Neutral Net Neutrality

Yiannis Yiakoumis, Sachin Katti, and Nick McKeown

Stanford University

{yiannisy, skatti, nickm}@stanford.edu

ABSTRACT

Should applications receive special treatment from the network? And if so, who decides which applications are preferred? This discussion, known as *net neutrality*, goes beyond technology and is a hot political topic. In this paper we approach net neutrality from a user's perspective. Through user studies, we demonstrate that users do indeed want some services to receive preferential treatment; and their preferences have a heavy-tail: a one-size-fits-all approach is unlikely to work. This suggests that users should be able to decide how their traffic is treated. A crucial part to enable user preferences, is the mechanism to express them. To this end, we present *network cookies*, a general mechanism to express user preferences to the network. Using cookies, we prototype *Boost*, a user-defined fast-lane and deploy it in 161 homes.

CCS Concepts

•**Social and professional topics** → **Net neutrality;** •**Networks** → *Network architectures; Network protocols; Cross-layer protocols; Middle boxes / network appliances; Network economics; Network manageability; Home networks;*

1. INTRODUCTION

Net neutrality is currently a charged debate. The core of the argument is about which applications should receive special treatment from the network, and whether special treatment is in the best interest of users. Several examples of special treatment already exist, for example Facebook Zero [3] and T-Mobile's Music Freedom [8] programs promise that certain traffic is free from data caps (aka *zero-rating*). Similarly, others have proposed giving some traffic a fast-lane over the last mile. However these proposals have raised concerns [23, 13] that the consumer will ultimately be harmed. The fear is that ISPs and content providers will decide which applications get the best service, squeezing out new services, and creating an insurmountable barrier to entry for innovative new applications. Consequently, to protect users, net neutrality has devolved to *"don't do anything"*, i.e., treat all traffic the same.

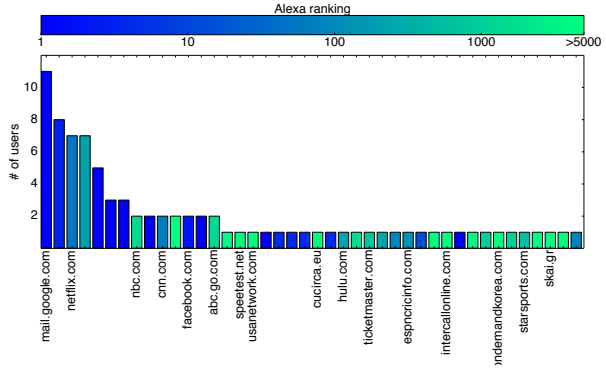


Figure 1: If given the choice, which websites would home users prioritize? The preferences have a heavy tail: 43% of the preferences are unique, with a median popularity index of 223.

We believe the debate is backwards. While many arguments are made in the name of protecting users, those very users do not seem to have a say in the matter. Specifically, if we are nervous that ISPs and content providers will make decisions detrimental to users, why not let users decide for themselves how their traffic is treated? Done right, this could potentially benefit everyone: Users get what they want, ISPs provide more value to their customers, and popular content gets preferential treatment, even if the service is small and new.

We are not the first to propose involving users more closely and ask them what they want—others have pointed out the importance of users during the recent update in FCC's Open Internet rules [31, 1]. In this paper we try to advance the net neutrality debate by

SIGCOMM '16, August 22 - 26, 2016, Florianopolis , Brazil

 ISBN 978-1-4503-4193-6/16/08…$15.00

DOI: http://dx.doi.org/10.1145/2934872.2934896

resolving two open questions related to users' preferences.

First, do users want a say over how their traffic is treated by the network? After all, users might prefer not to be bothered with such details, and just have the network treat all their traffic the same. To answer this question, we conducted two studies of user behavior. We prototyped a service—a user-defined fast lane called *Boost* and deployed it in 161 homes, during an internal "dogfood" test of OnHub, a commercial home WiFi router built by Google. With Boost, users can decide which traffic gets higher priority (or decide to not give higher priority to any traffic). They can express their preferences through a web browser extension, either by prioritizing a specific tab, or by always prioritizing the traffic from a specific website. Figure 1 summarizes the behavior of a fairly homogeneous[1] group of users: 43% of expressed preferences were unique, i.e., the preferred website was picked by only one user, while the median popularity index of prioritized websites was 223.[2] While our sample is small, it demonstrates the diverse and heavy-tailed nature of user preferences, and that users are willing to express them if it is easy to do so. To strengthen our results, we surveyed 1,000 smartphone users about their interest and preferences in fast lanes and zero rating services. Given the option to select which application to zero-rate, users chose 106 different applications from different categories (e.g., video, audio, social, news). Both studies demonstrate that user preferences have a heavy tail, and suggest that a one-size-fits-all approach is unlikely to work for most users.

This naturally leads to the second question we wish to resolve: Now that we know users have diverse preferences, we need to pick (or design) a mechanism to give them control over their preferences. In other words, how should users express their preferences to the network? Should users express their detailed preferences, or should they pick among popular applications shortlisted by an ISP? How can the network provide preferential treatment to an application, without the user having to reveal what the traffic contains? How can ISPs account and charge for offered services? How can we audit that the parties adhere to their agreements? And so on. All of these questions need to be addressed if we are to make such a system practical.

As we will show, existing mechanisms (like DiffServ, DPI, or even new SDN-like approaches) do not adequately address the concerns raised above. This was emphasized during the last FCC hearings, when policymakers rejected an AT&T proposal for user-driven services, concerned that the mechanism itself (or the lack of a proper one) could potentially undermine the desired outcome [1].

To tackle this challenge we present ***network cookies***—a mechanism for users to express their preferences to the network and to content providers. A network cookie—similar to HTTP cookies—is a small piece of data, that users can attach to their packets. A network with appropriate information can lookup the state associated with this cookie and apply the desired service. Network cookies provide a ***simple* yet *expressive*** mapping abstraction to users: we can use them to boost, zero-rate, or arbitrarily map any traffic to any state in the network, not just a few popular preconfigured applications. ***Cookies facilitate the tussle*** between users, applications and ISPs, and respect the trust relationships between different parties. They provide built-in authentication so that only authorized users can use a given service; revocability for users to easily change their preferences or completely withdraw from a service; they respect user privacy, as users do not have to reveal the content, type or origin of traffic getting special treatment; and they protect against replay and spoofing attacks preventing a third-party from replaying an overheard cookie. Perhaps most important, cookies are policy-free—in fact, an ISP could use cookies to prioritize a single content provider, all the way to let each user choose her own. By separating mechanism from policy, they enable a wide set of policies which can be easily adjusted, enforced, and verified according to trust relationships and regulatory frameworks. Finally, ***cookies can be practically deployed*** in existing networks. They are independent from packet headers and payload, we can match against them with high accuracy and regardless of content popularity, presence of encryption and middleboxes, or task complexity (e.g. we can boost a webpage, or a mobile application); they can be incrementally deployed by individual ISPs and applications; and we can leverage different transport mechanisms to carry them (e.g. a special HTTP header, a TLS-handshake extension, an IPv6 extension header).

This user-driven approach can enable a plethora of network services in the future. A video application could ask for a short burst of high bandwidth when it runs low on buffers (and risks rebuffering), while a researcher could do the same to upload a large file before an upcoming deadline. Users can pay per burst, or get a limited monthly quota for free. Moreover, cookies and user preferences are not bound to net neutrality—we can use them in other scenarios to let individuals customize network services for their own needs. This paper takes no position on what these services should look like, or how to implement them in the network. Our goal is to simply enable users to express their traffic preferences to the network.

Our main contributions are:

1. Network cookies: a policy-free mechanism to express

[1]Employees of the same company (Google), living in the same city.

[2]We use Alexa ranking as the popularity index.

which applications get special treatment from the network in a traffic-agnostic way (§4).
2. We advocate for a user-centric view on net neutrality and demonstrate that this is both practical and beneficial. Through a user study and an online survey we show that preferences have a heavy tail, suggesting that a one-size-fits-all approach is unlikely to work for many users (§2,§5).
3. Prototype services that use network cookies and user preferences, including a Boost fast-lane service, and AnyLink, a cloud-based version of Boost which provides slow (instead of fast) lanes that we make publicly available online (§5).

2. WHY A USER-DRIVEN APPROACH

Fast lanes and zero-rating services have been deployed in several countries, in cellular and last-mile residential networks. For example, Microsoft and Comcast joined forces in the US to offer a special service for Comcast content to Xbox consoles. Traffic went through a dedicated, high-bandwidth and data-cap free channel. Similarly, Netflix partnered with Australian ISPs to exempt their video traffic from a home user's monthly data-caps. MusicFreedom and BingeOn are services offered by T-Mobile in the US to exempt a handful of music and video services from monthly data caps. Spotify has similar partnerships with European mobile operators. Facebook-Zero and Wikipedia-Zero allow users in emerging markets to access Facebook and Wikipedia without a data-plan. The incentives for such services vary—some ISPs absorb the cost for special treatment to differentiate from competition and satisfy their customers, others (especially in emerging markets) do it in an effort to familiarize their users with the Internet, and in some cases ISPs directly charge content providers for special treatment.

These services are considered controversial as they limit user choice to a few pre-selected applications. They often lead to regulatory complaints and even withdrawal of the service due to public backlash [22, 4].

So, what do users really want? We asked 1,000 smartphone users their preferences on zero-rating through an online survey.[3] 65% of users expressed interest in a service that lets them choose one application that does not count against their monthly cellular data cap, or not even require a data plan, which explains why these services are being deployed. But when we asked them to choose a particular application, responses were heavy-tailed, and many users preferred websites and applications not available for special treatment by existing services. Users expressed preference for applications ranging from social networks, video, music streaming, messaging and VOIP, news sites, maps, games, and educational applications (Figure 2). Using the number of downloads in Google Play Store as a proxy for popularity, some users chose applications with 10^9 users (e.g. Spotify, Facebook), while others chose specialized applications with only a few thousands users (e.g. Indie 103.1, an app from a radio station with $< 50k$ users). Current zero-rating services only allow users to choose from among a small set of popular services; our survey suggests this is not what a majority of users want. For example, Wikipedia Zero covers only 0.4% of our users' preferences, and Music Freedom just 11.5%. We found similar results when surveying user interest in fast-lanes, and a music-only zero-rating service [12].

In summary, our results strongly suggest users are interested in services like fast-lanes and zero-rating, but each user would prefer to boost or zero-rate different applications. A natural question to ask is how to let users express their preferences to the network. We start with the question: Do existing mechanisms allow users to correctly express their preferences to the network?

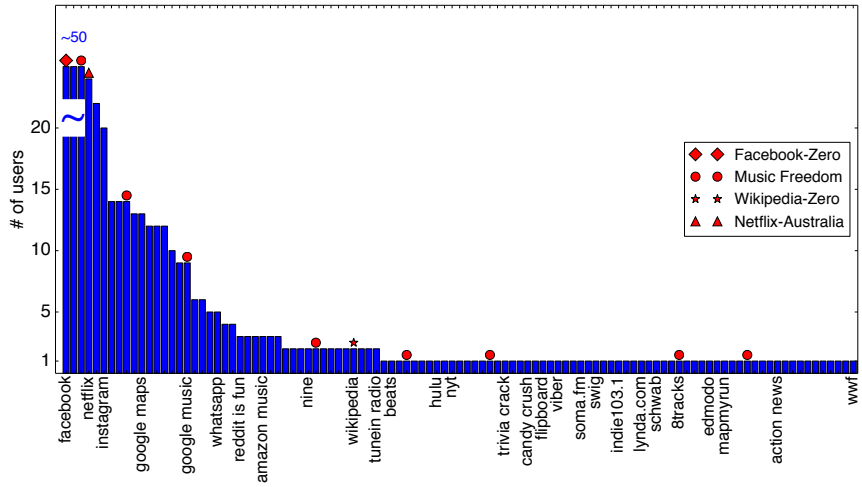
3. EXISTING MECHANISMS DON'T WORK

Ideally, an existing mechanism would let users express their preferences and meet our requirements. Unfortunately, none of the commonly used mechanisms pass muster. Let's look at the three most common ones.

Deep Packet Inspection (DPI). DPI is widely used to identify a subset of traffic (e.g., traffic coming from a specific content provider) and then apply a special service to it (e.g., higher bandwidth or zero-rating). DPI sits in a middlebox and typically matches traffic at line-rate, by examining IP addresses, TCP ports, SSL's SNI field, and packet contents. Typically, a new set of rules is added for each application and web-service.

DPI suffers from what we call a *high transaction cost*—adding a new preference (i.e., a new set of rules) is hard, particularly if the service is hidden inside https (which is increasingly common) or is hosted on a third-party CDN (e.g., video services hosted by Akamai). As a result, current DPI tools support only the most popular applications, and often require manual coordination between content providers and ISPs. For example, nDPI [17], a publicly available DPI system, recognizes only 23 out of 106 applications that our surveyed users picked for zero-rating. MusicFreedom, an existing zero-rating service implemented using DPI, works with only 17 out of 51 music applications mentioned in our survey [12]. Even when DPI detects an application, its view is very different from a user's view. Take `cnn.com` as an example. Loading its front-page generates 255 flows and 6741 packets from 71 different servers. nDPI marked only packets coming from CNN servers, which summed up to 605 packets (less than 10%)—everything else came from CDNs, advertisers, etc. But perhaps

[3]Users are aged 18-65 in the USA, surveyed via SurveyMonkey in August 2015.



Category	# of apps	Popularity	# of apps
AV Streaming	32	< 1M	16
Social	12	1M-10M	13
News	12	10M-100M	28
Gaming	9	100M-500M	14
Photos	4	> 500M	10
Email	4	N/A[a]	25
Maps	4		
Browser	3		
Education	2		
Other	24		

[a]Not listed in Google Play Store (e.g. iTunes, e-banking, XBox games).

Figure 2: *"If you could choose a single application (e.g. Facebook) to not count against your data caps (or not even require a data-plan), which one would you choose?"* Responses from 1000 smartphone users, and application breakdown by type and popularity. Preferences have a heavy-tail (106 apps in total), and they vary in type and popularity.

most problematic, DPI only works if a user is prepared to reveal to their ISP the service they are requesting special treatment for, which might hurt user privacy.

In summary, despite their wide deployment, DPI tools struggle to recognize many applications, have low accuracy, and do not respect user privacy. Their high transaction costs means new applications take a long time to add, and in the meantime, applications in the "tail" are not covered at all.

DiffServ. DiffServ allows endpoints to mark their packets (using the 6 DSCP bits in the IP header) and map them to a specific class in the network (e.g., high bandwidth, low latency). Although widely implemented in modern routers and operating systems, DiffServ cannot be practically used to express preferences to the network. Network operators often ignore or even reset DSCP bits across network boundaries, and popular platforms (like the Chrome browser or Android's SDK) do not allow developers or websites to mark their traffic.

While it is common to blame network operators or application developers for not respecting and/or not exposing the necessary APIs to mark the DSCP bits, we believe there are deeper limitations that make DiffServ insufficient for communicating user preferences.

First, the limited set of DSCP bits supports only 64 classes (2^6) and is already used internally by networks for their own purposes, leaving little room for customization. Expressing user preferences with DiffServ would i) require all networks, operating systems and applications to agree at a pre-defined meaning for each DSCP code (which has been proven impractical), ii) expect every element in the path to respect or at the very least not alter the DSCP marking, and iii) require networks to use a different mechanism for their own needs. Even then, the small number of available classes is restricting—ISPs cannot define their own custom classes, and if a packet crosses two networks there is no way to explicitly request special treatment only by one of them.

Second, and maybe most important, DiffServ has no authentication and revocation primitives: any application can set the DSCP bits and request service without the user's consent. Any developer can ask for special service even if it conflicts with user preferences, or—even worse—if it results in network charges for users, and users or operators do not have the means to easily revoke such access. Think of a legacy gaming console that opportunistically sets the DSCP bits for low-latency, at the same time that an operator charges for access to this class. To avoid charges a user would have to stop using the device, ask the console manufacturer to update the software, or configure her home router to reset DSCP bits coming from this device. Ideally we want to avoid these dependencies.

As a result, DiffServ is only suited as an internal marking mechanism, categorizing traffic into broad classes. It is not a practical way for a user to express her preferences to the network.

Out of band flow description. A more recent approach has been to leverage the flexible control plane of software-defined networks (SDNs), and expose an API for applications and users to express their preferences [32, 21]. In this approach, the application (or a user agent) tells the centralized control plane which flows to match on—via an out-of-band (OOB) channel—by describing which flows should get special treatment (e.g., using the 5-tuple). Subsequently, the control-plane programs the switches to match on these flows.

This approach has two main limitations. First, signaling user preferences through the relatively slow control plane can be expensive. Recall that the frontpage of CNN has 255 flows; sending each of them through

a centralized controller to reprogram multiple network switches is an expensive process. Second, giving a static flow description does not work when the flow changes—for example, as it traverses a NAT or is encapsulated. In a home network, the flow will change at the NAT module of the home router, making the 5-tuple description invalid for the head-end router. A workaround would be to describe a flow only with static fields, e.g., the server's IP address and port, but this causes false matches as a single server hosts content from several other applications. In §5 we quantify some of the limitations of DPI and OOB using example user preferences.

3.1 From Limitations to Requirements

The limitations of existing approaches motivated our search for a new mechanism that satisfies three fundamental requirements.

The proposed mechanism should be ***simple*** for users to understand and ***expressive*** enough to enable a variety of services (e.g., fast-lane, zero-rating and new services that come along), let users choose any application they like, and to express a complex set of changing user preferences (e.g., a website, or a mobile application).

It should ***respect the tussle*** between different stakeholders (i.e., users, the network, content providers and policymakers). In order to support many ways to express preferences, and different regulatory frameworks, the mechanism should be policy-free. It should not force a user to reveal which content they request special treatment for, and prevent unauthorized parties from requesting special treatment without the user's permission.

Finally, the mechanism should be ***practical to deploy***. It should not overly burden the user, the user's device or the network operator, and work well in the presence of CDN, NAT, and HTTPS. It should be incrementally deployable without requiring forklift changes in network or content provider infrastructure.

The next section introduces **network cookies**, a mapping abstraction that meets these requirements.

4. NETWORK COOKIES

Network cookies are a policy-free mechanism allowing users to express their preferences to the network and to remote service and content providers.

4.1 Cookies

A network cookie is a small piece of data that users attach to their packets. As a packet flows through the network, the network cookie communicates the user's preferences to the devices it encounters along the way, possibly all the way to the end host. Upon detection, the network and the end host lookup the cookie in a table to decide what service to apply to the packet (and possibly to all other packets from the same flow).

To prevent an unauthorized third-party from replaying or spoofing a cookie, each cookie is *unique*, *signed*, and can be *used only once*, in ways we describe below. One way to get these properties would be for the user to ask the network for a new cookie every time it sends a packet. Clearly this would be burdensome and slow. Instead, the user requests a *cookie descriptor* which is then used to locally generate multiple cookies. Periodically, the user gets a new descriptor from the network.

The workflow goes like this: The network advertises the special services it is offering on a well-known server; for example, it may advertise that it has cookies available to boost any website, or only cookies to boost Amazon Prime video. The user picks a *cookie descriptor* from the well-known server—the user might buy it, or be entitled to a certain number per month, via coupons, or on whatever terms the network owner decides. Once the user has a *cookie descriptor* she can use it to generate local cookies, and attach them to her packets. A cookie descriptor typically lasts hours or days, and is renewed by the user as needed.

To understand cookies, it helps to start by looking at the details of the *cookie descriptor*, from which cookies are generated.

```
struct cookie_descriptor {
 // a 64-bit value that identifies a cookie
 // descriptor and acts as a lookup key.
 uint64_t cookie_id,
 // Shared key used to sign a cookie.
 char * key,
 // Service data identify the network service
 // the packet should receive. It can be just
 // the name of the service (e.g., 'Boost'),
 // or any other information.
 char * service_data,
 // An optional list of attributes that chara-
 // cterize a cookie descriptor (e.g. when and
 // how to use it, whether it is valid or not).
 char * attributes[]
};
```

Listing 1: Cookie Descriptor

From this descriptor, the local host generates cookies with the following fields:

```
struct cookie {
 // 64-bit id copied from the descriptor.
 uint64_t cookie_id,
 // Universally unique identifier for this cookie.
 uuid_t uuid,
 // The time a cookie was generated. Limits the
 // duration a cookie is valid, prevents reuse,
 // and reduces state kept by the network.
 uint64_t timestamp,
 // Message Authentication Code that verifies a
 // cookie and prevents spoofing.
 char * signature
};
```

Listing 2: Cookie

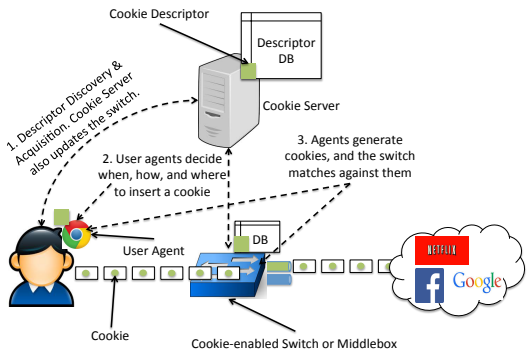


Figure 3: Workflow for a cookie-enabled service.

```
def generate_cookie(descriptor):
  value = descriptor.id + uuid() + now()
  digest = hmac.digest(descriptor.key, value)
  return value + digest
def match_cookie(cookie):
  cookie_desc = cookie_descriptors[cookie.id]
  if (!cookie_desc ||
      !cookie_desc.is_valid_sig(cookie) ||
      !cookie_desc.is_unique_uuid(cookie.uuid) ||
      abs(cookie.timestamp - now()) > NCT):
    return None
  cookie_desc.append_cookie(cookie.uuid)
  return cookie_desc
```

Listing 3: Pseudocode for cookie generation and matching.

4.2 Cookie Workflow

Let's take a closer look at how users express their preferences via the cookie mechanism, and the pieces we need to make it all work. Figure 3 shows the main architecture.

Components: The first component is the *user-agent* which has a GUI for the user to express her preferences. In the background it interfaces to the network to discover and acquire cookie descriptors. The user-agent also generates and adds cookies to packets. User-agents can be integrated into an OS, browser, or in the application itself (e.g., a video player or a VOIP client). The second component is the *well-known server* where users go to acquire cookie descriptors. Finally, the *network switches* (or middle boxes) that match against cookies and apply the right service.

Workflow: Getting and using a cookie descriptor happens in three stages:

1. **Cookie Descriptor Discovery and Acquisition**. Users and their clients learn of network services through standard discovery protocols (DHCP, mDNS) or it can be hardcoded in the application (e.g. Amazon Prime Video might know where to get special Amazon cookies). Once found, the descriptor is downloaded over an (optionally authenticated) out-of-band mechanism (e.g., a JSON API). For example, in a home network anyone who can talk to the AP might get a cookie, while a cellular network might require users to login first.

2. **Cookie Insertion:** There are several ways to use a cookie. First to consider is *when* to use a cookie (and the associated service). This can be explicitly requested by the user, or assisted by an application (e.g., a video client can ask for extra bandwidth if its buffer runs low). Next to consider is *where* to add the cookie to our packets. We suggest supporting multiple choices; we can add it at the application layer (as an http header for unencrypted traffic or a TLS handshake extension for https traffic [10]); at the transport layer (TCP long options [9], integration with QUIC, or a custom UDP-based header); or at the network layer (IPv6 extension header). Choosing the right layer depends on the applications and network services involved.

3. **Cookie Generation and Matching:** The last step is to actually generate and match against cookies. Generation is easy from the cookie descriptor, and the cookie is added to an outgoing packet. The network detects a cookie and verifies that it is valid by checking that (i) the cookie ID is known, (ii) the MAC digest matches, (iii) the timestamp is within the "network coherency time", and iv) that we haven't seen the cookie before. The network coherency time (NCT) is the maximum time we expect a packet to live within the network, and is set to 5 seconds. To verify uniqueness, we keep a list of recently seen cookies (within NCT). If a cookie is valid, the network applies the service directly (e.g., sends the packet through a high-priority queue). Alternatively it can mark the DSCP bits to enforce the service elsewhere in the network. If it fails to match, it behaves as if the cookie was not there, offering default services. We show sample code for generation and matching in Listing 3.

4.3 Cookie Attributes

Some network services will need more information to be supplied with a cookie. For example, a cookie might only be valid when the user is connected to a specific WiFi network, or in a specific geographic area, or in a specific network domain. Cookies therefore carry unformatted, optional attributes, added by the network or end host, to provide more service-specific information.

We expect some cookie attributes to become common-place, such as

• **Granularity:** A cookie can be applied to a packet or a flow. By default, a cookie characterizes the flow (5-tuple) that a packet belongs to. If set, granularity lists the header fields that compose the flow described by this cookie (e.g. 5-tuple for a TCP flow), or limit it to this packet only. We can also define whether the cookie applies the desired service for the reverse flow.

• **Shared:** When set, the cookie descriptor can be shared between multiple endpoints. In a home network,

the home router might acquire a descriptor from the ISP and then act as a cache, sharing it with devices connected to the home network.

• **Acknowledgment Cookie:** When set, the remote server is expected to send an acknowledgment cookie with the response. Acknowledgment cookies can be used for different reasons, like setup necessary state on the reverse path, or verify that the server received the cookie from the user. A server could just playback the original cookie sent by the user, or generate and send a new one (assuming the user shared the cookie descriptor with the server).

• **Network delivery guarantees:** When set, the network is expected to send an acknowledgment cookie with the reverse traffic to acknowledge that it received and acted upon a cookie sent by the user. It is similar with the previous attribute, but instead of the server, it is the network that acknowledges receiving the cookie. Guarantees can be useful when cookies are ignored (e.g., a bug in the client that creates an erroneous cookie, or a temporary loss of state in the network)—for example a video player could notify the user that she will be charged for the upcoming video. Network delivery guarantees assume that the network can modify traffic between endpoints to add a cookie, and therefore depend on the transport protocol (e.g., HTTP and IPv6 work fine, while SSL/TLS prevents third parties from modifying traffic between endpoints. New protocols (like mcTLS [26]) enhance SSL to allow middleboxes to change traffic between endpoints in a trusted way.

• **Cookie Transport:** A list of protocols over which a cookie can be carried (e.g., HTTP, TCP, IPv6, TLS).

• **Expiration:** Until when a cookie descriptor is valid. Expiration dates can be used to revoke a service, and also limit the risk of descriptor leakage.

4.4 Putting Everything Together

As a concrete example, consider an ISP that offers its customers a *fast-lane* for their high priority traffic. The home AP discovers that cookie descriptors are available at *http://cookie-server.com* (through the DHCP lease from the user's ISP). The AP connects to the server with the user's credentials and acquires a cookie descriptor, which is valid for one week. A browser extension goes to *http://192.168.1.1/getcookies*, discovers that a fast-lane is available in the network, and highlights a button which boosts a given website when clicked. The extension uses the cookie descriptor to add cookies to outgoing packets.

4.5 Cookie Properties

Cookies are designed to meet three high-level requirements: (i) they are *simple* for users to understand and *expressive* enough to enable a variety of services and user preferences; (ii) they *respect tussles* between users, applications, and ISPs, and allow for different outcomes by separating the mechanism (the cookie) from the policy (the preference); and (iii) they can be *deployed* in today's infrastructure without requiring forklift changes.

To meet these requirements, cookies exhibit specific properties which we describe below. Table 1 provides a summary and compares cookies with alternative mechanisms to express user preferences.

Cookies are simple yet expressive: Cookies are conceptually simple for users to understand: "I insert a cookie in my packets to make specific traffic faster". They are not tied to a specific network service and can be used for QoS, zero-rating, or generally for *linking arbitrary traffic to arbitrary state and processing* (e.g., consume a network service, identify a user or act as authentication credentials). Cookies have *low transaction cost* and can express the heavy tail of user preferences, not just a list of a few popular applications. They can also express *high-level and complex demands*, like prioritizing a webpage or mobile application. They are *composable*—users can combine multiple services (potentially by different networks) by composing multiple cookies together. Finally, cookies can be cleanly *delegated* by the users to either the content provider or some third party that figures out how to best use cookies for them. Specifically, users can choose to share their cookie descriptors with their desired content providers who in turn can generate cookies on their behalf and apply them to the downlink content. Delegation still keeps the users in control while respecting any tussle boundaries between content providers and ISPs.

Cookies respect tussles: Cookies respect the trust relationships between users, applications and ISPs in terms of privacy, accountability and authentication. They are unique and can be used only once, *preventing an unauthorized party from spoofing or replaying* an already used cookie. Cookies are signed and provide *built-in authentication* during the descriptor acquisition phase, which means that only authorized users can generate a valid cookie. *User privacy is respected*: the network does not need to know what the traffic is that the cookie is attached to, allowing, for example, a user to direct a video flow to the fast-lane, without revealing the content provider or even the fact that it is a video flow.[4] Cookies are also *revocable by both parties:* when users want to stop using a service, they just have to stop adding a cookie to their traffic, or ask the network to invalidate a descriptor (in case they cannot control the application); the network can similarly stop matching against a cookie to stop offering a service. Revocability

[4]By respecting user privacy we mean that use of cookies does not require users to reveal what is their preferred content to the network. They do not add any further protection for information already exposed, such as destination IP address and port.

is also helpful in case a descriptor gets leaked or an application gets compromised. Finally, cookies are *policy-free*: they separate mechanism from policy and enable different outcomes in tussles depending on trust relationships and regulatory frameworks between the different stakeholders. At the same time they are *easily auditable*—interested parties can monitor what traffic gets special treatment by the network just by looking at who gets access to cookie descriptors and how.

Cookies can be practically deployed: Cookies are *separate entities from the traffic itself* (header, payload and path). Therefore they are not affected by encryption (https), service co-hosting (CDN and cloud-based infrastructure), or packet-mangling middleboxes such as NAT. The network can capture complex contexts (e.g., a webpage, a mobile application) with *high accuracy*. They can be used in *multiple transport layers* (e.g., HTTP header, TLS handshake extension, IPv6 header). They can also be incrementally deployed by changing only the client and the network—we do not depend on servers to recognize or act upon cookies.

In the simplest scenario (e.g., charging, QoS over the last-mile), we can detect cookies and enforce the service for a flow in both directions using a single box. For more involved deployments, cookies do not need to be deployed in every switch/router in the network; an ISP can look up cookies at the edge (e.g., as a Virtualized Network Function) and then use an internal mechanism to consume a service within the network (e.g., DiffServ or FlowTags [19]). Because cookies are composable, we can incrementally use services from different network providers. For example, a videocall between two users could use two cookies to get sufficient bandwidth at both access networks, without requiring any coordination between the two network operators. Acknowledgment cookies expand cookies functionality to simplify service enforcement for *reverse flows*, especially for assymetric paths: we can ask the server to bounce back the cookie we sent, or generate a fresh one from a delegated descriptor and send it along with the downlink flow. Cookies also fail gracefully; when the network fails to match or verify a cookie, it can default to best-effort services. Furthermore, we can enhance the cookie workflow with *optional network delivery guarantees*. When the network detects a cookie, it generates an "acknowledgment" cookie from the same descriptor, and attaches it to the response. If the client doesn't receive an acknowledgement cookie, it shows an alert to the user asking whether she wants to continue nevertheless with best effort service.

While expressive, the separation of cookies and cookie descriptors results in a low overhead mechanism. Cookie descriptors are exchanged over a slow and less dynamic control plane. Therefore we can exchange arbitrary length state without worrying about overhead, add authentication primitives to protect acquisition of cookies and provide accountability, and add attributes to further define their use. Cookies themselves are tailored for dataplane use. They are generated, inserted, and matched locally on a straight-forward way, they are unique to protect against replay attacks, and carry cryptographic primitives to ensure integrity, authenticity and delivery guarantees for security.

4.6 Deployment Considerations

Deployment considerations for a cookie-based service will depend on the network and type of service to be offered (e.g., last-mile QoS, zero-rating).

To better understand scalability concerns, we built a cookie-based zero-rating middlebox on top of Click and DPDK [25, 2] using an off-the-shelf server.[5] Our middle-box keeps two counters per IP address (one for free and another for charged data), and enforces the service in software for both directions of a flow. Cookies are embedded as a special HTTP REQUEST header for HTTP traffic or TLS ClientHello extension for HTTPS. We expect this NFV-like approach to be common for services like zero-rating and last-mile QoS deployed in edge networks, and is on par with architecture trends in ISPs and mobile carriers.

For a given packet our middle-box has to perform one of three tasks: i) search for a potential cookie (first 2-3 packets of every flow), ii) search and verify a cookie (a packet that contains a cookie) or iii) simply map a packet to a given service (for a flow already updated in our system). As such, performance will depend on traffic parameters such as the number of packets per flow, or new flows per second.

We evaluated performance against a 15-hour anonymized trace that includes all wireless traffic from our university's main campus, student residences, and visitor WiFi. It contains 11.3 million HTTP(S) flows originating from 73613 distinct IP addresses (median flow size is 50 packets, and 99-percentile for new flows per second is 442).[6] We connected our middlebox with a MoonGen packet generator [18] which sends flows with cookies and monitors how fast our middlebox can forward packets. Assuming 50-packet flows, 100K cookie descriptors, and a cookie for each flow, our middle-box was able to saturate a 10Gb link with 512-bytes packets (~48000 new flows per second), much more than required by the university trace. Performance drops below line-rate for smaller packet or flow sizes (Figure 4).

There are certain steps to further scale our system. First, we can use multiple cores instead of one, and sim-

[5] 8-core Intel Xeon @ 2.60GHz, 128GB of RAM, 2 10Gb NIC.

[6] The trace was collected on Jan. 26th 2015 from 9am to 11:59pm.

		Cookies	DPI	OOB	DiffServ
Simple & Expressive	arbitrary traffic ↔ arbitrary state	✓	×	✓	×
	Low transaction cost	✓	×	✓	✓
	High-level preferences	✓	×	✓	✓
	Composable	✓	×	✓	×
	Delegetable	✓	×	✓	×
Tussle Aware	Protection from replay, spoofing	✓	✓	×	✓
	Built-in Authentication	✓	×	✓	×
	Respect Privacy	✓	×	✓	✓
	Revocable	✓	×	✓	×
Deployable	Independent from headerspace, payload, path	✓	×	×	×
	High Accuracy	✓	×	✓	✓
	Multiple transport mechanisms	✓	×	×	×
	Low overhead	✓	✓	×	✓
	Network Delivery Guarantees	✓	×	✓	×

Table 1: Network Cookies properties and comparison with alternative mechanisms to map traffic to a network service.

ilarly add more than one middle-boxes to scale-out the deployment, along with a load-balancer that shares the traffic among servers. A similar approach for software-based NAT was demonstrated to scale up to 40Gb/s with six commodity servers [27]. The main challenge to scale out cookies in a distributed deployment comes from verifying uniqueness as cookies from the same descriptor might appear in different places (a problem known as double-spending in digital cash schemes). We can relax uniqueness verification in certain cases—for example an ISP can ensure that all cookies from a specific descriptor always go through the same middle-box where uniqueness can be locally verified. An in-depth exploration of potential risks and methods to verify uniqueness on a distributed deployment is left for future work.

We now discuss some variations of the above deployment scenario.

Proxy-Mode: Instead of deploying cookies in-band with traffic of interest, cookies can also operate in proxy mode, i.e., co-located with a web proxy through which clients send their traffic. This can ease deployment (e.g., the service can be deployed in existing datacenters) and is particularly interesting for usecases where proxying overhead is not critical (e.g., zero-rating for cellular networks). AnyLink (§5) operates in proxy mode to emulate slower links for application developers.

Cookie→DSCP mapping: Service enforcement does not have to be co-located with cookie inspection. The ISP can look up cookies at the edge, and then use an internal mechanism to consume a service within the network (e.g., DiffServ, MPLS, or QCI for LTE networks) without requiring all switches to support cookies. In essence, cookies enable users to express their preference and communicate it to the network in a trusted and revocable way, independent of the path. The network can then use a different mechanism to interpret and realize this preference internally.

Packet-based cookies: Transport protocols that guarantee a cookie is contained within a single packet (e.g., IPv6 extension header, QUIC) can further improve performance. First, they require less state, as we don't need to reassemble part of a flow before processing a cookie. In the extreme, if every packet carries a cookie, flow-related state is eliminated (in the expense of bandwidth overhead and higher matching rates). Packet-based cookies can facilitate better hardware support which follows next.

Hardware support for cookies: Processing cookies will most likely take place in software, as current equipment does not not support HMAC-style verification or direct state setup for reverse flows. However, configurable hardware [15] and hardware-software coordination can still be beneficial. The hardware could detect and forward to software only packets that contain cookies, avoiding the extra overhead for all other packets. It could further verify the timestamp and look the cookie id against a table of known descriptors, further reducing the amount of packets that need to go to software. Discussions with hardware vendors implied that these capabilities are available today in modern hardware.

All in all, while deployment details will depend on the actual service and network under consideration, cookies seem practical for a wide variety of existing usecases. Furthermore, existing trends like Software Defined Networks, programmable hardware, and Network Function Virtualization will further improve their applicability and performance.

5. BOOST: A USER-DEFINED FAST LANE

To understand how users would like to customize their network, we prototype and deploy a service called *Boost*, which allows users to decide which traffic will

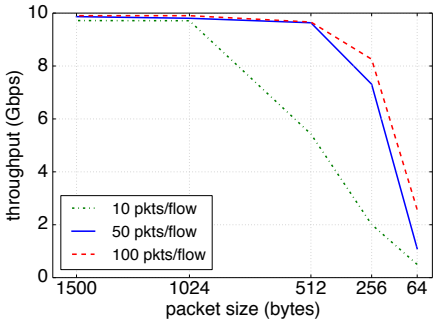


Figure 4: Matching performance for a Click-DPDK based cookie middlebox. Our prototype can provide 10Gb/s line-rate for 50 packet flows and 512-bytes packets using a single core; and process all wireless traffic from a university campus.

travel in the "fast lane" to their home network. Boost is very simple: it sends fast-lane traffic through a high priority queue, and occasionally throttles non-fast-lane traffic; it was designed to let us study user preferences rather than as a production-ready service. Boost was deployed in 161 homes as part of early testing for Google's OnHub home WiFi router. Interested readers can access sample code and try a cloud-based version of Boost which provides slow (instead of fast) lanes at http://anylink.stanford.edu.

Boost consists of two elements, a daemon and server running on the home access point, and a user-facing agent implemented as a Chrome browser extension.

5.1 Boost Agent

The agent is a Chrome browser extension, and lets users decide which traffic to boost in the following ways.

Boost a tab. All traffic from/to a specific tab is boosted. The user initiates this once per tab, and it lasts until she closes the tab (or after an hour).

Always Boost a website. Traffic related to a website gets priority.[7] The setting is remembered; whenever a user visits this website the agent informs the AP for related flows.

The browser provides an interesting vantage point. Users can boost any traffic they like, not just a short-list of popular applications; and they declare their preferences in an intuitive, easy-to-understand way: they identify webpages they would like to boost. While easy from the browser's vantage point, it is much more complicated if viewed from the network; all it sees is a large number of flows being started. For example, in the (otherwise simple) task of loading the frontpage of *cnn.com*, the browser starts 255 flows to 71 different servers. This highlights an interesting paradox: what is simple and meaningful for the user (e.g., a webpage, a mobile app) can be very complex for the network to detect; what is easy for the network (e.g., the IP of a server or a specific flow) is often meaningless for the users. Cookies with support from an agent can bridge the gap between users and networks.

Focusing on web traffic and the application layer gives us an easy place to start studying users' preferences, and a relatively easy deployment path. We add cookie-related functionality without requiring any kernel, server, or protocol support, and we can develop and deploy actual services on top of the Chrome browser which work with standard HTTP(S) traffic. Being close to the user, preferences can also capture user context: they can relate to content, as traffic is still unencrypted; or take into account the active tab of the browser; and they can be further enhanced by applications: a user preference can be combined with a trigger from a video client running low in buffer. These properties are not browser specific, but hint towards a more generic design choice: *should we place an agent closer to users and applications, or follow a more network-centric approach (e.g., place the agent at a network gateway)?* We choose the former, as it can capture user preferences and user context in a much better way.

We insert cookies as a special HTTP header for unencrypted traffic, and as a custom TLS extension (in TLS ClientHello messages) for HTTPS traffic. To better adjust with TLS and HTTP, we send a base64-encoded text cookie. We intercept outgoing `http(s)` requests using a Chrome API, extract related metadata (e.g., which tab generated it, the url in the address bar), and if it matches with user preferences we add a boost cookie to it. While adding an `http` header is straightforward, to add a TLS extension we had to modify Chrome's SSL/TLS library.[8]

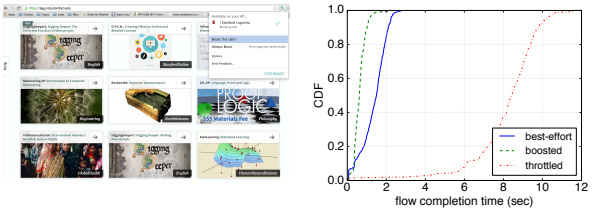
To start boosting traffic, the agent issues a boost request to a well-known server using a JSON message. The server responds with a boost cookie descriptor. A boost event (and the related cookie descriptor) expires by default after one hour. To resolve conflicts when multiple clients want to boost within a household, we have a *last one wins* policy, and expect users to resolve conflicts at a human level, if this is not enough.

5.2 Boost Daemon and Cookie Server

We keep cookie descriptors at a server already known to our Boost agents. We store them in a persistent SQL database and expose a JSON API for users to acquire them. We implement a python-based daemon on the WiFi router which sniffs traffic, looks up cookies and enforces the desired QoS service. Our daemon sniffs the first 3 incoming packets for each flow; if it detects a cookie, it tries to match the cookie against a known descriptor and verifies its integrity. If this is successful,

[7]We define a website by the domain at the browser's address bar, and boost all flows generated within this tab.

[8]Chrome uses BoringSSL, a fork of OpenSSL.



(a) (b)

Figure 5: a) User interface for Boost. b) Flow completion time for a 300KB flow in the presence of background traffic.

it adds this and the reverse flow to the fast lane using a set of standard tools in the WiFi router (iptables and Linux `tc`).

To provision the path for boosted traffic we i) use the high-bandwidth wireless WMM queue, and ii) throttle other traffic to ensure certain capacity for boosted traffic through the last-mile connection. The actual throttling rate depends on the capacity of the WAN connection which we estimate using periodic active tests. Figure 5(b) shows a scenario for a $6Mbps$ connection, where we throttle non-boosted traffic to $1Mbps$.

We should emphasize that our Boost prototype is far from perfect; for example, it is not work-conserving if the user does not use the fast-lane. This reflects our goal to understand user's preferences, rather than build the perfect service. We plan to improve the Boost prototype and install into more WiFi routers.

5.3 Measurements of Users' Preferences

Our first version of Boost, which uses an out-of-band API to communicate to the WiFi router, was made available to 400 home users, during an internal "dog-food" test of the OnHub home WiFi router. 161 users (40%) installed the extension in their browsers. Figure 5(a) shows a screenshot from the Boost extension while navigating to an online educational platform.

Figure 1 shows the websites prioritized by users, the number of clients that boosted a given domain, and their popularity. Many users boosted popular US video websites (Netflix, YouTube, HBO, ABC, Fox, and ESPN). The tail also includes less popular sites, such as a VoIP service, on-demand video services from several countries, as well as a website for ticketing auctions, where a few milliseconds might help secure a ticket for a popular event. Informal discussion with users also pinpointed interesting usecases: one user had a slow Internet connection ($3Mb/s$) and occasionally wanted to dedicate all resources to a specific task (stream a video); another wanted to prioritize business-related calls; and a third one wanted to prioritize Netflix on his TV, but not Netflix on his kids tablets.

5.4 Accuracy when boosting with cookies

Our prototype lets us check if cookies will boost the correct websites; and whether they would have been correctly boosted by alternative implementations that do not use cookies. As an example, we examine three preferences from our users (youtube.com, cnn.com, and skai.gr, a Greek media site). Navigating to the front-page of each site generates 80/3750, 255/6741, and 83/1983 flows/packets respectively. As shown in Figure 6(a), using cookies and our Chrome agent, we boost $> 90\%$ of traffic in all three cases. Our agent misses DNS requests and traffic prefetched by Chrome.

Next we compare cookies with a DPI-based design, to see if it could correctly identify and boost the same three websites. We use nDPI [17], a publicly available DPI system which can detect more than 220 popular applications, protocols, and websites. We ask nDPI to recognize and boost the websites, based on the traffic it sees in the network, then check to see if it identifies them correctly. DPI correctly identified only 18% of the traffic when we tried to boost cnn.com, and failed to detect any traffic for the Greek media site as it had no rules for it. Moreover, nDPI occasionally matched the wrong packets (false positives). When trying to match `youtube.com` it also matched 12% of packets from skai.gr, as it embedded YouTube's video player.

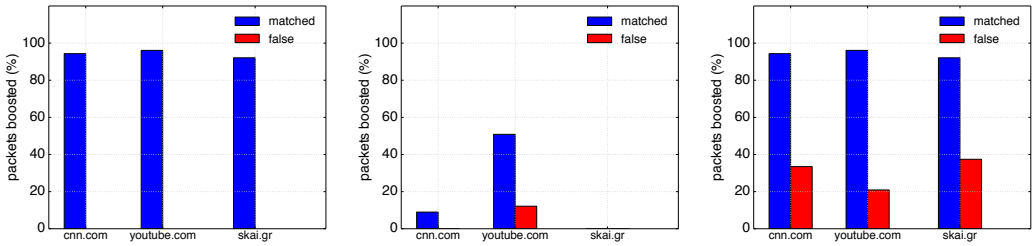
Finally we compare with an out-of-band (OOB) mechanism that sends a flow description to a central controller, asking it to boost traffic matching the rule. OOB detects the same flows with cookies, as they both detect traffic in the browser (Figure 6(c)). But it suffers from false positives. To make a flow description valid across a NAT we can only use the destination IP and port. This leads to 40% false positives in our example, as a major share of the traffic comes from the same servers (e.g., CDN, ads, social sharing plugins).

6. DISCUSSION

Differentiated network services are subject to multiple factors, such as stakeholders economic incentives, regulatory frameworks, industry competition, user familiriaty with Internet services, and the underlying technology. In this section we discuss the role of network cookies in the wider ecosystem.

Cookies are policy-free: They don't dictate what the policy is and can enable different outcomes, all the way from user-driven services to ones where an ISP can handpick a single service to differentiate. Their main value comes from streamlining the process for certain traffic to get special treatment: All we need to do is decide who gets access to cookie descriptors and how.

First, cookies enable new policies that are not available with existing technologies. This paper discussed one of them in depth: User-driven services. Others are also possible. For example, because cookies are independent from the traffic itself, a third party (other than the content provider or ISP) can pay for delivery of ar-



(a) Cookies (b) nDPI (c) Out-of-band

Figure 6: Matching accuracy for three sample user preferences. DPI cannot capture the complexity of a website, and fails to match any traffic from a less popular site. OOB can match a website with high accuracy, and respects the heavy tail of user preferences, but doesn't work well when flows change (e.g., by NAT or encapsulation). Using coarse granularity descriptions leads to false positive matches. Cookies can serve any user preference with high accuracy and no false positives.

bitrary content (e.g., a school or non-profit could subsidize the cost of data delivery for certain educational videos).

Second, they can significantly accelerate and streamline service provisioning, as any traffic can be easily differentiated, even in relatively complex scenarios (e.g., a video stream, with a side-chat service, and video advertisements from a third-party ad-network). Streamlining how traffic is mapped to different network policies not only reduces overhead for ISPs and content providers, but can also lead to services which are more inclusive, transparent, and easily auditable. This can be very important, especially in an environment of mistrust among different stakeholders.

Take T-Mobile's Music Freedom program as an example.[9] T-Mobile claims that any licenced music streaming provider is eligible to participate at its zero-rated program at no cost. But why are many services excluded? After two years of operations and seven service expansions, Music Freedom included 44 out of more than 2500 licenced online radio streaming stations [8, 6]. When we run our online user survey (August 2015), Music Freedom included only 17 out of 51 unique music applications listed by our respondents. We believe that this is related with the manual and involved technical process to add new participants in Music Freedom.[10] SomaFM, a popular online radio station, spent 18 months to become part of the program, having been first ignored by T-Mobile then confronted with technical limitations [28]. We experienced something similar: We worked with RockRadio.gr, a small regional radio station to help them participate in Music Freedom, but after three e-mails to the designated address and several months we heard no reply from T-Mobile. These incidents not only limit user choice, but can dramatically affect competition between services within the same category.

Small providers complain that T-Mobile ignores them because they are not big enough; T-Mobile claims that it is open to everyone and uses technical limitations to explain delays; and the FCC wants to investigate on a per-case basis, but it does not have the means to do so.

Cookies remove technical barriers and make the process straightforward—all an ISP has to do is give each content provider a cookie descriptor. This makes it easier for both ISPs and content providers to coordinate. Regulators can efficiently audit if involved parties play fairly. The FCC could demand that T-Mobile maintains a public database with the dates for all cookie descriptor requests, and it should be obliged to provide the descriptor to eligible parties within three days. This is similar to the FCC's "local number portability rules," which requires phone companies to complete the transfer of a phone number from one company to another within one business day from user's request [20].

7. RELATED WORK

Previous work in research and commercial products highlighted the need to give users and applications control over network functions, both in home and enterprise networks [32, 24, 29, 21, 5]. They all use derivatives of DPI and OOB techniques, and as such inherit the limitations we described in §3. FlowTags [19] use DSCP bits to co-ordinate with middleboxes and facilitate enforcement of network services in the presence of NAT and similar functions. Like DiffServ, they lack authentication primitives, support only up to 64 tags, and require full control of the path, which make them better fit for enterprise networks instead of functionality across network boundaries.

The advent of HTTPS emphasized the limitations of DPI and spurred interest in how to communicate between endpoints and middleboxes while preserving end-to-end encryption. mcTLS [26] extends TLS to allow endpoints to incorporate trusted middleboxes into secure sessions, while SPUD [7] proposes a new UDP

[9]Music Freedom represents a broad range of class-based services. Similar ones have been explored for video, VoIP, healthcare and educational services.

[10]According to anecdotal sources, T-Mobile uses DPI techniques to detect eligible traffic for Music Freedom.

transport layer that creates a "tube" to group multiple subflows between two endpoints. They both provide rich, bidirectional communication between endpoints and the network. In contrast, cookies provide a thinner interface (i.e., a mapping abstraction), but at the same time are much simpler to deploy: they work along with multiple existing protocols (e.g., HTTP(S), UDP, TCP), they can be incrementally deployed without modification or support from the servers, and they don't require symmetric paths or new encryption schemes. Furthermore, cookies can leverage the primitives given by these protocols. For example, each cookie can have its own mcTLS context, and allow the network to modify it in order to provide network delivery guarantees.

BlindBox [30] takes a different approach: it performs deep-packet inspection directly on the encrypted traffic using novel encryption schemes. It is mostly tailored for IDS and firewalls (instead of expressing user preferences), and the associated complexity is significant—it introduces a new protocol, new encryption schemes, and a heavyweight process to setup the necessary encryption context for a new flow (up to 90 seconds).

Network capabilities [14] was a proposal to prevent DDoS attacks. Despite the different context, they are perhaps the closest mechanism to network cookies. Capabilities are tokens granted by a server to a client, to verify a connection between the two and signal the network to forward it through a protected path. In contrast, cookies are granted by the network and are not bounded to a specific connection—both AnyLink and Boost work for any traffic without requiring support from the servers. Furthermore, cookie descriptors decouple discovery, acquisition and authentication from actual traffic, leading to a much lower overhead mechanism.

There is a vast literature covering net neutrality, mostly from an economics and policy perspective. During recent FCC hearings, both sides of the debate talked favorably about user-driven prioritization [31, 1]. Net Neutrality advocates welcomed the approach, but raised concerns on whether the actual implementation of user-driven prioritization could limit user choice [11]. Network cookies directly address these concerns, as they provide a simple, policy-free mechanism, and allow regulators to decide and subsequently audit the policy simply by monitoring who gets access to cookie descriptors and how. Moreover, our user studies provide insights on how to better preserve the users' interests.

The design of network cookies has been motivated by "the tussle" [16], a position that accommodating the tussle between different Internet stakeholders is a crucial part to the evolution of the network's technical architecture; and of course HTTP cookies, a small piece of data that users attach to their traffic when they visit a website to customize their content.

8. CONCLUSION

We live in a time of enormous focus on how to improve the Internet, how to encourage ISPs to keep investing to improve it, and how to ensure users continue to have unfettered access to a broad range of services, content and applications. This focus has taken place mostly between large companies and government, with the interests of network operators pitted against content providers. Government's role has been to look out for us, the users. But largely, there has not been a way for us to take part, outside lobbying our political representatives. This paper tries to make two contributions to the debate. Network cookies provide a neutral, policy-free mechanism to express which applications receive special treatment from the network, regardless of popularity, complexity of task, or presence of middleboxes while providing the necessary means for authentication, accountability and user privacy. Using network cookies we advocate for a user-centric policy, where users can directly state their preferences and look after themselves. Our basic premise is that if users can express (and potentially pay for) how they want their network traffic to be treated, then it becomes safe - in fact desirable - to treat some traffic as more important than other.

Acknowledgements

The authors would like to thank our shepherd, Phillipa Gill and anonymous Sigcomm reviewers for their feedback. We would also like to thank Barbara Van Schewick and Ramesh Johari for valuable comments in early drafts of this paper, and Andreas Terzis, Ankur Jain, Roshan Baliga, and all other participants for their help with our Boost prototype deployment at Google. This work is supported by the Open Networking Research Center, the Platforms Lab at Stanford University, AT&T and Intel. The opinions expressed in this paper are those of the authors only.

9. REFERENCES

[1] AT&T comments to the FCC. http://apps.fcc.gov/ecfs/document/view?id=7521679206.
[2] Data Plane Development Kit. http://www.dpdk.org.
[3] Facebook Zero Wikipedia Entry. https://en.wikipedia.org/wiki/FacebookZero.
[4] Netflix apologizes for undermining Net Neutrality. http://www.fastcompany.com/3045150/fast-feed/netflix-apologizes-for-undermining-net-neutrality-in-australia.
[5] Qualcomm StreamBoost for Home Routers. https://www.qualcomm.com/news/releases/2013/01/04/qualcomm-introduces-streamboost-technology-optimize-performance-and.
[6] Sound Exchange Non-Profit Organization.
[7] Substrate Protocol for User Datagrams (SPUD) Prototype. https://tools.ietf.org/html/draft-hildebrand-spud-prototype-03.

[8] T-Mobile Music Freedom. http://www.t-mobile.com/offer/free-music-streaming.html.

[9] TCP Extended Data Offset Option (IETF Draft). https://tools.ietf.org/html/draft-ietf-tcpm-tcp-edo-03.

[10] Transport Layer Security (TLS) Extensions. http://www.iana.org/assignments/tls-extensiontype-values/tls-extensiontype-values.xhtml.

[11] Washington Post Article on User-Driven Prioritization. https://www.washingtonpost.com/news/the-switch/wp/2014/09/15/atts-fascinating-third-way-proposal-on-net-neutrality/.

[12] What is wrong with zero-rating and how to fix it. https://medium.com/@gyiakoumis/what-is-wrong-with-zero-rating-and-how-to-fix-it-7eb229e9e610#.5xzq2jktg.

[13] Why Music Freedom May Hurt Net Neutrality. http://venturebeat.com/2014/08/30/why-t-mobiles-music-freedom-is-hurting-net-neutrality/.

[14] T. Anderson, T. Roscoe, and D. Wetherall. Preventing internet denial-of-service with capabilities. *ACM SIGCOMM Computer Communication Review*, 34(1):39–44, 2004.

[15] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, et al. P4: Programming protocol-independent packet processors. *ACM SIGCOMM Computer Communication Review*, 44(3):87–95, 2014.

[16] D. D. Clark, J. Wroclawski, K. R. Sollins, and R. Braden. Tussle in cyberspace: defining tomorrow's internet. In *ACM SIGCOMM Computer Communication Review*, volume 32, pages 347–356. ACM, 2002.

[17] L. Deri, M. Martinelli, T. Bujlow, and A. Cardigliano. ndpi: Open-source high-speed deep packet inspection. In *Wireless Communications and Mobile Computing Conference (IWCMC), 2014 International*, pages 617–622. IEEE, 2014.

[18] P. Emmerich, F. Wohlfart, D. Raumer, and G. Carle. Moongen: A scriptable high-speed packet generator. *arXiv preprint arXiv:1410.3322*, 2014.

[19] S. K. Fayazbakhsh, V. Sekar, M. Yu, and J. C. Mogul. Flowtags: Enforcing network-wide policies in the presence of dynamic middlebox actions. In *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*, pages 19–24. ACM, 2013.

[20] FCC. Number Transfer Process. https://www.fcc.gov/consumers/guides/keeping-your-telephone-number-when-changing-service-providers.

[21] A. D. Ferguson, A. Guha, C. Liang, R. Fonseca, and S. Krishnamurthi. Participatory networking: An api for application control of sdns. In *ACM SIGCOMM Computer Communication Review*, volume 43, pages 327–338. ACM, 2013.

[22] http://consumerist.com/2012/04/16/netflix-ceo-rips-comcast-on-net-neutrality. Netflix CEO, Comcast & Net Neutrality.

[23] http://www.wired.com/2015/05/backlash-facebooks-free-internet-service-grows/. Backlash against Facebook Free Internet Service grows.

[24] H. Kim, S. Sundaresan, M. Chetty, N. Feamster, and W. K. Edwards. Communicating with caps: Managing usage caps in home networks. In *ACM SIGCOMM Computer Communication Review*, volume 41, pages 470–471. ACM, 2011.

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

[26] D. Naylor, K. Schomp, M. Varvello, I. Leontiadis, J. Blackburn, D. R. López, K. Papagiannaki, P. Rodriguez Rodriguez, and P. Steenkiste. Multi-context tls (mctls): Enabling secure in-network functionality in tls. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, pages 199–212. ACM, 2015.

[27] V. A. Olteanu, F. Huici, and C. Raiciu. Lost in network address translation: Lessons from scaling the world's simplest middlebox. In *Proceedings of the 2015 ACM SIGCOMM Workshop on Hot Topics in Middleboxes and Network Function Virtualization*, HotMiddlebox '15, pages 19–24, New York, NY, USA, 2015. ACM.

[28] Rusty Hodge, SomaFM Founder. Unfairness in T-MobileâĂŹs unmetered music streaming. http://rainnews.com/rusty-hodge-unfairness-tmobile-unmetered-music-streaming/.

[29] J. Schulz-Zander, C. Mayer, B. Ciobotaru, S. Schmid, and A. Feldmann. Opensdwn: Programmatic control over home and enterprise wifi. In *Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research*, SOSR '15, pages 16:1–16:12, New York, NY, USA, 2015. ACM.

[30] J. Sherry, C. Lan, R. A. Popa, and S. Ratnasamy. Blindbox: Deep packet inspection over encrypted traffic. *SIGCOMM Comput. Commun. Rev.*, 45(5):213–226, Aug. 2015.

[31] B. Van Schewick. Network neutrality and quality of service: What a non-discrimination rule should look like. 2014.

[32] Y. Yiakoumis, S. Katti, T.-Y. Huang, N. McKeown, K.-K. Yap, and R. Johari. Putting home users in charge of their network. In *Proceedings of the 2012 ACM Conference on Ubiquitous Computing*.