for plug-and-play functionality of services at customer premises without disruption to basic connectivity and other services.

[2]Customer premises are the basic network unit of an ISP.

2.3 iSDF Architectural Components

The iSDF architecture divides the control of flexible ISP services by employing programmable middlebox functionality at the edge, control plane functionality that can be distributed, and a centralized management interface for the ISP. We next describe components that enable this functionality.

2.3.1 *Services Manager*

The **Services Manager**, residing on the management plane, acts as a centralized coordinator for installing, configuring, updating, and revoking services. This centralized location at the ISP core shall not be a scalability constraint as we expect a low churn for management operations, perhaps on the order of a single operation per day/customer.

2.3.2 *ISP services*

A big component of our framework are the ISP services written over a programmable data-plane (Applications A, B, and C at customer premises in Figure 1). These applications use our monitoring and control APIs to build ISP services. *Optionally*, they can have a centralized component (Controller for App B in Figure 1) that allows for correlation and coordination by a controller application over a programmable (perhaps using OpenFlow [9]) control-plane.

2.3.3 *ISP Programmable Middlebox (iProM)*

The core novelty of our proposed framework is a programmable end-device, iProM, at the customer premises that iSDF will leverage to deliver, monitor, and implement ISP services. This middlebox supports hosting of services offered by the ISP at the application and service delivery layers. iProM provides the ISP's IT department with an easy and programmable platform to monitor the traffic and network setup, essentially allowing for a traditional NOC-like view. After discussion with ISP operators, we envision that this box will be deployed on the access device provided to customers.

2.3.4 *Monitoring and Control API*

Our API provides two high level abstractions. The first is an API for passively *monitoring the data plane* using a domain-specific language that allows easy and programmable registration to application-relevant network events. The second is a set of APIs for controlling the DP, where we propose that the CP, similar to other SDN models, is made fully programmable. An iSDF service can thus be run entirely in the DP, entirely in the CP (through packet punts), or partly in control and data planes. This framework provides the service programmer with flexibility to design an application based on its scalability and visibility requirements.

In the API design context, we observe that the monitoring required for ISP applications is similar to that required by network intrusion detection systems (NIDS). Thus, from an implementation perspective, we propose reusing an industry standard NIDS (e.g. [13, 16]) to write new network services.

Table 1 provides an overview of our control API that ISP applications can leverage from *within* service scripts. We have currently divided the API into three broad categories: Traffic filtering (flow blocking), traffic shaping (QoS of flows), and network virtualization. Our current API is, admittedly, restricted and a work-in-progress. Currently it is geared towards enabling the applications that we use for

Category	Example Applications	API	Comments
Traffic Filtering	**P2P Blocking**, Content censorship, Botnet and DDoS protection	`block(flow, duration)`	Block `flow` for specified `duration` of seconds. Supports wildcards for `flow` definition.
Traffic Shaping	Video on demand, VoIP, P2P traffic shaping, **Pay-per-Site**	`addClass(classId, bandwidth)`	Defines a new `class` of traffic which should be throttled to use specified `bandwidth`.
		`removeClass(classId, bandwidth)`	Undefines an existing `class` of traffic
		`addClassMember(classId, flow)`	Adds a `flow` to a `class`. All traffic matching the `flow` definition shall be throttled to the respective `bandwidth`. Supports wildcards for `flow` definition.
		`removeClassMember(classId, flow)`	Removes a `flow` from an existing `class`.
Network Virtualization	**Distributed VPNs**, VPLS, L2TP	`addTunnel(remoteAddress, localAddress, localTunnelAddress, remoteTunnelAddress)`	Create a tunnel between `localAddress` and `remoteAddress` and assign respective `*tunnelAddress`es.
		`addRoute(network, remoteTunnelAddress)`	Route data for `network` to `remoteTunnelAddress`.

Table 1: iSDF API for a programmable data plane (at customer premises) and an optional control plane at service-specific controller. Data plane *monitoring* uses the comprehensive Bro language API.

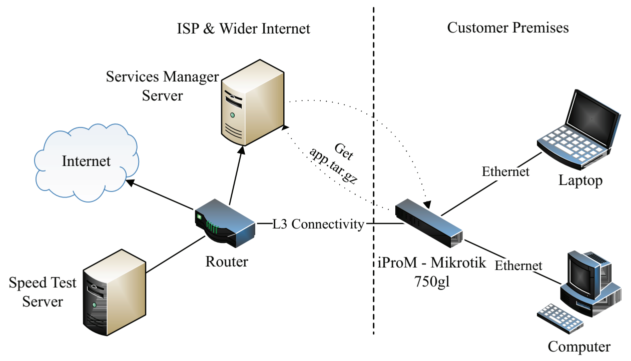

Figure 2: Our iSDF prototype emulating customer premises with access to ISP core and Internet.

evaluation in Section 4, but it is meant to demonstrate the flexibility of our programmable data-plane. We provide implementation details for this API in Section 3.

3. iSDF IMPLEMENTATION

We now discuss and evaluate our prototype implementation of the iSDF framework. This new prototype of a SDN-powered ISP service delivery framework comprises three parts: a network setup emulating ISP-customer connectivity, the service delivery mechanism, and implementation of the data plane monitoring and control API.

3.1 Operational Setup of iSDF Prototype

We begin by describing the network setup we employ to emulate ISP-customer connectivity (Figure 2). We implement iSDF over this network setup to evaluate the delivery of ISP services using our framework.

Our operational vision of the service delivery system relies on developing a low-cost, active, and flexible middlebox at customer premises. We therefore implement iProM on a commodity home-router (Mikrotik 750gl router with 64MB RAM @ 400Mhz) for a low-cost solution. We choose OpenWRT platform as it allows us to leverage the large base of open-source Linux tools; it thus facilitates an iProM device that provides expressive and flexible monitoring as well as active control of network traffic and configurations.

We emulate access to the ISP core and wider Internet using an L3 router. We locally host both the *Services Manager* (conceptually at the ISP) and any application-specific server (conceptually anywhere on the Internet) as two-hop, L3-accessible machines for the iProM.

3.2 Data Plane Monitoring and Control

In our implementation of iProM, we choose the Bro programming language to provide an expressive and powerful layer to *monitor* the data plane [13]. Bro, while initially intended for network intrusion detection, has evolved to provide very efficient, event-based monitoring of network traffic. Furthermore, Bro filters network events such that its processing requirement is proportional to the depth of packet inspection, thus allowing us to scale with application needs by using a more powerful (and costly) middlebox. The choice for using Bro is made easier owing to an active Bro community that allows our monitoring framework to remain updated. To enable this choice, we ported Bro to OpenWRT[3].

ISP services therefore are Bro scripts written by the service developer. We provide three different categories of control API (Table 1), *within* the Bro framework, that a developer can use to control aspects of network traffic and configurations. We believe that the expressive language of Bro and our control API will together allow developers to quickly build novel network services.

We provide a single function for the **Traffic Filtering** category. This function is implemented using *iptables* and requires the Bro-specific *connection structure* for representing a flow as a 5-tuple, and generates a filtering rule for this flow. We also generate an "anti-rule" that rescinds the original rule and is executed after the `duration` specified. We believe that our implementation of flow definition, with wildcards, is sufficient for most ISP applications. However, this is not a constraint on the framework, and flow definitions can be enhanced using other tools more sophisticated than iptables.

For **Traffic Throttling** we provide functions for managing different traffic classes. Our API creates classful queuing disciplines (*qdisc*) via the *tc* utility, with each class assigned a specific bandwidth. We use the iptables firewall to categorize each flow, defined by a 5-tuple, into a particular class. The kernel queuing mechanism then restricts each flow to the bandwidth specified for its class. Our API provides a way to not only remove differentiated flows from a class (policy-based) but also traffic classes themselves (service removal).

[3]For details of port and all applications developed in Section 4 visit: sysnet.org.pk/w/ISDF

The final category for **Network Virtualization** provides an API to build L3 tunnels and correspondingly add routes on the iProM devices. We use the *iproute2* package for building a GRE tunnel interface between specified endpoints. We then bring up this interface, setup a routing rule to the remote tunnel subnet, and use iptables to add a firewall rule allowing forwarding on the new tunnel interface. We expect to expand our API in this section to allow L2 virtualization (using for example, VPLS or L2TP).

We also implement simple heuristics (similar to work by Ferguson et al. [4]) to identify conflicts between iProM rules regarding traffic classes and network virtualization, thus simplifying the job of a service developer.

3.3 Service Delivery Mechanism

We need a robust and scalable service delivery platform with minimal overhead and the ability to hot-plug (install) and hot-swap (update) applications. For this purpose, we employ a hybrid push/pull model for service delivery. The iProM periodically (default 30 minutes) pulls applications from an ISP-specified, customer-specific URL. These application are downloaded as a tarball containing the application scripts along with their respective MD5 hashes. These scripts are executed immediately on success of a uniqueness check performed by comparing with previous hash. We envision that ISPs will digitally sign these scripts to ensure service integrity. We augment this pull mechanism with an asynchronous push that the ISP services manager can execute by sending a connection attempt on a port where iProM is listening. Once a connection is established at this port, the iProM simply repeats the pull process described above.

While we currently have a proprietary implementation to push services, looking into using package managers present in OpenWRT is a future work we will explore.

3.4 Discussion

Here we discuss our reason for choosing a custom SDN implementation, forgoing the benefit of using a standardized implementation like OpenFlow [9]. While OpenFlow is one possible option, we see two constraints that render OpenFlow inappropriate for implementing iSDF.

Scalability: A fundamental design objective of iSDF is to allow operators to roll out new services *at scale*. With an increasing ISP customer base, scalability can be achieved only by pushing as much packet processing functionality to the data plane as possible, and only forwarding the absolutely-necessary information to the control plane (CP). OpenFlow architecture, however cannot support such an implementation, as it requires the main packet processing application to run in the CP, while a simple $< match, action >$ packet processing engine operates in the DP.

Programmability: Application use cases will vary considerably across different ISPs and target demographics. These applications require different types of L2 to L7 packet processors and programs that could control these packet processors to come up with the requisite metrics.[4] To achieve this level of programmability with OpenFlow, iSDF would again have to forward every packet to the CP. We instead chose to provide a contained set of ISP-specific data-plane APIs (admittedly limited at this point) that can be invoked as part of a code by. This API-constrained *code* (and not simple flow-rules) can now be pushed by the iSDF control plane to our middlebox, allowing for a flexible and simple path to service deployment. Thus our framework enables application writers to decide their split of application functionality between the data and control planes.

[4]For instance, detecting that a particular URL is present in an HTTP query, or finding that a particular application is communicating on an abnormal number of unique ports.

4. EVALUATING OUR iSDF PROTOTYPE

We now evaluate our prototype implementation (Figure 2), along three axes (programming ease, deployment flexibility, and performance overhead) using three diverse use-cases. These use-cases are by no means exhaustive; in fact, we believe that our programmable platform will enable implementation of novel and innovative applications.

4.1 Programmability: Distributed VPN

Our first application implements the enterprise VPN service described by Benson et al. [1], to demonstrate the ease in developing services, having even the optional control plane component, using iSDF. Our implementation of this service has two novel facets. First, our distributed implementation at enterprise end-points decouples tunnel creation from both vendor hardware and implementation of the ISP core. This decoupling simplifies management and allows ISPs to incrementally scale out, in contrast with current solutions that are expensive, difficult to configure, and vendor-locked [1]. A second novelty is the automation of tunnel creation by using a control plane decision triggered for every new site. Thus, for enterprise sites running this service, a bootstrap portion provides credentials to a central service-specific server (App B in Figure 1). This central server, after validating credentials, treats every connection as a new *virtual* link coming up, and makes a control-plane decision to set up a virtual L3 network between clients that match the enterprise VPN policy (hub/spoke, mesh). Our applications at each endpoint then establish tunnels and routes for the new site.

We have implemented this powerful and complex service using just 74 lines of code (31 lines of Bro script and 43 lines of php at the central server) in our iSDF prototype implementation. This clearly demonstrates the potential of our framework for improving existing services as well as for lowering cost and implementation complexity for ISPs.

4.2 Rapid Deployment of Services: Pay-per-Site Packages

Our next application is inspired by mobile ISP operators providing differentiated and site-specific access to their users (e.g. unlimited access to TV streams [23]), with an aim to show rapid deployment of a new service through iSDF. We demonstrate the installation and application of such a "pay-per-site" service by hosting two servers on our emulated Internet. The ISP services manager provisions for this application in the ISP core and subsequently pushes a new service that, using our API, renders appropriate bandwidth for the selected destination. This service is implemented in iSDF using just 25 lines of code. As comparison, PTCL and Nayatel currently employ Juniper SRC solution, for approximately half-million USD, to provide similar services [18].

For performance evaluation, we downloaded 600 MB files from two different mirrors (denoted as mirrors A and B). The bandwidth is throttled to 150kBps on the Internet connection. As shown in Figure 3, initially both downloads equally share the 150kBps *base* bandwidth; at 200s the cus-

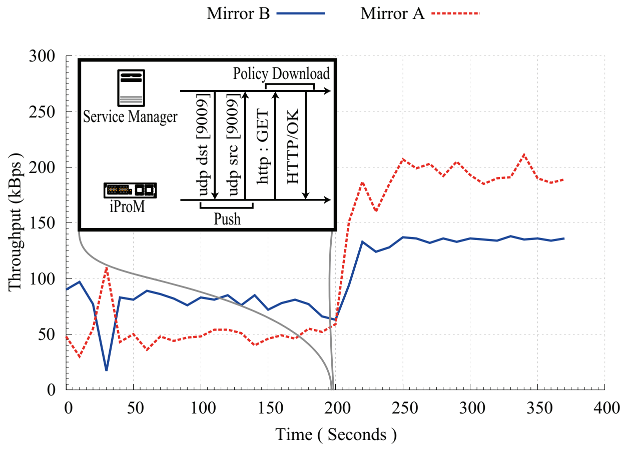

Figure 3: At ≈ 200s a customer selects our Pay-per-Site service (for mirror A) to get higher bandwidth as iSDF installs a new service.

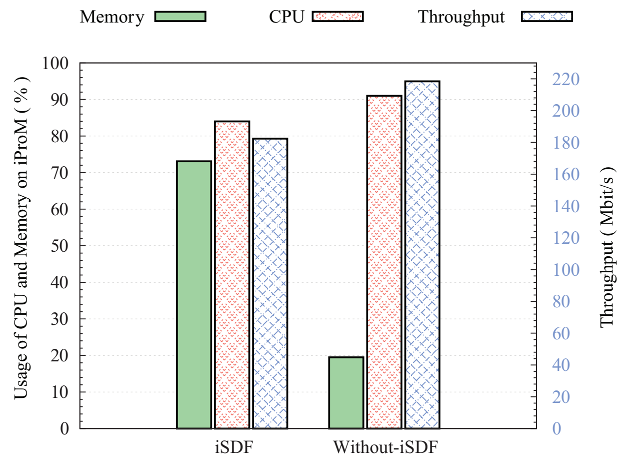

Figure 4: Impact of running a data plane intensive application detecting BitTorrent flows.

tomer purchases (out-of-band) the service for upgrading his bandwidth for mirror A. The ISP services manager, having validated the payment, installs the service using the push mechanism described in Section 3.3 (inset in Figure 3). With user-imperceptible delay, iProM upgrades the bandwidth to 200kBps for the flow to mirror A while the flow to mirror B starts utilizing the full base bandwidth of 150kBps.

4.3 Performance Overhead: Torrent Blocking

Our final application is the detection and blocking (or shaping) of BitTorrent flows, using custom protocol analyzers that *predict* BitTorrent connections (including failed TCP handshakes) by analyzing peer-coordination traffic (trackers, DHT, PEX) [6]. Since this application requires detailed packet analysis with no correlation across multiple iProM devices, it is architecturally well-suited to be implemented completely in the data plane. Furthermore by requiring processing of all packets – the depth of inspection varying based on headers – this application represents the worst-case processing overhead for an iSDF application. Similarly, the cost of maintaining state (for *possible* BitTorrent connections) makes implementation of this service infeasible at the ISP core. This service aims to demonstrate a data-plane intensive application that cannot be implemented at the core and to bound the impact that iProM, like any middlebox, has on performance.

We evaluate this service, running on an iProM, as two local computers download torrents from the Internet. Simultaneously, we perform speed tests from a local server (shown in Figure 2). We use our traffic filtering API to block identified BitTorrent flows for a policy-specific time. Figure 4 shows that while memory consumption increases on our iProM, it stays within the 64Mb RAM common for low-cost home routers. We notice that since CPU usage decreases, the observed throughput decrease is due to per-packet inspection that increases forwarding latency. However, the resulting throughput decrease of about 16% is itself acceptable for such an intensive middlebox application.

5. RELATED WORK

Benson et al. [1] analyze several years' worth of configuration data on a tier-1 ISP and conclude that device configuration complexity, which grows with time, is a major bottleneck in ISP service deployment. iSDF aims to resolve this by equipping ISPs with a scalable mechanism to roll out services using a programmable middlebox at customer premises.

Project BISmark and Project Homework have both developed SDN-inspired home/SOHO based solutions for ISP *customers* [2, 20, 19, 22]. While most of these involve passive monitoring and collection of information for applications like network troubleshooting and net-neutrality observation [2, 20], a small subset has looked at active manipulation to allow users to manage bandwidth caps per device and improve intrusion detection [3, 10]. All these works are complementary to ours as they target end-user products but have no interaction with the ISP.

Recent research has been directed at managing and configuring middlebox deployments using SDN techniques; SIMPLE and OpenMB are two examples [15, 5]. The focus of these work remains in providing greater reliability and simplifying the management for (re)configuration of middleboxes. Our work focuses on a identifying a specific location for middleboxes that provides a mechanism to deliver ISP-specific services at customer premises. However, these middlebox management frameworks can (in parallel) be used for load-balancing and failure recovery of iProM at SOHO customer premises.

Finally, the idea behind Netcalls presents a simplified API to logically abstract network services for clients [17]. This work is from the perspective of clients, with services being implemented inside the network, by either the clients' home ISP or a peering ISP, with service resolution performed by the home ISP. In contrast we scalably implement network services at the customer premises using our programmable middlebox.

6. CONCLUSIONS AND FUTURE WORK

In this paper we propose a distributed ISP service delivery framework (iSDF) that decouples ISP services from its core allowing portable, cost-effective, flexible, and easy management of ISP services across different hardware vendors and ISP core configurations. As future work, we plan to work closely with ISPs to enrich our API for greater convenience in developing ISP services. In addition, we aim to incorporate security measures in the service delivery mechanism to prevent exploitation of the push/pull mechanism for DDoSing the Services Manager. Similarly, the authenticity of the service payloads shall be verified using a public-key signature scheme, or by employing the existing package management systems provided in Linux/OpenWRT. We also want to explore a more refined application delivery system, inspired by the app store model, for customers to use fine grained (in time) services through micro transactions. We are currently interacting with several ISPs to deploy our framework and get experimental results over a production network.

Acknowledgements

We would like to thank Zainab Abaid, Naveed Bhatti, Osama Haq, and the anonymous reviewers at CCR for their feedback that has improved the quality of this manuscript considerably. This work is supported by a grant from the Pakistan National ICT R&D Fund.

7. REFERENCES

[1] Theophilus Benson, Aditya Akella, and Aman Shaikh. Demystifying configuration challenges and trade-offs in network-based ISP services. In *Proceedings of the ACM SIGCOMM 2011 conference*, SIGCOMM '11, pages 302–313, New York, NY, USA, 2011. ACM.

[2] Kenneth L. Calvert, W. Keith Edwards, Nick Feamster, Rebecca E. Grinter, Ye Deng, and Xuzi Zhou. Instrumenting home networks. *SIGCOMM Comput. Commun. Rev.*, 41(1):84–89, January 2011.

[3] Marshini Chetty, Richard Banks, A.J. Brush, Jonathan Donner, and Rebecca Grinter. You're capped: understanding the effects of bandwidth caps on broadband use in the home. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '12, pages 3021–3030, New York, NY, USA, 2012. ACM.

[4] Andrew D. Ferguson, Arjun Guha, Chen Liang, Rodrigo Fonseca, and Shriram Krishnamurthi. Participatory networking: an API for application control of SDNs. In *Proceedings of the ACM SIGCOMM 2013*, SIGCOMM '13, pages 327–338, New York, NY, USA, 2013. ACM.

[5] Aaron Gember, Robert Grandl, Junaid Khalid, and Aditya Akella. Design and implementation of a framework for software-defined middlebox networking. In *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, SIGCOMM '13, pages 467–468, New York, NY, USA, 2013. ACM.

[6] Kamran Khan, Affan Syed, and Ali Khayam. Traffic analyzer for differentiating bittorrent handshake failures from port-scans. `http://arxiv.org/abs/1309.0276`, 2013.

[7] Teemu Koponen, Martin Casado, and Natasha Gude et al. Onix: a distributed control platform for large-scale production networks. In *Proceedings of the 9th USENIX conference on Operating systems design and implementation*, OSDI'10, pages 1–6, Berkeley, CA, USA, 2010. USENIX Association.

[8] Pakistan TeleCom Ltd. `www.ptcl.com.pk`.

[9] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. Openflow: enabling innovation in campus networks. *SIGCOMM Comput. Commun. Rev.*, 38(2):69–74, March 2008.

[10] Syed Akbar Mehdi, Junaid Khalid, and Syed Ali Khayam. Revisiting traffic anomaly detection using software defined networking. In *Proceedings of RAID*, RAID'11, pages 161–180, Berlin, Heidelberg, 2011. Springer-Verlag.

[11] Nayatel. `www.nayatel.pk`.

[12] Radhika Niranjan Mysore, Andreas Pamboris, Nathan Farrington, Nelson Huang, Pardis Miri, Sivasankar Radhakrishnan, Vikram Subramanya, and Amin Vahdat. Portland: a scalable fault-tolerant layer 2 data center network fabric. In *Proceedings of the ACM SIGCOMM 2009 conference on Data communication*, SIGCOMM '09, pages 39–50, New York, NY, USA, 2009. ACM.

[13] Vern Paxson. Bro: a system for detecting network intruders in real-time. In *Proceedings of the 7th conference on USENIX Security Symposium - Volume 7*, SSYM'98, pages 3–3, Berkeley, CA, USA, 1998. USENIX Association.

[14] BT NetProtect Plus. `http://bit.ly/156F5yV`.

[15] Zafar Ayyub Qazi, Cheng-Chun Tu, Luis Chiang, Rui Miao, Vyas Sekar, and Minlan Yu. Simple-fying middlebox policy enforcement using sdn. In *Proceedings of the ACM SIGCOMM 2013 Conference*, SIGCOMM '13, pages 27–38, New York, NY, USA, 2013. ACM.

[16] M. Roesch. Snort - lightweight intrusion detection fornetworks. In *Proceedings of USENIX LISA'99*, 1999.

[17] Justine Sherry, Daniel C. Kim, Seshadri S. Mahalingam, Amy Tang, Steve Wang, and Sylvia Ratnasamy. Netcalls: End host function calls to network traffic processing services. Technical Report UCB/EECS-2012-175, EECS Department, University of California, Berkeley, Jul 2012.

[18] Juniper Networks SRC. `http://juni.pr/1cQMNz5`.

[19] Srikanth Sundaresan, Walter de Donato, Nick Feamster, Renata Teixeira, Sam Crawford, and Antonio Pescapè. Broadband internet performance: a view from the gateway. *SIGCOMM Comput. Commun. Rev.*, 41(4):134–145, August 2011.

[20] Mukarram Bin Tariq, Murtaza Motiwala, and Nick Feamster. Nano: Network access neutrality observatory. In *In Proc. 7th ACM Workshop on Hot Topics in Networks (Hotnets-VII*, 2008.

[21] T-Mobile Bobsled ™. `http://www.t-mobilecleverconnect.com/`.

[22] Matthew Chalmers Beki Andrew Crabtree Tom Rodden, Paul Tennent. Homework: Developing a corpus of domestic network usage. In *CHI 2009 Developing Shared Home Behavior Datasets to Advance HCI and Ubiquitous Computing Research workshop*, 2009.

[23] UMobile TV. `http://bit.ly/nOMbXE`.