EXPRESSING USER PREFERENCES WITH NETWORK COOKIES

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF ELECTRICAL ENGINEERING
AND THE COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Yiannis Yiakoumis
April 2016

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Nick McKeown, Primary Adviser

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Ramesh Johari

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Sachin Katti

Approved for the Stanford University Committee on Graduate Studies.

Patricia J. Gumport, Vice Provost for Graduate Education

Abstract

Personalization, the practice of dynamically tailoring functionality for the needs and wants of each user, is a natural step in the evolution of many computing systems (such as PCs, search engines, and recommendation systems). The merits of personalization are obvious: Users get services more quickly and accurately, context and information gets more relevant, and the interaction between users and the personalized system becomes more amicable. This thesis explores personalization as a natural step of the network's evolution, along with its associated benefits and challenges.

At first, personalization might seem irrelevant for networks—after all, the network has a single, unambiguous and objective task to complete: Carry packets from one side to the other as quickly as possible. But in practice, networks are more complicated, continuously taking decisions about which traffic gets priority, how different applications are being charged, and to which WiFi network a user should connect and with what password. Making these decisions while ignoring users often produces suboptimal results. For example, the net neutrality debate highlights the dangers in dampening user choice and innovation when ISPs and Content Providers decide which applications are prioritized or zero rated. Similarly, our day-to-day interaction with several user-agnostic networks results in a fragmented user experience: We use different authentication methods and credentials for each network; reachability to desired resources is subject to the network to which we are connected; and we have very little, if any, control over policies such as firewalls and QoS settings.

This thesis presents a generic network architecture for personalizing network functionality. Network operators structure and expose functionality as high-level services (e.g., a fast lane or a zero-rated lane), and then let users tailor these services by expressing their own preferences. A critical piece for personalized network services is how do users communicate their preferences to the network. *Network cookies*, a generic mapping abstraction, provide this user$\rightarrow$network interface in a way that is simple yet expressive, respects the tussle between different stakeholders (e.g., users, ISPs, content providers and policymakers) in terms of security, privacy, revocability and authentication, and can be practically deployed in existing networks.

Leveraging network cookies and user preferences, I describe a user-focused alternative for the net neutrality debate: Enable fast or zero-rated lanes and allow users to decide which traffic goes over them. Through user studies for zero rating and fast lanes, I demonstrate that user preferences are heavy tailed, and users are willing to express their preferences if there is a simple way to do so. Network cookies allow users to express high level preferences (e.g., prioritize a website or a mobile application) with high accuracy in the presence of encryption and middleboxes. I validate the approach through the design and prototype implementation of Boost, a user-driven fast lane deployed in 160 home networks.

The last part of the thesis extends the concept of personalization by enabling a fully personalized network experience: Users define their network properties once (i.e., their WiFi SSID and password, devices, and policies such as QoS and firewalls) and their network follows them wherever they go—at workplace, public hotspots, or a friend's home. Personal WiFi networks provide users with a simplified and consistent experience regardless of how they are connected to the network. I describe the design and prototype implementation of BeHop, a personalized WiFi infrastructure deployed in a student dorm at Stanford University.

Acknowledgments

A PhD is a long journey, and mine was particularly long. I feel deep gratitude to everyone that accompanied me through it and helped to form my perspective over the last years.

Nick McKeown is the main reason I pursued a PhD. His stance on how to do research, pick interesting research problems and focus on them, have impact, work with colleagues and students, balance work and life, and even how to end the death penalty, looked interesting during my first two years at Stanford. Working with him towards a PhD seemed a good opportunity to understand how he achieves his goals and I am deeply honored to be his student. I hope I can apply some of the learnings through my post-PhD life.

I am thankful to Sachin Katti (my co-advisor) and Guru Parulkar for their invaluable mentorship and support throughout my studies. I would also like to thank Ramesh Johari and Barbara van Schewick whose insights shaped my perspective on net neutrality, and George Papadopoulos (my undergraduate advisor) for giving me several opportunities to grow and encouraging me to pursue graduate studies abroad.

Kok Kiong Yap, Te-Yuan Huang, and Manu Bansal have been amazing collaborators in numerous projects. I will miss our long conversations, endless technical arguments, and hectic nights against paper deadlines and demos.

Being part of the McKeown Group was a unique privilege. Many thanks to Brandon, Nikhil, Dave, Adam, Glen, Peyman, James, Neda, Jad, Lavanya, Guido, Martin, Rob, Srini, Masa, Ali (and others I forgot) for always being supportive and willing to provide constructive feedback to vague (and often silly) ideas. I particularly want to thank Lisa Yan for reviewing early drafts of this thesis.

Thanks to Andreas Terzis, Ankur Jain, and Umesh Krishnaswamy who mentored me during internships at Google and Juniper, Johan van Reijendam and Charlie Orgish for being the best IT administrators a network PhD student could ever wish for, and Chris, Betul and Catalina for taking care of everything administrative. I am also thankful to Stanford's Studio 5 residents that trusted their WiFi connectivity to our BeHop experiments. Thanks to NSF, ONRC, the Clean Slate Program and the Fulbright Foundation that funded my PhD studies.

My friends made my stay to the Bay area memorable, celebrating the good days and getting through the hard ones. Thanks to Dimitrios, Lampros, Gabriel, Panos, Yianni, Nikos, Manolis, Petros, Charis, Manuel, Gemma, Borja, Nicholas, Stephanie, Idoia, Mikel, Nadine, Carlos, Daniel, Christina, Argyris, Simla. And to my friends back in Greece who I greatly miss and who always (try to) keep me aware of the world outside the Silicon Valley bubble: Dimitris, Apostolos, Yorgos, Dimitris, Babis, Giannis, Markos, Vassilis, Yorgos. The Hellenic Association at Stanford gave me the opportunity and resources to work on initiatives related to Greece at times when I felt obliged to do so.

My parents, Dimitra and Iacovos, sacrificed way more than what I can imagine to give me and my brother all we needed to pursue our dreams, and insisted on us going after these dreams even when this meant we are far away from them as they grow older. Also thanks to the rest of my family (Zachos, Anna, Alexis, and Polia) for all their love and encouragement.

Last, but not least, I would like to thank Sotiria for all her love, patience and support since our undergraduate years. For all it takes to leave your family, friends, job, country, and come to a place that you don't even speak the language to be the wife of a student, I can't thank you enough, my love.

To 3, M. Merkouri Street,

&

To Sotiria

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation

Personalization is a natural step in the evolution of many computing systems. Personal computers (PCs) marked the move to an era where each individual had his own, dedicated machine, customized to fit his very own needs—installed applications, configuration, and wallpaper picture. Websites remember our preferences—our language, country of origin, and last activity when we visited this site. Movie and restaurant recommendations are adjusted based on our location, browsing history, places visited, ratings, and recommendations from friends. Even cars now come with driver profiles so that seat positioning, temperature, speeding, and braking behavior are all customized on a per-user basis.

The merits of personalization are obvious. Baseline functionality tailored for the needs and preferences per-individual enable users to get services more quickly and more accurately; system information becomes more relevant, and the interaction between users and the personalized system is more amicable.

One layer below this personalized front end lies the actual network that interconnects and enables these services. Ideally, personalization should be irrelevant here—the network has a single, unambiguous and objective task to complete: carry packets from one side to the other as fast as possible. But in practice, networks are more complicated. Which packets get priority over others, which traffic gets exempted from data caps, or which WiFi network should a user connect to and with what password?

There is no good answer of how to deal with these dilemmas. Should networks be ignorant of the desires of their users, is there a one-size-fits-all solution, or can a personalized network improve the experience for each individual user?

The goal of this thesis is to explore *personalization* as a natural step of the network's evolution. along with its associated benefits and challenges.

1.2 The Personalized Network

Networks provide various functions to enable users access the content they want: authentication to protect the network from unauthorized access; routing to send packets to their destination; QoS to provide better bandwidth or latency guarantees; and billing to charge network usage. In a *personalized network*, these functions are tailored to accommodate the needs of specific individuals, as opposed to general user groups or network administrators. Personalization is not limited to a specific network service (such as a high-bandwidth service) but rather suggests a generic, user-focused approach while building and exposing network functionality. Let's look at some specific use cases where personalization can be helpful:

Fast Lanes and Net Neutrality: Fast lanes provide higher bandwidth to certain traffic. At a technical level, we have all the pieces to enforce fast lanes today: scheduling algorithms, capable routers and switches, automation, and expertise to configure them in the actual network. What we lack is a critical piece of information: which traffic should go over the fast lane? Users can directly answer this question for themselves: one will choose Netflix, another Hulu, and a third ESPN when the local team is playing. A small group of PhD students would choose their conference paper upload website a few moments before an upcoming conference deadline. A personalized fast lane adjusts to the needs of individual users, maximizing utility for each one separately. Today, ignoring what users want, we are stacked with two mediocre options, as expressed in the current *net neutrality* debate: fast lanes where ISPs decide which traffic goes through them, and a neutral network that treats all traffic the same, as the best alternative to preserve user choice and innovation in the Internet.

Data-caps and Zero-Rating: Monthly data-caps and zero-rating services are network functions related to data usage and charging. While data-caps limit usage for baseline traffic (e.g., 1 GB/month for a cellular data-plan, or 100 GB/month for a home network), zero-rating services pick certain traffic to be excluded from these usage limits; some do not even require data-plans. They help ISPs cope with increasing traffic demand and experiment with new business models; and users access content which would otherwise be expensive or inaccessible. The question is similar with the previous fast lane example: which traffic should be zero-rated? There are several zero-rating services deployed today. Facebook Zero is a partnership between Facebook and mobile operators in developing countries, where users can access Facebook with no need of a data plan; Wikipedia follows the same model. T-Mobile's Music Freedom and BingeOn programs whitelist and zero-rate a small number of music and video streaming applications respectively. Existing zero-rating services try to guess what users want, offering one-size-fits-all solutions. An online survey with 1000 smartphone owners (§4) showed that these don't work for everyone, revealing several different usage profiles: people who stream video or music on their commute; social network or news addicts; avid runners who want to map their runs; and people who only care about maps and GPS navigation while on the go. A personalized network would allow each user to adjust zero-rating for his own needs.

Guest WiFi and Authentication: Our sources of WiFi connectivity continuously increase: our home WiFi network, corporate WiFi, public hotspots, restaurant and coffeeshops, supermarket and shopping stores. Unfortunately, complexity increases with the number of sources, as we have to deal with the diversity of all these networks: register or ask for SSID, password, and username; connect to unencrypted networks, setup VPN and use cloud-enabled services to allow our devices to talk across networks. All these networks are unaware of their users, statically configured by their own administrator. What would a personalized WiFi connectivity look like? We would always connect through our own SSID and password; our firewall and parental control policies would always apply; and our devices (at home, work, and mobile) would be reachable all the time. Our personal network would follow us wherever we go.

Non Shortest-path Routing: The route that packets follow is invisible to users—it results from a combination of shortest-path routing, traffic engineering, and peering agreements between ISPs. However, control over packet routing can sometimes be useful for users. This is verified by existing overlay services. Virtual Private Network (VPN) and proxy overlays enable users to surf the web anonymously, or access a website as if they were in a specific country or location (e.g., a university campus). Similarly, developers would benefit from routing flexibility if they could test their applications by routing traffic through the paths their customers use (e.g., to experience different data plans, pass through the core mobile network of an ISP, or even go through a different ISP to check how their traffic engineering policies might affect the application). These choices are not in conflict with ISP policies—they can complement each other. A personalized network would enable users to adjust routing for their needs, when shortest path is not enough.

Internet Plans: Internet plans are static and coarse. Users sign up for a plan that they must adhere to for at least a few years. However, their needs are much more dynamic and unique. Some users overprovision their plans to address this mismatch. A family may purchase a year-long, high-speed plan for their vacation lake house, yet only visits occasionally and stream a few movies per year. Others are more conservative; they underprovision and lack desired network resources when needed. Urban users surrounded by WiFi might sign up for a limited LTE plan, but they regret their choice when they want to use navigation or stream music during a road-trip. A personalized network would enable users to customize their Internet plans dynamically to fit their very specific needs.

Several other use cases will evolve, such as personalized firewalls and services that let users choose trade-offs between quality, bandwidth and/or battery usage. I will revisit some of these examples as motivating use cases throughout this thesis, and I will describe them in more detail in chapters §4 and §5.

1.3 Challenges to personalization

Personalizing network functions comes with certain challenges. What is the right interface to expose to users? How can we translate high-level user preferences to low-level network semantics? What are the incentives and concerns of different stakeholders (users, ISPs, content providers, policymakers) and how to balance across them? How can we scale network provisioning to account for each user's unique preferences? Some of these challenges will be solved once in a generic way, and others will have to be revisited for each network service separately. I discuss some of these challenges below:

Users Vs. Network Comprehension: What is simple for users is complex for networks; and what is simple for networks is meaningless for users. Users understand high-level semantics, like their favorite website and mobile application, or that they are watching a movie on their living-room TV. In contrast, the network sees a collection of packets and flows, often encrypted with HTTPS, aggregated by network address translation boxes (NAT), and destined to/originating from IP addresses of general-purpose servers in Content Distribution Networks (CDN) or datacenters. Bridging this gap between users and networks is essential to enable meaningful personalization.

Network owners have their policies: Owning and operating a network comes with responsibilities and cost. Personalization should coexist with the ability of network administrators to enforce their own policies. For example, while users decide which traffic goes through their fast lane, the ISP should be able to limit or charge usage appropriately. While users are able to connect to any WiFi AP using their own SSID and password, the owner of a home WiFi network should be able to control who can connect to the Internet through his own access point.

Heavy-tailed user preferences: Existing network services limit user choice to a few, very popular applications (e.g., T-Mobile's Music Freedom program). But users want different things, and we know that their desires are heavy-tailed. Letting users personalize their networks should enable them to realize all their preferences, regardless of how popular or appropriate they are. To accommodate this heavy-tail, personalization should minimize the cost for adding a new preference.

Accommodating *the Tussle* [18]: Stakeholders (users, networks, content providers, applications, and policymakers) have different interests and trust relationships. For example, a user wants to prioritize an application, but he wants to protect his privacy and not reveal to the network which application this is. A content provider wants to be zero-rated, but keep some of its traffic unidentified for A/B testing. And an operator doesn't trust DiffServ markings from other networks, resetting DSCP bits across network boundaries. Conflicts and tensions will differ with type of network services, stakeholders involved, regulatory frameworks. Network personalization should accommodate this tussle by providing the necessary primitives and allowing different outcomes in different places.

Scaling to thousands of users: Personalizing network functions comes with scalability concerns. We now need to provision the network with tens of thousands of preference-sets (defined by users) instead of a single one (defined by the network administrator).[1] Scalability depends on the network

[1] We focus on access/edge networks and assume moderate aggregation levels.

feature to be personalized, capabilities of equipment (e.g., number of hardware queues in a router), and will have an impact on what services and guarantees a network can offer to users.

Existing mechanisms do not work: The above-mentioned challenges are manifested by limitations of existing mechanisms which might initially seem good candidates to enable network personalization, such as Deep Packet Inspection (DPI) and DiffServ. DPI requires heavy configuration both to the application and the network to identify a simple service, such as a video flow from a specific content provider. It becomes impossible to detect more complex structures, like a website or a mobile application with contents from multiple sources (javascript libraries, social sharing plugins, embedded videos). This *high-transaction cost* prevents DPI from covering the heavy tail of user preferences and dynamic user context. DiffServ—a packet marking mechanism—doesn't provide any means of authentication, and assumes a small number of predefined services. As a result, it is used only within a single administrative domain, with markings ignored or reset across network boundaries. I discuss limitations of existing mechanisms in detail in §3.

1.4 Our Solution: Expressing User Preferences with Network Cookies

This thesis has two main goals; first, to demonstrate that personalized network services are practical, beneficial, and applicable in different contexts; and second, to provide a technical foundation for implementing and deploying such services.

I demonstrate the value and practical benefits of personalizing networks through three real-life examples. Two of them are relevant to net neutrality. A user-driven fast lane called Boost, deployed in 300 homes; and an online survey of 1000 smartphone owners with regards to zero-rating services. The third one, a personalized testbed that improves WiFi connectivity in a university dormitory, shows the applicability of the proposed approach in different contexts.

These examples suggest a generic approach for personalized networks: structure and expose network functionality as high-level services (e.g., a fast lane, or a zero-rated lane), and then let users tailor these services by *expressing their own user preferences* (e.g., "I want traffic from Netflix to be fast, or my favorite podcast to be zero-rated").

To communicate user preferences to the network, I developed a user↔network interface called *network cookies*[2]. A network cookie—similar to HTTP cookies—is a small piece of data that users append to their traffic to use a specific network service. To address the challenges described earlier, cookies follow certain design choices. They provide a generic mapping abstraction which can map any traffic to any service (or state in general), e.g., boost or zero-rate a website or a mobile application. Cookies place most of the intelligence to translate high-level user preferences to low-level network semantics closer to the user. This enables users to directly express their preferences (serving their

[2]I will be using cookies and network cookies interchangeably.

heavy tail), but also enhances preferences with rich and dynamic user context (e.g., unencrypted content, user location, activity monitoring). They provide the necessary security and accountability primitives through cryptography-based authentication and revocation; they can be incrementally deployed in the presence of HTTPS, NAT, and CDNs. In essence, network cookies decouple what users want from how their desires are implemented in the network.

1.5 Organization

Chapter 2 describes a generic architecture for personalized networks and a simple example (a personalized fast lane) which will act as a reference for following chapters. Chapter 3 introduces network cookies, a mechanism used to express user preferences to the network. I describe the concept of network cookies, discuss implementation and deployment details, and compare it with existing mechanisms. Chapter 4 studies how user preferences and network cookies can be applied to the net neutrality debate. I discuss a prototype deployment of a personalized fast lane called Boost in 300 homes, an online survey of user preferences with regards to zero-rating, and discuss the implications of user preferences and cookies for network regulation. In Chapter 5 I show how user preferences can be applied to personalize and improve WiFi connectivity. Chapter 6 concludes this thesis.

Chapter 2

Architecture

This chapter proposes a generic architecture to structure a personalized network service. Coming up with a system architecture involves two main tasks: (i) identifying the individual components that provide the desired functionality, and (ii) dividing the labor between them.

The first task is simple, as there are two main entities: users who declare their preferences, and the network that configures the switches and routers to provide the desired network service.

The second task is more complex—there are many ways to divide functionality between users and the network. A critical design choice for a personalized network service is the interface between the users and the network; i.e., how users communicate their preferences to the network. This choice can shape what type of functionality the system can offer, how inclusive it can be with user preferences, and how well it can meet the needs and incentives of different stakeholders.

In this chapter, I look at the individual components and explore the design space for a personalized network architecture (Figure 2.1). I also outline the choice of this thesis, which follows a simple principle:

*Users define **which** traffic gets **what** type of service, and **when** this happens. The network decides **how** and **where** service provisioning is implemented in the network.*

2.1 Users and User agents

Users are the individuals who use the network.[1] They interact at a high level with the overall system and use their preferences to tailor the personalized service for their own needs. "I want Netflix to be fast," or "I want my work-related video calls to get the best available service." Besides preferences, users have unique knowledge of their current context beyond what any network can deduce. For

[1]I shall clarify the distinction between users and network administrators. Unlike administrators, users lack knowledge, access rights, and interest to directly access and configure the networking equipment.

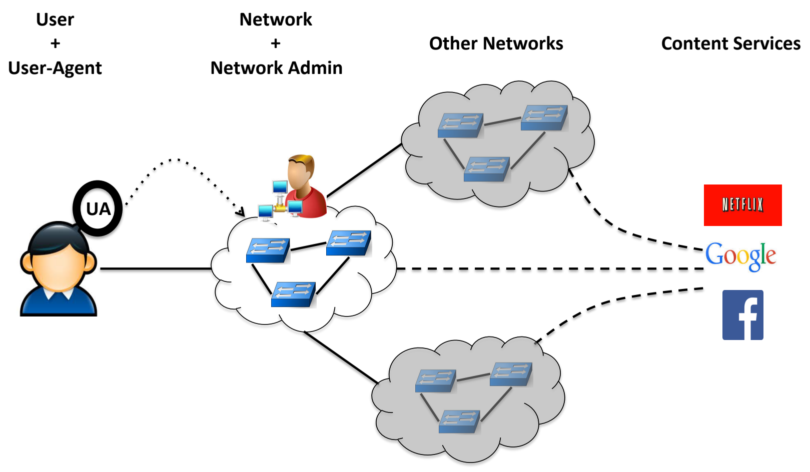


Figure 2.1: Basic architecture for personalized networks. Users express their preferences to user-agents which then communicate them to the network through a well-known interface.

example, users get frustrated when their video stream is rebuffering. Similarly, two video streams from the same content provider may have different priorities, depending on which plays on the living room TV. Often, user desires might seem unreasonable as long-term policies but can be logical given the current context. Consider a file upload: while uploads are typically considered background traffic, a student uploading a large submission for an impending deadline will consider this upload the most important task in his network.

Giving users control might sound counter-intuitive at first. Many argue that typical users have neither the expertise nor the interest to do so, and instead the network should be smart enough to do the right thing. I claim that this is not true. Only users know their preferences; neither the network nor content providers do.

Users express their preferences with the help of user agents. User agents essentially serve three roles: i) interact with the network to discover available network services, ii) provide an interface for users to express their preferences and extract relevant user context (e.g., traffic associated with a website), and iii) communicate these preferences to the network.

User agents can directly forward a preference to the network (e.g., “make Netflix faster”). Alternatively, they can leverage user context and application-specific state and enable richer preferences (e.g., ask for extra bandwidth when the video client buffer runs low, or prioritize traffic from the active tab of a browser). User agents will vary a lot and rapidly evolve over time. They can be deployed in different places (e.g., embedded in an application or be part of the Operating System), enable dynamic or static preferences, and expose diverse network functionality. I will discuss different types of user agents in Chapters §4 and §5.

2.2 The Network

The network encompasses all parties involved to provide network-related functionality: routers, switches, and middleboxes, as well as network administrators who set the policies and configure the equipment. To enable personalization, network services should allow users to tailor this functionality for their own purposes. One example is a virtual Wi-Fi network where each user can define and connect through his own SSID and password. Similarly, a fast lane where users decide which traffic gets prioritized.

Letting users decide does not imply that administrators give up control. Network Administrators still own, manage, and operate the network, define the network services and their characteristics, and ensure that the infrastructure can deliver on them. They can still maintain and enforce their own policies (e.g., how is a fast lane provisioned, what is the rate limit for zero-rating traffic, or who can access a home AP). It is up to the administrator to decide what control to provide to users. One might choose to expose an abstract service to users (e.g., a fast lane) and hide the details of how this is implemented internally, while another can can expose the internal structure of the network and let users pick the exact path their packets would follow (similar to source routing protocols).

2.3 User-Network Interface

To personalize network functions, we need a mechanism to express user preferences to the network. This mechanism reflects the division of labor between users and the network, and it can shape the type of functionality the system can offer, how inclusive it can be with user preferences, and how well it can meet the needs and incentives of different stakeholders.

To serve a plethora of user preferences, applications, and network services, the user-network interface has to be generic and stable. Getting it right is therefore very important.

First to consider is at what level should users and networks communicate. User preferences are high-level (e.g., "I want Netflix to be fast") while networks operate at a lower level—they just see a collection of packets and flows with very little context beyond that. One approach is to implement everything in the network—store user preferences, detect Netflix related traffic, and enforce the desired service. Another approach is to keep intelligence closer to the user and provide a low-level, application agnostic interface between users and the network. The former minimizes endpoint dependencies but is heavyweight for the network. The latter requires endpoint support, but leverages user context and enables preferences impossible to serve solely from the network (e.g., "prioritize traffic from the active tab of my browser"). As we will see in the next chapter, this decision has implications on whether network services can support the heavy tail of user preferences, or how well they can respect users' privacy.

A second consideration is how much control to give users. To make a video faster, users can just tell the network which traffic to prioritize, or alternatively, choose their own optimal path on a hop-by-hop manner. Complete user control is good for personalization, but introduces extra complexity for endpoints, requires ISPs to reveal their internal structure, and can be insecure. Exposing network functionality at a higher level (e.g., a fast lane) enables users to choose what they want without worrying about implementation details.

Other properties can also be important. Authentication primitives might be required to charge for a network service (e.g., prevent unauthorized applications from using a fast lane without user's consent). Similarly, a revocation mechanism (i.e., the capability to stop using or providing a service anytime) can provide trust and accelerate adoption of network services in certain environments. Low overhead and co-existence with well-established network trends (like HTTPS, CDNs, NAT) can be desired for a practical deployment.

Surprisingly, existing methods (such as DiffServ and DPI) lack critical properties to facilitate personalized networks (§3.1). A core contribution of this thesis is the design and implementation of a novel abstraction—*network cookies*—to provide the interface between users and networks. Network cookies follow a simple principle to divide the labor between users and the network:

Users define ***which*** *traffic gets* ***what*** *type of service, and* ***when*** *this happens. The network decides* ***how*** *and* ***where*** *service provisioning is implemented in the network.*

Cookies decouple what users want, from how this is implemented in the network. They place intelligence closer to users to leverage user context, enable diverse user preferences, and remove unnecessary overhead from the network. They expose network functionality as high-level services, and therefore simplify how users consume it and ISPs implement it. They provide a built-in authentication mechanism, revocation primitives, and can be incrementally deployed with existing infrastructure. While cookies are not the only plausible choice for personalized networks, I believe they capture the needs, capabilities and trust relationships of different stakeholders today. Chapter 3 describes network cookies in detail, along with the limitations of existing methods.

2.4 Motivating example: Boost, a user-driven fast lane

Boost is a use-case that follows the proposed architecture to implement a personalized fast lane. I will briefly describe it here to ground the discussion and use it as a reference for the next chapter. More details about implementation and deployment follow in Chapter §4.

Boost enables users to prioritize any website they like. They express their preferences to the Boost user agent on a simple way: every time they want to prioritize a website, they click a button on the right side of their browser. The user agent expresses this preference to the network which then enforces it and sends related traffic through the fast lane.

The Boost agent is embedded at the web browser and provides a basic user interface. By default, users boost a website ad-hoc, but they can also save their preferences so their favorites websites are automatically boosted. Besides the basic UI, the agent has visibility to raw, non-encrypted content as well as related browser state. It can easily associate flows with a specific website, understand which tab is active at any point in time, or detect a video client that runs low in buffer and could benefit from extra bandwidth.

Once the user agent knows the website to be "boosted," it can follow different strategies to communicate user preferences to the network. It can tell the network the website and let it figure out the rest; detect all network flows associated with this tab and share their description (5-tuple) with the network; mark the traffic using the DiffServ bits; or attach a network cookie to each outgoing flow.

Finally, the network enforces Boost at the head-end router of its network, just before the last mile connection. To provide a fast lane the network maintains two queues: a high-priority queue for traffic from/to the boosted tab; and a default priority for everything else. The high-priority queue is capped at 90% of available bandwidth to ensure that non-boosted traffic is not starved. How the network detects traffic to be boosted is a direct outcome of the user-network interface to communicate preferences. It might have to detect traffic related to Netflix, expose an API to learn the flows to be boosted, or just detect specially marked packets and send them through the appropriate queue.

Chapter 3

Network Cookies

Chapter 2 highlighted the user-network interface as a critical piece of a personalized architecture, as it will shape what type of functionality a system can offer, how inclusive it can be with user preferences, and how well it can meet the needs and incentives of different stakeholders.

This chapter starts with an overview of existing mechanisms that can provide this user-network interface and their limitations. Following this, I will describe *network cookies*, an abstraction that allows users to express their preferences to the network and to remote service providers.

3.1 Limitations of Existing Mechanisms

A user preference typically results in a subset of traffic getting special treatment from the network. In this section, I look at three existing mechanisms we could use to communicate these preferences to the network.

3.1.1 Deep Packet Inspection

Deep-packet inspection (DPI) is commonly used to provide special treatment for a subset of traffic. In DPI, the network auto-detects traffic to serve differently (e.g., traffic coming from a specific content provider) and then applies the service (e.g., higher bandwidth or zero-rating). DPI matches on traffic at line-rate—typically in a middle-box—by examining IP addresses, TCP ports, SSL's SNI field, or the packet content. Typically, a new set of rules is added for each application and web service.

To personalize a network function using DPI, users communicate their actual preference to the network, which then detects the related traffic and applies the service. In our Boost example (§2.4), users would explicitly ask the network to prioritize their website of interest.

DPI has what we call a *high transaction cost*—adding a new preference (i.e., a new set of rules) comes with high overhead. Originally, this overhead was acceptable, given that DPI did not require any changes in the endpoints; the network could detect traffic of interest simply by observing packets. This is no longer true. New services use HTTPS (which is increasingly common) or are hosted on a third-party CDN (e.g., video services hosted by Akamai). [1] TCP ports and IP addresses are meaningless (most traffic comes through ports 80 and 443, and the same IP address serves traffic from hundreds of services), and content is encrypted. As a result, DPI either fails to detect traffic of interest, or requires manual coordination between content providers and network operators to ensure that traffic is properly detected, further increasing the cost to add a new preference. This high transaction cost results in **low coverage**: Only a few, very popular applications will be included in DPI-based user preference services, as it would be costly for an operator to deal with the overhead of managing the preferences of each individual user.

Another disadvantage of DPI is that it cannot capture the complexity of typical objects associated with user preferences, such as websites or mobile applications. Take `cnn.com` as an example, the front page for news provider, CNN. Loading its front page generates 255 flows and 6741 packets from 71 different servers. Most of these servers are not related to CNN, but they host third-party javascript libraries, plugins for sharing (like Twitter, Facebook, and LinkedIn), or embedded video players. Nevertheless, they are an inherent part of CNN's website.

To evaluate the limitations of the approach, I checked how well DPI could boost three potential user preferences: i) cnn.com, ii) youtube.com, and iii) skai.gr (the website of a Greek TV station).[2] I used nDPI [19], a publicly available DPI system that can detect more than 220 popular applications, protocols, and websites. I asked nDPI to recognize and boost the websites, based on observed network traffic, and then verified whether it identified them correctly.

Navigating to the front page of each site generated 255/6741, 80/3750, and 83/1983 flows/packets, respectively. Of the packets, nDPI matched 9% of actual packets from `cnn.com` and 51% from `youtube.com` (Figure 3.1). It didn't match any packet from `skai.gr`, as there was no rule for it. Moreover, nDPI occasionally matched the wrong packets (false positives). When trying to boost `youtube.com`, it also matched 12% of packets from skai.gr, as it embedded YouTube's video player.

Similar to websites, DPI tools fail to detect other objects that can be associated with user preferences, like a mobile application or a video player. MusicFreedom users recently complained that they were charged for using music applications advertised as zero-rated[5]. Similarly, a video-streaming program was able to zero-rate traffic from a specific content provider but failed to detect advertisement videos coming from a CDN [8].

Perhaps the most problematic issue with DPI is that its functionality depends on the network knowing the actual, high-level preference of the user. In other words, users have to reveal to their ISP the service they are requesting special treatment for, potentially hurting their privacy. Clearly we would prefer a mechanism that allows users to request improved service for any traffic without

[1] As of 2015, 30% of total traffic in the U.S. was encrypted, and this is expected to double by the end of 2016 [11].
[2] As we will see in Chapter 4, these are real user preferences from a Boost deployment.

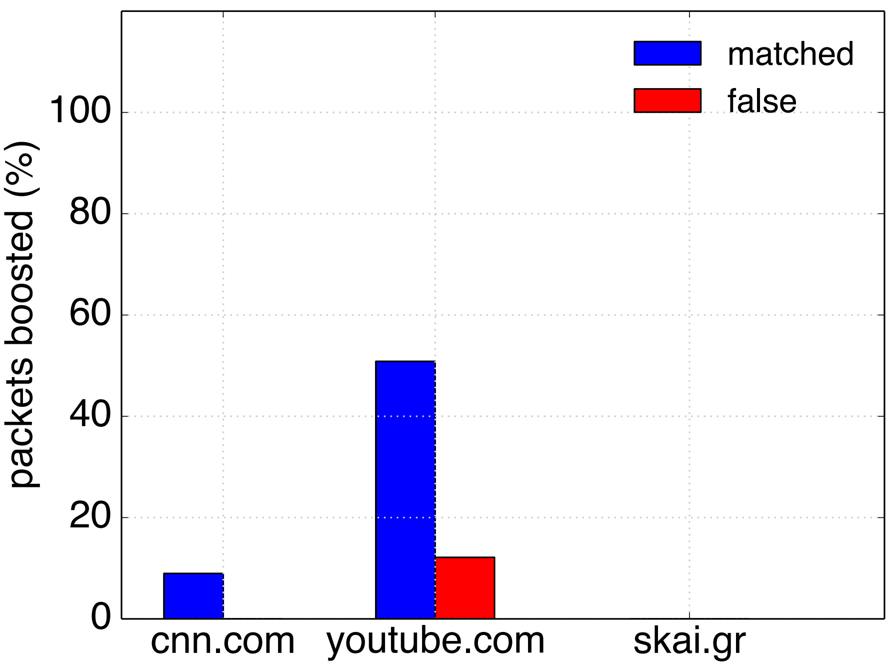


Figure 3.1: Matching accuracy for three websites using DPI. DPI cannot capture the complexity of a website, matching only a small percentage of related traffic. It fails to match any traffic from skai.gr, a less popular website.

explicitly revealing the service they are using. Furthermore, to be compatible with DPI techniques, content providers must structure and build their service in a way that the network can detect it, which is then hard to revoke. This is problematic, as it becomes hard to run an A/B experiment with only a subset of traffic getting special treatment, or to completely withdraw from a service.

In summary, despite their wide deployment, DPI tools struggle to meet the needs of modern applications and have severe limitations in enabling users to express their preferences. Their high transaction costs result in high overhead for network operators and content providers and low coverage, which cannot serve the long tail of user preferences. They cannot expose high-level user semantics (e.g., a website or a mobile application), as they cannot relate a flow with a higher-level object. Finally, they require actual knowledge of the user preference, which is not necessarily aligned with the privacy interests of users or content providers.

3.1.2 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 [47, 24]. In this

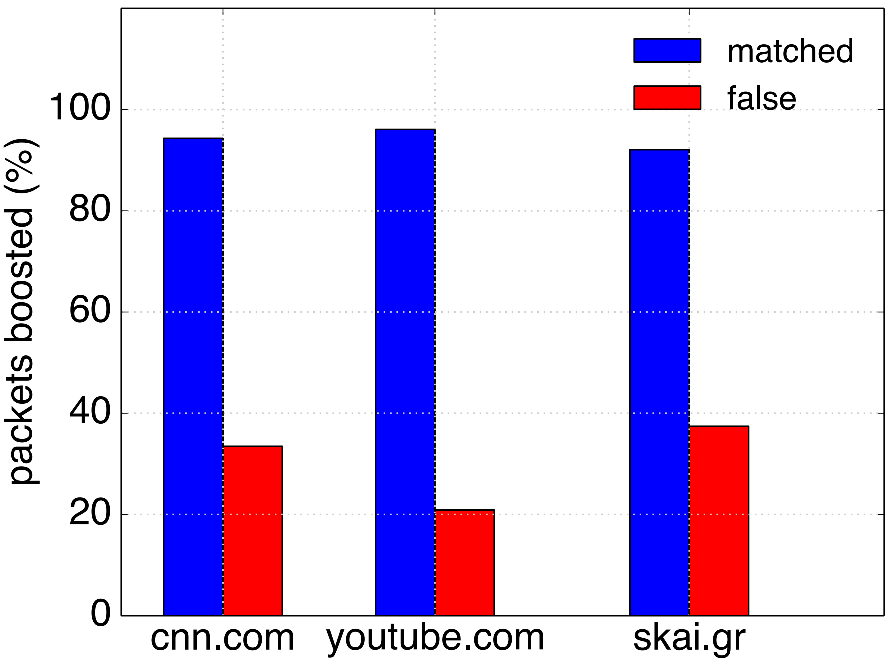


Figure 3.2: Matching accuracy for three websites using an out-of-band approach. OOB can match a website with high accuracy, and respects the long-tail of user preferences. It doesn't work well when flows change (e.g., by NAT or encapsulation). Using coarse granularity descriptions leads to false positive matches.

approach, the user agent tells the centralized control plane which flows to match to (via an out-of-band [OOB] channel) by describing the actual flows to get special treatment (e.g., using the 5-tuple). Subsequently, the control-plane programs the switches to match these flows.

OOB removes the burden from the network to detect traffic related to a user preference. User agents can identify the exact flows related to this preference and directly provide the necessary flow descriptions to the network, which can then act upon them. OOB therefore removes the high transaction cost associated with DPI, and—given the appropriate user agent—it can cover meaningful user abstractions (e.g., a website or a mobile application) with high accuracy (Figure 3.2).

In our Boost example, the user agent would detect all flows related with the boosted tab, and then inform the control plane through a REST API.

Unfortunately, there are two important limitations associated with this approach. First, signaling user preferences through the relatively slow control plane can be expensive. Recall that the front page of CNN has 255 flows; sending each of them through a centralized controller and then reprogramming multiple network switches appropriately is an expensive process. Often, the actual flow will be completed before provisioning can take place.

Second, giving a static flow description does not work when the flow changes—for example, as it traverses a NAT or is encapsulated. Therefore a user's description of a flow is most likely invalid within the network. In the Boost example, the flow will change at the NAT module of the home router, making the 5-tuple flow description invalid for the head-end router ISP. 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. Boosting skai.gr accidentally boosted 40% of traffic from cnn.com and youtube.com, as a big part of each traffic website's originated from the same servers (CDN, analytics, ads, social sharing plugins).

The limitations of this OOB approach highlight a more generic shortcoming of mechanisms that depend on the headerspace (i.e., the fields of a packet header) to identify traffic of interest. We've already seen how CDNs and wide use of HTTP(S) affect DPI tools. In Chapter 5, I will discuss how dependencies on MAC addresses result in privacy implications and have become a showstopper for research and industry efforts to add features in WiFi networks.

3.1.3 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). DiffServ is widely deployed in commercial routers and is used within ISP or enterprise networks to give preferential service.

DiffServ has some nice properties. It has zero transaction cost, and it can accurately match any flow or group of flows with high accuracy, without worrying about false positives. Moreover it does not depend on other fields in the header—all the information is contained within the DSCP bits, and, in principle, it can work even if the rest of the flow changes.

However, despite being supported by all modern routers and operating systems, DiffServ cannot be practically used today to express user preferences to the network. Network operators 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. Even when an application is allowed to mark the bits for its own traffic, the marking will apply only for the outgoing traffic, which typically is less critical compared to downlink traffic.

A common fallacy is to blame network operators or application developers for not respecting and/or not exposing the necessary APIs to mark the DSCP bits. I believe that this lack of support is simply the outcome of two deeper limitations of the DiffServ mechanism.

First, and most important, DiffServ lacks authentication primitives: Any application can set the DSCP bits and request service without the user's consent. Any developer could go and ask for low-latency, high-bandwidth service, even if it conflicts with user preferences, or—even worse—if it results in network charges for users. A workaround would be to trust the OS to provide the necessary authentication functionality, but this would create dependencies on operating systems and can be easily bypassed by users and devices.

The second limitation is that the limited set of DSCP bits supports only 64 classes (2^6), which leaves little room for customization. The same class could be used to mark production traffic in one company, and background traffic or traffic coming from the CEO in another. Expecting all networks to agree on a small set of predefined network services has proven to be unrealistic.

As a result, DiffServ is only suited as an internal marking mechanism, and it cannot be practically used today to express user preferences to the network

3.1.4 From Limitations to Requirements

The limitations of existing approaches have motivated my 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) to let users choose any application they like and to express a complex set of changing user preferences (e.g., a flow, a website, or a mobile application).

It should ***respect the tussle*** and trust relationships between different stakeholders (i.e., users, the network, content providers, and policy makers). It should preserve a user's privacy so that users do not have to explicitly declare their preferences to the ISP when asking for special treatment. It should prevent an unauthorized party from requesting special treatment without the user's consent, and should provide the required security to enable accounting and facilitate services that might involve charging. Most importantly, the mechanism should be policy-free and should not preclude a specific outcome on how the tussle is being accommodated. This flexibility is necessary to adjust within different contexts, regulatory frameworks, and relationships among various parties.

Finally, the mechanism should be ***practically deployable***. It should come with low overhead and work nicely in the presence of CDN, NAT, and HTTPS. It should be incrementally deployable without requiring forklift changes in network or content-provider infrastructure.

In the next section, I propose **network cookies**, a new mapping abstraction that meets these requirements.

3.2 Network Cookies and Cookie Descriptors

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. The network and the end host inspect the cookie, look up the cookie in a table, and decide which service to apply to the packet (and possibly to all other packets belonging to 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 I describe below. One way to attain 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 very slow and burdensome. Instead, the user requests a *cookie descriptor*, which is then used to locally generate multiple cookies. Periodically, the user requests a new cookie 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 Amazon Prime video. The user picks a *cookie descriptor* from the well-known server; the user may buy it or be entitled to a certain number per month, via coupons or according to 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,
  // A symetric, shared key used to sign a cookie.
  char * key,
  // Service data identifies the network service
  // the packet should receive. It is unformatted, and
  // can be just the name of the service to be used
  // (e.g. a fast-lane) or any other information (e.g.
  // a username or a device type).
  char * service_data,
  // An optional list of attributes
  // that characterize a cookie descriptor (when and how
  // to use it, whether it is valid or not). We discuss
  // some characteristic attributes in 3.4.
  char * attributes[]
};
```

Listing 3.1: A network cookie descriptor

From this descriptor, the local host generates cookies with the following fields (List. 3.2):

```
struct cookie {
  // 64-bit id copied from the descriptor.
  uint64_t cookie_id,
  // A universally unique identifier for this specific
  // cookie.
  uuid_t uuid,
  // The current time when the cookie was generated
  // from its descriptor, to limit the duration for
  // which a cookie is valid and prevent reuse.
  // It also reduces the amount of state kept by the
  // network (see \ref{subsec:network_phases}).
  uint64_t timestamp,
  // A Message Authentication Code (MAC) using the
  // descriptor's key (a shared secret between the
  // user and the networks) to prevent an
  // unauthorized person from generating a cookie.
  char * signature
};
```

Listing 3.2: A network cookie

3.3 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 3.3 shows the main architecture.

The first component to enable a cookie-based workflow is the *user agent*. The user agent has a GUI for users to express their preferences, and in the background, it interfaces with the network to discover and acquire cookie descriptors. The user agent also generates and adds cookies to packets. User agents can be integrated with an operating system, a browser extension, or be embedded in an application (e.g., a content provider's video player or a VOIP application). The second component is the *known server*, which is where users go to acquire cookie descriptors. Finally, *network switches* (or middle boxes) are the ones that inspect cookies and apply the right service.

Getting and using a cookie descriptor happens in three stages, which I describe below.

3.3.1 Cookie Descriptor Discovery and Acquisition

Users and their agents learn of network services through standard discovery protocols (DHCP, mDNS) or from information hardcoded in the application (e.g., Amazon Prime Video might know where to get special Amazon cookies). Once found, the cookie descriptor is downloaded over an (optionally authenticated) out-of-band mechanism (e.g., an HTTPS REST API). Different networks will choose different ways to allow their users to discover and acquire available descriptors. For example, in a home network, anyone who can talk to the AP might be eligible to get a cookie descriptor, while in a cellular network, users might need to log in through their ISP credentials to get cookies for a given service.

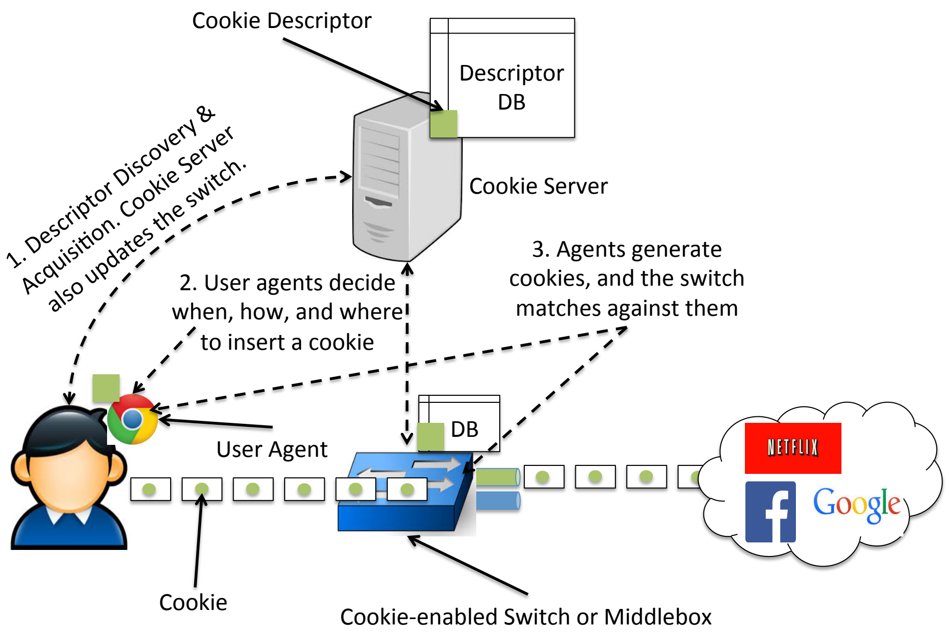

Cookie Descriptor
Descriptor DB
Cookie Server
1. Descriptor Discovery & Acquisition. Cookie Server also updates the switch.
2. User agents decide when, how, and where to insert a cookie
3. Agents generate cookies, and the switch matches against them
DB
User Agent
Cookie
Cookie-enabled Switch or Middlebox
NETFLIX
Google

Figure 3.3: Architecture and workflow for a cookie-enabled service.

3.3.2 Cookie Insertion

Once users get a cookie descriptor, they have to insert related cookies in their traffic to request a specific service. There are several ways to use a cookie.

The first consideration is *who* adds a cookie. Typically, cookies will be added by user agents. In some cases (as we will see later in this chapter), cookies can also be added by a remote server, which can act on a user's behalf.

The next consideration is *when* to use a cookie. Users insert a cookie when they want to use a specific service, which can be explicitly requested by the actual user, or assisted by an application (e.g., a video client can ask for extra bandwidth if its buffer runs low). Moreover, since descriptors can be tied to a given network, it makes little sense to use them when not connected to it. Cookie attributes (§3.4) are optional metadata associated with a cookie descriptor that describes how and when a cookie can be used.

A key aspect of using a cookie is *how* to "transport" it (i.e., *where* to add a cookie to our packets). There is no single answer to this—I suggest supporting multiple choices and picking the one that works better given the scenario. For HTTP traffic, we can add a cookie as a special HTTP header; for SSL traffic, we can add a cookie in the TLS handshake (as a custom extension in ClientHello messages) [14]; at the transport layer, we could use TCP long options [12], integrate cookies with a protocol like QUIC, or encapsulate traffic using a custom UDP-based header; and when IPv6 is

```
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 3.3: Pseudocode for cookie generation and matching.

granted widespread adoption, we can add cookies as IPv6 extension headers. In Chapter 5, I describe the use of cookies in WiFi Probe Requests to personalize WiFi's discovery process.

Choosing the right layer to transport a cookie depends on the applications and network services involved. Even though the basic workflow works regardless of the transport, in 3.6, I discuss some deployment considerations that could affect where to add a cookie.

3.3.3 Cookie Generation and Matching

The last step to use cookies is to generate and match against them. Generation is easy through the cookie descriptor, and the cookie is added to an outgoing packet.

When the network detects a cookie, it looks it up and, if needed, enforces the appropriate service. To verify that the cookie is valid, it checks that (i) the cookie ID is known, (ii) the MAC matches, (iii) the timestamp is within the "network coherency time," and (iv) it hasn'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, the network keeps a list of recently seen cookies (within the NCT). This is just a membership test, and we can use Bloom Filters to reduce state keeping. A reasonable NCT (e.g., 5 seconds) reduces state keeping to verify uniqueness. If a cookie is valid, the network applies the service directly (e.g., sends the packet through a high-priority queue). If it fails to match, it behaves as if the cookie were not there, offering default services. Sample code for generation and matching can be seen in Listing 3.3. I will discuss network deployment considerations in Section 3.6.

3.4 Cookie Attributes

Occasionally, users (and their agents) might need additional information on how and when to use 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 location, or applicable for traffic related to a specific network domain. Therefore, cookie descriptors carry unformatted, optional attributes, added by the network or end host, to provide more specific information about a cookie.

I expect some cookie attributes to become commonplace, such as the following:

- **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. The granularity field can also define whether the service associated with a cookie should also be applied to the reverse flow. For the Boost example, a cookie attached on an HTTP GET request would put all packets of this flow (and the reverse response flow) through the fast lane. A cookie attached in a WiFi Probe Request (§5) would result in a personalized Probe Response only for this request.
- **Shared:** When set, the cookie descriptor can be shared among multiple endpoints. In a home-network example, 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. Because of the cookie UUID field, endpoints can generate unique cookies even when they share the same descriptor.
- **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 setting up necessary state on the reverse path or verifying 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 the 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. There are many reasons why a cookie could be ignored—for example, a bug in the client or the network, a crash in the switch that caused temporary loss of state, or an erroneous use of an expired cookie by the client. Network delivery guarantees can be useful in certain scenarios: Imagine a zero-rating service, through which a user can stream a video with no charge by using a cookie. Doing so without a cookie would lead to a \$20 charge with standard charging rates. Instead, if the client fails to see an acknowledgment cookie from the network, it could notify the user that this video will be charged with nominal rates, giving the option to stop the stream before any charges occur. This option is better than trying to dispute an unexpected charge at the end of the month. Network delivery guarantees assume that the network can modify traffic between endpoints to add a cookie and therefore depends on whether the transport protocol supports traffic modification. This would be possible using HTTP and IPv6, but SSL/TLS prevents third parties from altering the communication between endpoints. New protocols (like mcTLS [37]) enhance SSL to allow middle boxes to change traffic between endpoints in a trusted way.

- **Cookie Transport:** This attribute contains a list of protocols over which this cookie can be carried (e.g., HTTP, TCP, IPv6, TLS).
- **Expiration:** The expiration attribute states until when a cookie descriptor is valid. Descriptors can be valid for a short period (a few hours or days) or much longer (a year or infinitely). While use of a service can be revoked at any time by the client or the network (by ceasing use or matching against a cookie), expiration dates can be used to inform users that a service is no longer valid with the current credentials, and also limit the risk of a descriptor leakage.

3.5 Putting Everything Together

Let's revisit our Boost example with cookies in mind. 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 then connects to the known server with the user's credentials and acquires a cookie descriptor, which is valid for one week and can be shared. The Boost extension embedded in the web browser goes to `http://192.168.1.1/getcookies`, discovers that a fast lane is available in the network, and asks for the user's permission to use it. Cookies are inserted as special HTTP headers for plain HTTP traffic or as TLS extensions for encrypted traffic. Upon user requests, the Boost agent uses the cookie descriptor to generate cookies, adds them to outgoing traffic, and the associated traffic goes through the fast lane.

Figure 3.4 demonstrates how cookies can boost the desired content with high accuracy and no false positives. The small percentage of undetected packets are non-HTTP(S) traffic, such as DNS queries.

3.6 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, I built a cookie-based zero-rating middlebox on top of Click and DPDK [33, 2] using an off-the-shelf server.[3] The 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. I 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.

[3]8-core Intel Xeon @ 2.60GHz, 128GB of RAM, 2 10Gb NIC.

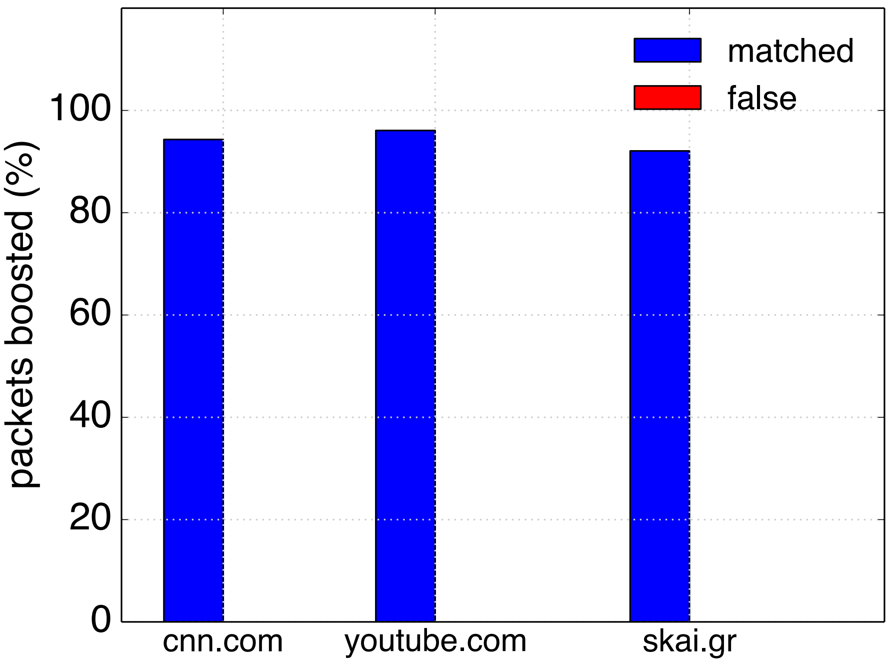


Figure 3.4: Matching accuracy for three websites using network cookies at HTTP and HTTPS flows. Cookies can match a website with high accuracy and no false positives, respecting the long tail of user preferences and work in the presence of NATs, CDNs, and encryption. The small percentage of undetected packets are non-HTTP(S) traffic, such as DNS queries.

For a given packet, the 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.

I evaluated performance against a 15-hour anonymized trace that included all wireless traffic from Stanford University's main campus, student residences, and visitor WiFi. It contained 11.3 million HTTP(S) flows originating from 73613 distinct IP addresses (median flow size is 50 packets, and 99th percentile for new flows per second is 442).[4] I connected the middlebox with a MoonGen packet generator [21], which sent flows with cookies and monitored how quickly the middlebox could forward packets. The setup is shown in Figure 3.5. Assuming 50-packet flows, 100K cookie descriptors, and a cookie for each flow, the 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 dropped below line rate for smaller packet or flow sizes (Figure 3.6).

[4]The trace was collected on 01/26/15 from 9a.m. to 11:59p.m..

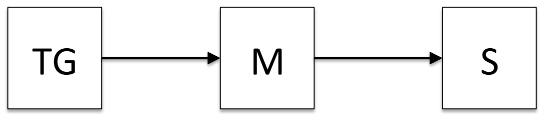


Figure 3.5: A simple testbed to evaluate cookies. A traffic generator sends flows from hypothetical clients, the middlebox inspects for and acts upon cookies, and the sink detects dropped packets to benchmark performance.

There are certain steps to further scale this system. First, we can use multiple cores instead of one and similarly 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 6 commodity servers [38]. The main challenge of scaling 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.

I will 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 use cases in which proxying overhead is not critical (e.g., zero-rating for cellular networks).

Cookie$\rightarrow$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) without requiring all switches to support cookies. In essence, cookies enable users to express their preference, while the network uses a different mechanism to realize it 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 (at 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

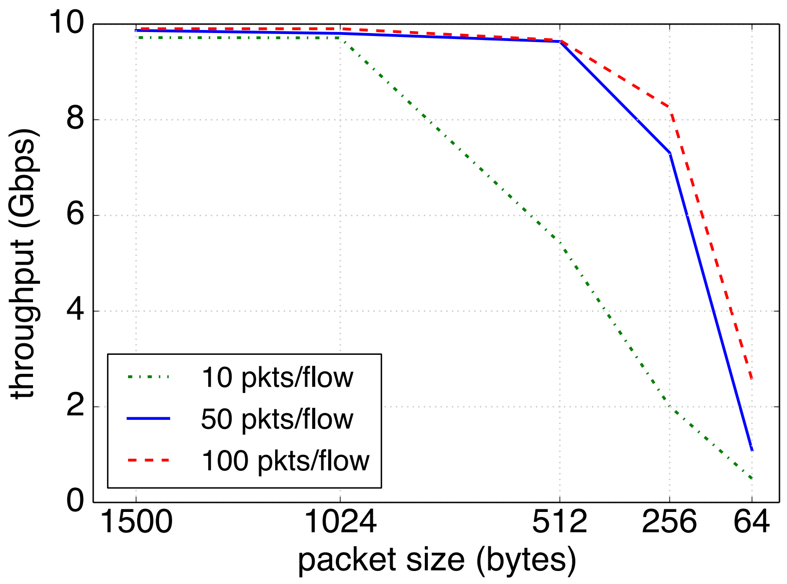


Figure 3.6: Matching performance for a Click-DPDK-based cookie middlebox. Our prototype can provide 10Gb/s line rate for 50 packet flows and 512-bytes packets using a single core and process all wireless traffic from a university campus.

current equipment does not support HMAC-style verification or direct state setup for reverse flows. However, configurable hardware [16] 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 compare 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 use cases. Furthermore, existing trends, like software-defined networks, programmable hardware, and network-function virtualization will further improve their applicability and performance.

3.7 From Requirements to 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 I describe below. Table 3.1 provides a summary of these properties and compares them 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 a general mechanism for expressing a wide variety of different user preferences and network services. 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 any preference users have (i.e., the long 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 by composing multiple cookies together. Finally, cookies can be cleanly *delegated* by the user to either the content provider or some third party that figures out how to best use cookies for this user. 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 among 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 cookie-acquisition phase, which means that only the authorized user 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. 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; the network can similarly stop matching against a cookie to stop offering a service. By separating mechanism from policy, cookies can enable different outcomes in tussles, depending on trust relationships and regulatory frameworks between the different stakeholders. This will become more clear in Chapter 4, in which I will discuss the implications of cookies in the net-neutrality debate.

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 and then use an internal mechanism to consume a service within the network (e.g., DiffServ or FlowTags [22]). In short, cookies enable users to express a preference, and the network uses a different mechanism to realize it. Acknowledgment cookies expand cookies functionality, to simplify service enforcement for *reverse flows*, especially for asymmetric 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 data plane use. They are generated, inserted, and matched locally in 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.

		Cookies	DPI	OOB	DiffServ
Simple & Expressive	arbitrary traffic $\leftrightarrow$ 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 3.1: Network cookies properties and comparison with alternative mechanisms to map traffic to a network service.

3.8 Limitations

Network cookies are not a panacea to expose and implement network services—they have certain limitations, which I discuss below.

To express user preferences, cookies provide a user-network communication primitive. This happens through a mapping abstraction driven by endpoints and associates traffic to state established through an out-of-band exchange of cookie descriptors. **This one way user$\rightarrow$network mapping abstraction can be limiting in certain use cases**. Cookies assume a model in which users explicitly express their preferences and coordinate with the network. When coordination is not applicable, cookies cannot help (e.g., a firewall that scans for peer-to-peer traffic without the user's consent). Often, other communication primitives outside of user-driven mapping might be required. A network-driven mapping abstraction can enable an ISP to help video clients choose the right streaming quality by sharing information regarding the bandwidth and data caps of a user's data plan. A generic, bidirectional communication primitive between endpoints and networks can be used to communicate arbitrary values (instead of predefined mappings through descriptors), such as the battery level of a device, the queue occupancy of a router, or the signal-to-noise (SNR) of a mobile phone. While not designed to enable these use cases, cookies can provide some workarounds. To provide a network$\rightarrow$user mapping abstraction, we can send cookies originating from the network, assuming that the endpoints allow the network to modify traffic. To communicate dynamic state to the network (e.g., battery level or SNR) using cookies, we can quantize the space to a small number of predefined values (e.g., low, medium, full battery) and keep a cookie descriptor for each of them.

Despite existing authentication and accounting mechanisms, **cookies can be used by unauthorized parties** in certain cases. The most obvious case is a cookie descriptor leaked to a third party. Verifying cookie uniqueness can also be challenging, especially in a short time window (i.e., sub-second) when cookies are distributed across different middleboxes. Moreover, as cookies are standalone entities, a man in the middle could potentially "steal" a cookie—drop a flow that carries a cookie and then use it for his own purposes. The probabilities of, risks of, and protections from such attacks should be taken into consideration when deploying a cookie-based service. First, reasonable expiration dates for cookie descriptors can reduce the risk for a leaked descriptor. Network-coherency time can limit the time window for a replay or stolen-cookie attack. As an extra measure, an ISP can further confine a cookie to a physical port and/or IP address of a given subscriber to ensure that it cannot be used by a third party elsewhere.

Lastly, **cookies require support both at the endpoints and the network**, a significant challenge towards deployment and adoption. Endpoint support is directly related to where the cookie is inserted. Depending on the scenario, endpoint support can involve just the application itself, a shared library, and/or the operating system. Further, if network delivery guarantees are required, the transport protocol should allow the network to modify traffic.

3.9 Related Work

In §3.1, I described existing alternatives to network cookies (DPI, OOB, and DiffServ), and how they fall short to meet the requirements for expressing user preferences to the network. This section refers to more recent work related to network cookies and expressing user preferences to the network.

FlowTags [22] facilitate enforcement of network services in the presence of middleboxes. To do so, network administrators and middlebox providers use TOS bits to appropriately tag and track flows, despite the flow header being modified. FlowTags focus on enterprise settings and assume a trusted environment and full control throughout the network path a packet follows. Unlike cookies, they are an internal mechanism for network management, not an interface between users and ISPs. They lack authentication primitives, support only up to 64 tags, and can be easily spoofed by others.

In OpenSDNW [41], authors use DPI to optimize wireless performance for specific applications based on user preferences. In [24], the authors suggested a participatory, OOB interface to bring user and application insights into the network. As discussed earlier, DPI and OOB methods suffer from false positives (DPI, OOB) or false negatives (DPI). By allowing users to add network cookies, as tokens to indicate preferred service, these services would be more accurate.

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 [37] extends TLS to allow endpoints to incorporate trusted middleboxes into secure sessions. It splits the payload into multiple contexts, and upon approval from client and server, an authenticated middlebox can be read and/or write access on a per-context basis. Additionally, mcTLS provides 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, they are much simpler than mcTLS. Unlike mcTLS, we can carry cookies along with multiple existing protocols (e.g. http[s], UDP, TCP) without modifications to protocols or the design of a new transport protocol itself. Cookies don't even require support from the server and can be incrementally deployed today. While mcTLS requires both directions of a flow to follow the same path, cookies can relax this assumption by bouncing back an acknowledgment cookie from the server. If mcTLS eventually gets deployed, it can be combined with cookies to explore richer communication primitives. For example, if we put a cookie in its own mcTLS context and allow middleboxes to modify it, we can use cookies for network$\rightarrow$user communication or to generate network acknowledgment cookies.

BlindBox [42] takes a different approach: It performs the deep-packet inspection directly on the encrypted traffic. BlindBox solves a different problem from expressing user preferences to the network by making desired traffic properties (e.g., Does a flow contain adult content?) detectable from encrypted traffic. BlindBox is designed for network tasks like firewalls and IDS; while in principle one could use it to express a user preference to the network, the associated complexity is not justified. For example, BlindBox introduces a new protocol, new encryption schemes, and a heavyweight process to set up the necessary encryption context for a new flow (up to 90 seconds).

Network capabilities [15] 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.

The design of network cookies has been motivated by "the tussle" [18], a position that accommodating the tussle between different Internet stakeholders (users, ISPs, content providers, and regulators) is a crucial part to the evolution of the network's technical architecture.

3.10 Analogy to HTTP cookies

Network cookies are named after HTTP cookies, from which they borrow many ideas. An HTTP cookie is a small piece of data that users attach when they visit a website to customize content for their needs. A network cookie is a small piece of data that users attach to their traffic to customize packet delivery based on their preferences. HTTP cookies provide a mapping primitive between a client and a server, while network cookies provide a mapping between a client and a trusted network. Similar to network cookie descriptor attributes, HTTP cookies use attributes such as domain, path, expiration date, and security to decide when a cookie should be inserted on a web request. It is my hope that network cookies one day become as widely used as HTTP cookies.

Chapter 4

User Preferences and Net Neutrality

In this chapter I discuss user preferences and network cookies in the context of net neutrality. More specifically, I will look into two types of services that relate to net neutrality (fast lanes and zero rating), examine the status quo through existing use cases, and discuss how network cookies and user preferences can move the debate forward. I finally demonstrate the value of network cookies and user preferences in the context of net neutrality through two user studies.

4.1 Background on Net Neutrality

Policy makers, ISPs and content providers are locked in a debate regarding which applications receive special treatment from the network, and whether special treatment is in the best interest of users. This discussion—often referred to as net neutrality, fast and slow lanes, or zero rating—goes beyond technology, and is a hot political topic; even making into John Oliver's weekly show and President Obama's speeches [6, 1].

Net neutrality is important because of its potential impact on the innovation model, economics, and use of the Internet. On the other side is the desire to improve the Internet, encourage ISPs to keep investing to improve it, and compete between each other for users through better services. On the other side is the concern that if ISPs and content providers can decide which applications get the best service, they could squeeze out new services, create an insurmountable barrier to entry for innovative new applications, and dictate user behavior.

Services related to net neutrality can be categorized based on three basic properties, which I briefly describe below.

1. What kind of special treatment does the network provide? Traditionally, net neutrality has been about *Quality of Service* (i.e., high bandwidth and/or low latency). For example, video applications could use high bandwidth to ensure flawless streaming in high quality, and VoIP or gaming applications might be forwarded through a low latency channel for better user experience. A recent example for QoS-based network services is a partnership between Microsoft and Comcast, in which traffic to Xbox consoles was routed through a dedicated, high-bandwidth channel that did not interfere with the rest of that home's traffic. Another type of special treatment (that has recently gained popularity) is *zero rating.* Zero rating does not provide any guarantees for traffic performance, but offers financial benefits to users. Certain traffic does not count against monthly data caps; sometimes it does not even require a data plan (zero rating is applied either at home or cellular networks). Several zero-rating services exist today. Music Freedom and BingeOn are programs from T-Mobile in which some music (and respectively video) applications do not count against data caps. Netflix had a partnership with Australian ISPs for streaming traffic to be exempted from caps. Wikipedia Zero and Facebook Zero are similar services for developing countries where access to Wikipedia (or Facebook) does not even require a data plan.

2. Which traffic gets special treatment? Zero-rated or faster applications will have a significant advantage over others, making it hard to compete against. Not surprisingly, which traffic benefits creates a lot of controversy between stakeholders. A long-standing suggestion has been to differentiate specific classes of traffic based on their widely accepted needs (e.g., low latency for VoIP traffic, and high bandwidth for video streaming) or based on a special feature provided by an ISP feature (e.g., zero-rated music streaming). But any form of classification brings along the risk of misclassification. For example, if we were to classify VoIP applications as only those that use the Session Initiation Protocol (SIP) we would miss the most popular ones (such as Skype, Google Hangouts, and Apple FaceTime). Yet if we were to whitelist the very popular VoIP applications, how could a newcomer disrupt the existing market, given the disadvantage? And if we give low latency to VoIP applications, why not do the same for video conferencing or gaming, which have similar latency requirements? Apart from wide classes of traffic, ISPs often provide special treatment for their own content services (e.g., Verizon zero rates its own Go90 mobile video service, and Comcast does the same with its StreamTV service). Similarly, some ISPs partner with other content providers to provide special treatment for their traffic (e.g., Facebook Zero and T-Mobile's Music Freedom and BingeOn).

3. Who pays whom? Flow of money depends on the service under consideration. Very often, participants claim no money exchange; for example, an ISP differentiating his own video service (Verizon and Go90), offering a network service as a free feature (T-Mobile and Music Freedom), or using a popular application to promote and penetrate use of cellular data plans (Facebook Zero). Alternatively, network services can be sponsored; that is, the content provider can pay the ISP for special treatment.

There are numerous network services today that raise concerns around net neutrality all over the world: Music Freedom and BingeOn, sponsored data from Verizon and AT&T, Facebook Zero and Wikipedia Zero, direct partnerships between ISPs and providers such as Spotify and Netflix. In most cases, ISPs define all three dimensions (network treatment, participating traffic, and who pays) through a slow and non-transparent process, rousing fear that they will be able to pick up winners, squeeze out new services, and create an insurmountable barrier to entry for innovative and disruptive new applications. To protect users, net neutrality has devolved to "don't do anything; i.e., treat all traffic the same. Meanwhile, the regulatory landscape is vague and changes from country to country. In February 2015, the FCC's updated Open Internet rules banned fast lanes over the last mile, but let the door open for zero-rating services, which the FCC will examine on a case-by-case manner. Netflix had to withdraw from its zero-rating partnership with Australian ISPs after public backlash from its users. Facebook renamed Facebook Zero to Free Basics and added more content providers to its zero-rating program after several users and Internet activists in India complained. Nevertheless, in February 2016, TRAI (India's regulatory institution) banned all zero-rating services (including Free Basics) that are content-specific, following the paradigm of other countries such as the Netherlands.

There is a recurring pattern whenever such a service is launched, emphasizing the tussle among stakeholders and absence of transparent, clear, and enforceable policies. Users, non-participating companies, and net neutrality advocates vocally complain. Participating companies get bad publicity and occasionally voluntarily withdraw from the service. Regulatory complaints are being filed against ISP practices, and often contestants end up in courts for resolution.

4.2 How network cookies and user preferences will help

This thesis makes two contributions to the net neutrality debate.

First, network cookies provide a policy-free mechanism to express which traffic gets special treatment from the network. Cookies alone do not ensure a user-friendly service; in fact, an ISP could still choose to benefit only its own content. However, cookies provide the technical means to improve status quo. Existing services can become more inclusive, due to low transaction cost and the capability to work in different scenarios (e.g., in the presence of encryption and CDNs). Regulators can better audit ISPs and content providers, as they can easily monitor who has access to cookie descriptors and how. Trust is better due to cookie revocability, and ISPs can experiment with different charging models, given the fine-grained authentication primitives. Further, since cookies are policy-free, each country can dictate its own policy.

Second, and perhaps more importantly, network cookies enable a user-driven approach. While many arguments are made in the name of protecting users, those very users do not seem to have a say in the matter. If we are nervous that ISPs and content providers will make decisions that are 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.

In this chapter I demonstrate the value of network cookies and user preferences in the context of net neutrality through two user studies. The first, focuses on zero-rating services. I examined a popular zero-rating service for music streaming (T-Mobile's Music Freedom) and its properties. I then surveyed $1,000$ smartphone users about their interest and preferences in fast lanes and zero-rating services, and saw how well Music Freedom meets users' needs. Contrary to T-Mobile's claims, Music Freedom includes only a small fraction of music applications desired by users. It also makes the assumption that music applications are more important than others. Given the option to select which application to zero-rate, users chose 106 different applications from different categories (e.g., video, audio, social, news); existing zero-rating services cover a very small fraction of them (~11%).

The second user study is a prototype fast-lane service (Boost) deployed in 161 homes. As mentioned in previous chapters, Boost enables users to 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. Usurprisingly, individual users had their own preferences: 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 (based on Alexa ranking) was 223.

Both studies demonstrate the diverse and long-tailed nature of user preferences, and that users are willing to express them if there is an easy way to do so. In contrast, any one-size-fits-all approach is unlikely to work for most users. The rest of this chapter describes these user studies in more detail.

4.3 Analyzing Zero Rating: T-Mobile's Music Freedom

Music Freedom is a program by T-Mobile for zero-rated music streaming. According to its description, Music Freedom lets users "stream unlimited music from their favorite music services with no data charges," such as data from Spotify, Google Music, and other services whitelisted by T-Mobile do not count against monthly caps. Any music-streaming service is eligible to participate, and there is no associated cost—it is free for content providers and available at no cost to users with certain contracts. T-Mobile uses DPI to detect eligible traffic, through a list of IP addresses, URLs, or SNI fields (a URL exposed during the SSL handshake) provided by each content provider [9]. The details of this particular program highlight issues in many zero-rating services.

4.3.1 Music Freedom does not include all music services

To test T-Mobile's argument that users can stream their favorite music for free I performed a simple experiment. I ran a US-wide online survey with 1000 smartphone owners regarding zero-rating,

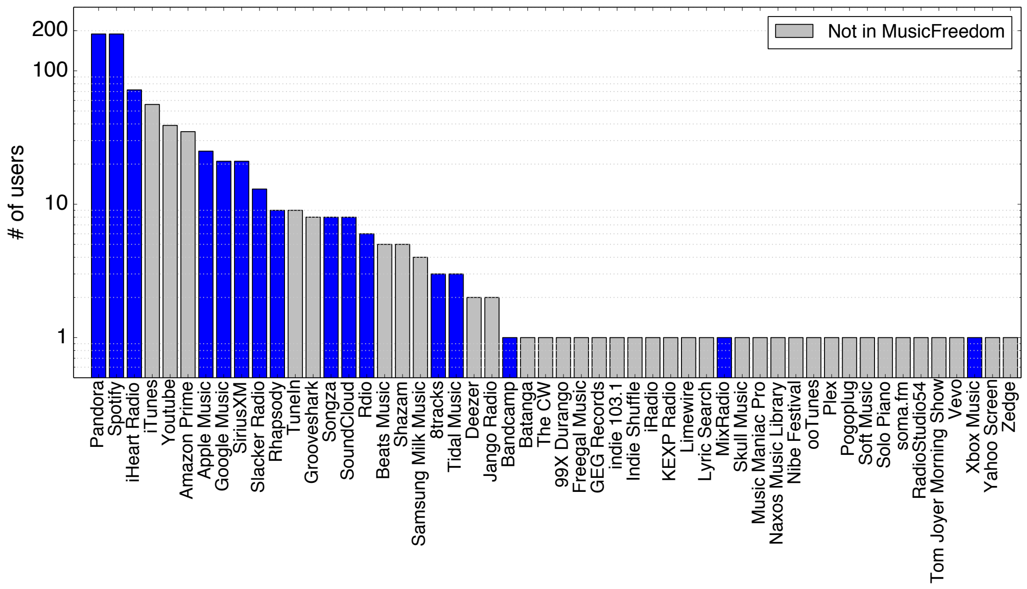


Figure 4.1: *Which music services would you like to see included in Music Freedom (e.g., Spotify)?* 1000 users mentioned 51 different services. 34 of them (66%, greyed-out) do not participate in Music Freedom as of November 2015. Online Survey of 1000 smartphone owners, US-wide, $18 - 65$ year old (Survey Monkey Audience Platform, August 2015). Results for Spotify rounded-down to the second most popular service to remove bias from question.

using SurveyMonkey's Audience Platform.[1] Among participants, 63% were interested in zero-rating programs, which explains why these programs exist. I asked these people which music services they would like to be part of Music Freedom. Many of them selected a small number of popular services (like Spotify and Google Music), but they also chose a large number of less popular ones, for a total of 51 music services. This is what we call a long tail, a well-known and recurring theme in the web.

Only 17 of chosen services were part of Music Freedom as of November 2015.[2] 34 out of 51 services (66%) were not participating. Non-participating services vary in popularity: Amazon Prime Music or TuneIn have hundreds of millions of users, while Naxos Music Library—a service for classical music--has a few thousands users.

Excluded services exist either because T-Mobile did not care to add them, or the services themselves were not interested in being included, or they might not have met the technical criteria set by T-Mobile. [3]

[1]Online Survey of 1000 smartphone owners, US-wide, 18-65 year old (Survey Monkey Audience Platform, August 2015).

[2]This list changes with time. Music Freedom originally launched in June 2014 with 7 services, covering only 14% of music services chosen by users.

[3]While Music Freedom does not publicly list technical requirements for inclusion, BingeOn—a sister zero-rating program by T-Mobile for video—requires traffic to be unencrypted and TCP (explicitly banning encryption and UDP).

4.3.2 Why getting a music service zero rated is hard

T-Mobile claims that any licenced music streaming service is eligible to join the program, but why are most of them excluded today? I believe a critical reason is the high transaction cost to add a new service. Let's look at a real-life example.

I co-ordinated with a radio producer from RockRadio.gr, a Greek radio station. RockRadio sent an e-mail to T-Mobile (following their online workflow) and asked to be included in Music Freedom. There was no reply after several months and many reminders. The reason is that T-Mobile has to manually add RockRadio to Music Freedom, and this is hard. First, somebody from RockRadio.gr needs to contact T-Mobile and ask to be included. Then, a product manager from T-Mobile has to check RockRadio.gr and decide whether its eligible for Music Freedom. Finally, T-Mobile has to be able to detect traffic from RockRadio.gr and exempt it from data caps. This is tricky, and requires co-ordination between product managers and engineers on both sides. T-Mobile could ask RockRadio to send traffic unencrypted using HTTP, or use a small, predefined number of IP addresses to stream music (both guidelines ignore best practices for modern services, namely use of https and content distribution networks). This is a heavyweight, manual process for both sides and, given the small number of RockRadio fans in the US, there is no business incentive for T-Mobile to spend resources to add it in its program. Consequently, only the few very popular services can be included in the program, ignoring the long tail of user preferences.

There have been several similar cases that failed or struggled to get attention from T-Mobile. 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 [40]. Even if a service eventually gets included, 18 months is a very long time and can affect the viability of a new Internet service.

4.3.3 How to improve Music Freedom

The first step to improve Music Freedom is to make it more inclusive. Cookies reduce the overhead to add a new participant (both for T-Mobile and content providers), so more services will be included. T-Mobile just has to provide a cookie descriptor to participants and match associated cookies, instead of understanding traffic properties for each of them. Content providers will not have to change their IP addresses or adjust their URL structure to be detected by T-Mobile, which is easier and provides more trust. For example, if content providers suspect that T-Mobile rate-limits their service upon detection, they can perform an A/B experiment, by not putting cookies to a small percentage of traffic.

The second step is to make the process of adding a new participant straightforward

YouTube uses HTTPS and a custom UDP-based protocol called QUIC. Even if T-Mobile and YouTube manage to workaround these limitations, this will be a showstopper for a small start-up or academic group experimenting with new streaming technologies.

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[4]	25
Maps	4		
Browser	3		
Education	2		
Other	24		

Table 4.1: *If you could choose a single application to be zero-rated for you, which one would that be?* 1000 users chose 106 different applications, from various categories and popularity buckets. I used the number of downloads at the Google Play Store as a metric for popularity. Online Survey of 1000 smartphone owners, US-wide, 18-65 years old (Survey Monkey Audience Platform, August 2015)

and easily auditable. 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. 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 [23].

Third—and perhaps most important—is to let people decide how they want to use their own zero-rating plans. Several users from my survey responded that they were not interested in music, but they were interested in zero-rating other applications. When asked to pick a single application that they would like to zero-rate, many more profiles became visible besides music fans. Some wanted to watch videos during their daily commute, others preferred social networks or news applications, a few wanted to do their online courses while on the go, and some were surrounded by WiFi and they would let their data plans go if they could only get access to Google Maps directions. Table 4.1 summarizes the categories chosen by users. As with music, many applications were listed for each category. Some of these applications were very popular (hundreds of millions of downloads in the Google Play Store), and others had a few thousands downloads. Instead of guessing what users want, T-Mobile could be much better off just by letting users decide what they want. For example, it could let users pick their own favorite application to be exempt, or give them access to a zero-rated but slow lane (e.g., rate limited to 200Kbps) and let them send any traffic they want.

4.4 Boost: a user-driven fast lane

Boost is a user-driven fast lane for home networks. Fast lanes over the last mile of the network have been at the core of the net neutrality debate since the very beginning—in fact,the recent FCC and European Union rules focused merely on fast lanes, excluding zero-rating services. We have already seen Boost as a use case for personalized networks in Chapters §2 and §3. Here I will discuss an actual implementation and deployment of such a service.

I prototyped and deployed Boost to understand how users would customize their networks by giving them the choice to decide which traffic will 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 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 router.

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.

4.4.1 Boost Agent

The agent is a Chrome browser extension, and lets users decide which traffic to boost in the following ways:
1. 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).
2. Always Boost a website. Traffic related to a website gets priority.[5] The setting is remembered; whenever a user visits this website, the agent informs the home router of related flows.

Figure 4.2 shows a screenshot from the Boost extension while navigating to Stanford's online educational platform. The browser provides an interesting vantage point. Users can boost any traffic they like, not just a shortlist 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 the network 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 (recall matching accuracy for cookies and DPI from §3). Cookies with support from an agent can bridge the gap between users and networks.

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

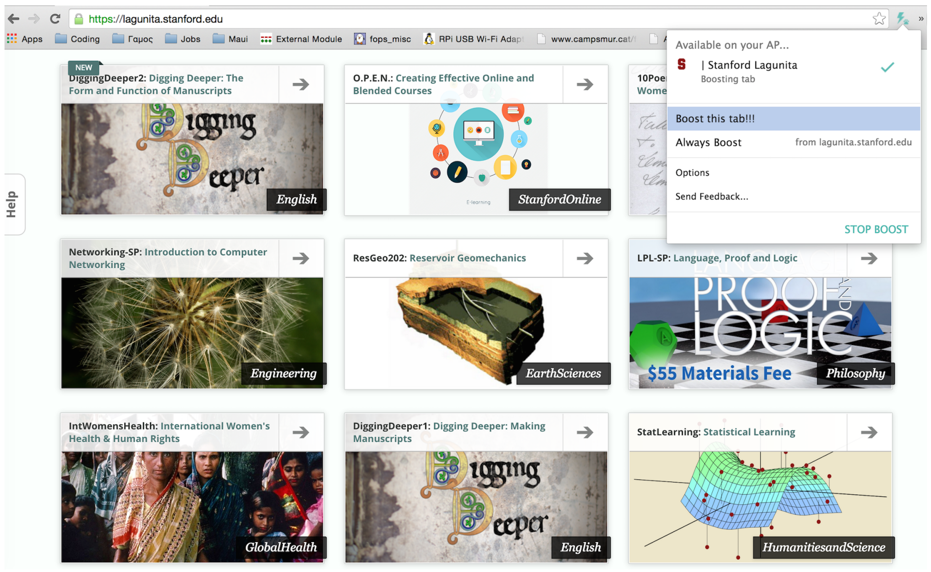

https://lagunita.stanford.edu
Available on your AP...
| Stanford Lagunita
Boosting tab
Boost this tab!!!
Always Boost
from lagunita.stanford.edu
Options
Send Feedback...
STOP BOOST
DiggingDeeper2: Digging Deeper: The Form and Function of Manuscripts
O.P.E.N.: Creating Effective Online and Blended Courses
Networking-SP: Introduction to Computer Networking
ResGeo202: Reservoir Geomechanics
LPL-SP: Language, Proof and Logic
IntWomensHealth: International Women's Health & Human Rights
DiggingDeeper1: Digging Deeper: Making Manuscripts
StatLearning: Statistical Learning
English
StanfordOnline
Engineering
EarthSciences
Philosophy
$55 Materials Fee
GlobalHealth
HumanitiesandScience
Help

Figure 4.2: User Interface for Boost.

Focusing on web traffic and the application layer provides an easy place to start studying users' preferences, and a relatively easy deployment path. Cookie-related functionality is added without requiring any kernel, server, or protocol support, and we can develop and deploy actual services on top of the Chrome browser, which works with standard HTTP(S) traffic. Boost can also leverage available user context to make preferences more meaningful: They can relate to content, as traffic is still unencrypted, or take into account the active tab of the browser. They can also 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 toward a more generic design choice: *Should we place an agent closer to users and applications, or declare them on a more network-centric approach (e.g., at a network gateway)?* I chose the former, as it can capture user preferences and user context in a much better way.

Cookies are inserted 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, cookies are text-based using base64 encoding. The user agent intercepts outgoing `http(s)` requests using Chrome's extension API, extracts related metadata (e.g., which tab generated it, the url in the address bar), and if the request matches with user preferences it adds a boost cookie to it. While adding an `http` header is straightforward, in order to add a TLS extension I had to modify Chrome's

SSL/TLS library.[6]

To start boosting traffic, the agent issues a boost request to a well-known server (located at the home router) 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, I followed a simple *last one wins* policy, and expect users to resolve conflicts over a beer, if this is not enough.

4.4.2 Boost Daemon and the Cookie Server

Cookie descriptors are available at a server already known to Boost agents. They are stored in a persistent SQL database and users can acquire them through a JSON API. I implemented a python-based daemon on the WiFi router that sniffs traffic, looks up cookies, and enforces the desired QoS service. The daemon sniffs the first three 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, it adds this and the reverse flow to the fast lane by using a set of standard tools in the WiFi router (iptables and Linux `tc`).

To provision the path for boosted traffic, the daemon i) uses the high-bandwidth wireless WMM queue, and ii) throttles 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 is estimated using periodic active tests. Figure 4.3 shows a scenario for a 6Mbps connection, where non-boosted trafic is throttled to 1Mbps.

I should emphasize that this Boost prototype is far from perfect; for example, it is not work-conserving if the user does not use the fast lane. This reflects to the original goal to understand user's preferences rather than build the perfect Boost service.

4.4.3 Measurements of Users' Preferences

The 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 "dogfood" test of Google's OnHub home WiFi router. 161 users (40%) installed the extension in their browsers.

Figure 4.4 shows the websites prioritized by users, the number of clients that boosted a given domain, and their popularity, and demonstrates in practice the long-tailed user behavior we saw earlier. Many users boosted popular U.S. video websites (Netflix, YouTube, HBO, ABC, Fox, and ESPN). The long tail also includes less popular sites, such as a VoIP service and 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 ($3Mb/s$) due to neighborhood constructions and

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

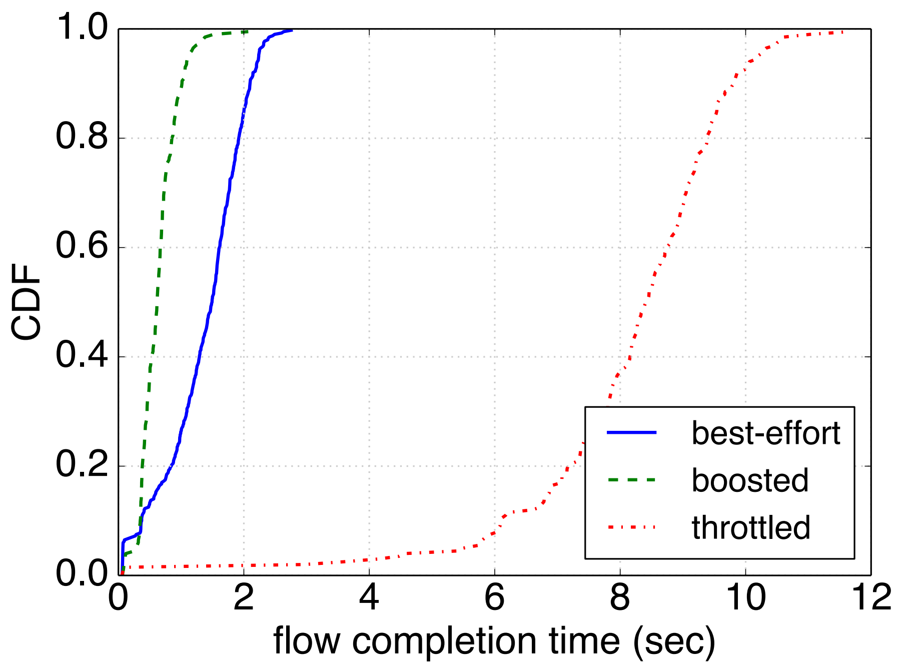


Figure 4.3: Flow completion time for a 300KB flow in the presence of background traffic.

occasionally wanted to dedicate all resources to a specific task (e.g., stream a video or join an online meeting); another wanted to prioritize business-related calls; a third one wanted to prioritize Netflix on his TV, but not Netflix on his kids' tablets; and another wanted the active tab of the browser to always have priority.

4.4.4 Structuring user behavior through incentives

A frequent question regarding Boost is "why not boost everything?" This is a natural concern: Users would make everything faster and cheaper if they had the choice, but this does not necessarily makes economic sense for the operators who try to monetize their infrastructure. When ISPs define a network service such as a fast lane, they try to ensure reasonable use by choosing which traffic benefits from it.

Giving users the chance to express their preferences does not exclude reasonable use, but rather suggests that ISPs should give the right incentives to users. For example, my Boost deployment is a zero-sum game—to Boost some traffic, users have to slow down other traffic. ISPs could provide different incentives. For example, they could give users a monthly quota for extra bandwidth, or ask them to pay per-hour (e.g., a user could have a $10Mbps$ connection, but get short access to a $30Mbps$ lane to watch a movie in $4k$ definition).

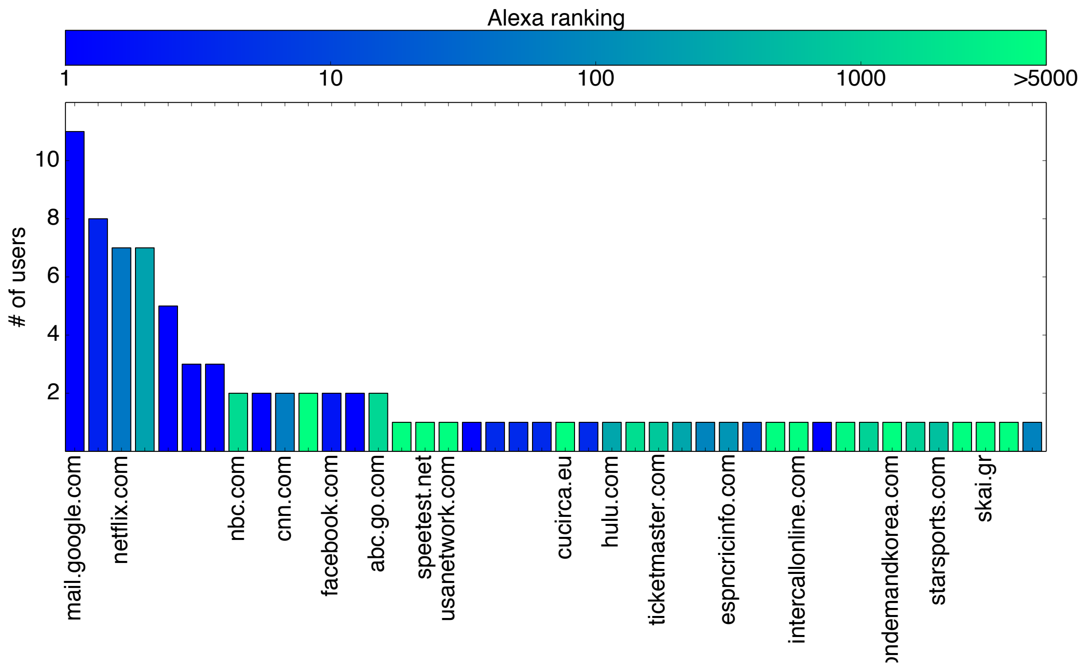


Figure 4.4: If given the choice, which websites would home users prioritize? The preferences have a long tail: 43% of the preferences are unique, with a median popularity index of 223.

A missing piece is to better understand ISP incentives. Is it better for an ISP to boost or zero rate a single application, or let each user choose the one they prefer? Does it make sense to provide a slow but zero-rated lane along with a fast, and let users decide how to distribute traffic between them? Would boosting a single application lead to more expensive high bandwidth plans? The incentives and economics for such services will play a significant role to better understand their viability and how to structure them. I expect that as these services become popular, we will have more data to reasonable argue about user and ISP behavior.

4.5 Related Work

There is a vast literature covering net neutrality, the majority of which is from an economics and policy perspective.

Net neutrality advocates argue that fast lanes and zero rating will hurt innovation and user choice [20, 44]. Opponents of net neutrality-related services claim that services like zero rating and fast lanes improve competition and user choice, reduce cost, and can dramatically increase Internet connectivity in developing countries [26, 35]. Policy makers in different countries define (often controversial) policies to regulate services such as fast lanes and zero-rating [3, 10, 13].

This is not the first attempt to involve users more closely and ask them what they want—such during the recent update in the FCCs Open Internet rules. AT&T suggested allowing traffic differentiation upon user request, and van Schewick claimed that the only accepted way to engage in traffic discrimination is for the process to be driven by users [44, 8]. Net neutrality advocates welcomed the approach, but raised concerns on whether the actual implementation of user-driven prioritization could limit user choice [30], and the proposal was ultimately rejected. Network cookies directly address these concerns, as they provide a simple, policy-free mechanism to express user preferences and allow regulators to decide and subsequently audit the policy simply by monitoring who gets access to cookie descriptors and how. Moreover, the user studies demonstrate that letting users decide is both practical and beneficial, and provide insights on how to better preserve the users' interests.

The recent increase in zero-rating programs led to detailed analysis of existing services and their properties. Similar to my study on Music Freedom, van Schewick provided an in-depth analysis on BingeOn, T-Mobile's zero-rated video service [45]. Other studies have looked into how zero-rated services affect user behavior. For example, a study in Nigeria and Indonesia showed that 60% of people participating in Facebook's zero-rating programs thought that "Facebook is the Internet" [7]. Similarly, using a tool that collects usage data from mobile phones (`mySpeedTest` [17, 4], researchers showed that when some services are zero-rated, users eventually prefer them compared to other services. Over the next few years, as zero-rating services become mainstream and we get more insights into how they affect user behavior, I expect regulators, ISPs and content providers to experiment with a variety of different policies. Network cookies provide the flexibility to enable a wide range of policies, from the very exclusive, all the way to ones that enable users to customize the network for their own needs.

Chapter 5

Personal WiFi Networks

Chapter 3 introduced network cookies, a mechanism for expressing user preferences to the network, and Chapter 4 demonstrated how users can choose which traffic to send through specially provisioned "lanes," overcoming the limitations imposed by the net neutrality debate.

This chapter presents a more holistic view of personal networks through the following question: Can we enable a user to define her desired network experience (i.e., her personal network) once, and let this personal network follow her wherever she goes? I explore this question through personal WiFi networks, a first step toward a fully personalized network experience. I start by building the case for personal WiFi networks and then describe their architecture and core features. Finally, I demonstrate their feasibility through a prototype deployment in a university dorm.

5.1 The Case for Personal WiFi Networks

WiFi has become the dominant mode of personal connectivity around the world. Users expect WiFi everywhere, at home, in coffee shops, and even on planes. According to a recent study [31], daily mobile use reached about six hours in 2012, of which 90% was via WiFi (and only 10% cellular).

Yet, despite its ubiquity, user experience with WiFi is highly variable and fragmented. The reason is that users consume WiFi from a wide variety of deployments. They connect through their home WiFi networks less than 40% of the time [31]; the rest is split among different locations (the workplace, a friend's WiFi, retail stores, public hotspots, travel locations). Each deployment is independently set up and managed, which leads to very different user experiences. Users use different methods and credentials to authenticate to each network; their ability to reach resources is subject to the networks to which they are connected; and they have very little, if any, control over policies such as firewalls and QoS settings.

Network personalization has the potential to eliminate this variability. Users will be much better served by a *personal network* abstraction that delivers a consistent, customized, and user-centric experience everywhere. Users define their desired network experience once, and then their personal networks follow them wherever they go. With personal networks, users would *always connect using their own SSID and password* to any WiFi infrastructure, without needing to navigate through different networks until they find one that works, or ask for the WiFi password. They would be able to *reach all devices to which they are granted access*, without needing to manually setup VPNs or restrict themselves to cloud-based applications. And their *own policies*, such as firewalls and parental controls, would apply everywhere, not just at home.

Let's look at this from a user's perspective. Alice can connect to her home WiFi Access Point (AP) and to a set of public hotspots operated by an ISP (Figure 5.1). She stores all of her documents in a drive at home, which she can access from her phone and laptop. When Alice goes to a coffee shop, the network detects her and presents her with her own network (i.e., "SSID = Alice"). Alice connects to it and can access her documents from her hard drive even when she is connected to a public hotspot, without requiring a third-party cloud-based service, such as Dropbox or Google Drive. When Alice visits her friend Bob for dinner, she can connect to Bob's network; Bob's AP detects Alice and connects her to her personal network. Alice can access her hard drive back at home and open photos from her latest trip. Bob's television (TV) is also visible on Alice's personal network; her phone detects a nearby TV and displays photos directly on it.

The main challenge to realizing personal networks stems from WiFi's AP-centric approach. By setting a few AP parameters, anyone can set up a WiFi network and be online in a few minutes. While this simplicity is one of the primary reasons for WiFi's popularity, it leads to a distributed, statically configured, user-agnostic WiFi infrastructure; as users, our experiences vary greatly from one WiFi network to another. The natural question to ask is, *"Can we serve millions of personalized networks on a WiFi infrastructure while retaining WiFi's deployment simplicity?"*

The rest of this chapter explores answers to this question while considering three core features of personal WiFi networks: i) user-centric WiFi discovery and authentication, ii) user-centric reachability, and iii) user-centric resource sharing. I start by considering the high-level architecture and then discuss each of these features in detail.

5.2 Architecture

The high-level architecture is familiar from modern cloud-controlled WiFi systems (Figure 5.1): WiFi APs implement low-level WiFi tasks (beaconing, acknowledgements, rate selection, traffic forwarding), and are all connected to centralized WiFi controllers that take care of WiFi management packets (probe, authentication, and association requests). The first main difference is in how users connect to a WiFi network. When Alice's device scans for available networks, she sends a specialized probe request that securely identifies her (through a "probe cookie"). A centralized controller checks her request against a personal network directory; if Alice is recognized and has permission for access,

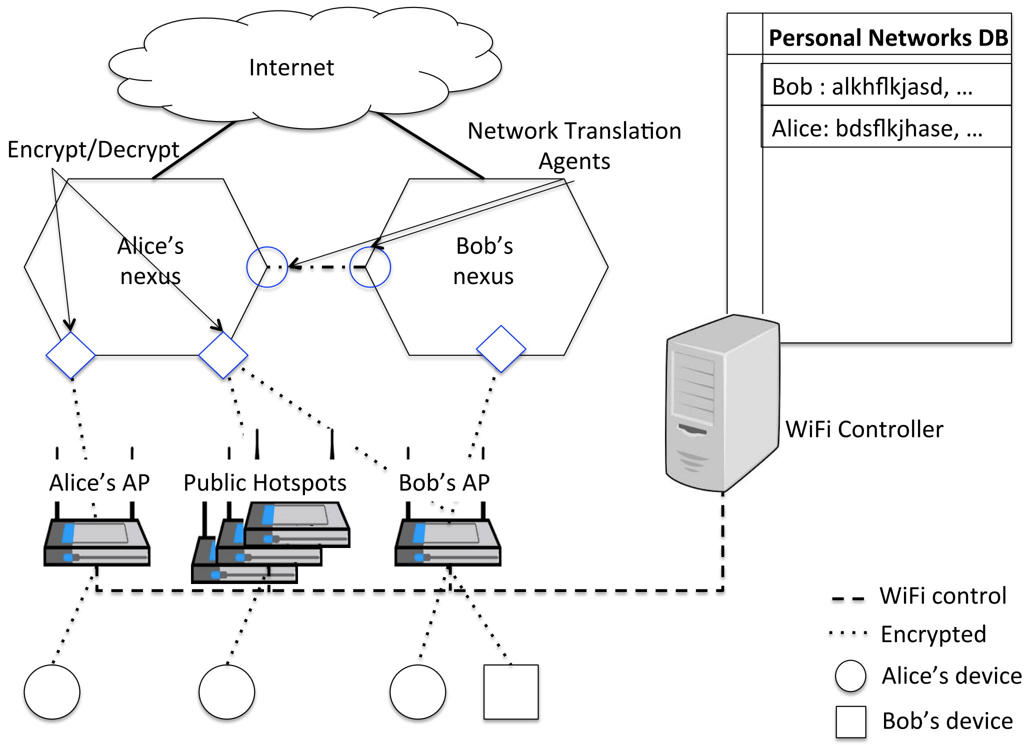


Figure 5.1: Overall architecture for personal networks. Each user has a dedicated nexus that provides basic network services (bridging, NAT, DHCP, WPA encryption/decryption). A centralized WiFi controller personalizes WiFi discovery and installs necessary state for each PN. Network translation agents compose multiple private networks and resolve namespace conflicts.

the network tells her that her personal network is available (with "SSID = Alice"). I describe the network discovery workflow in more detail in §5.3, and in §5.6, I demonstrate from a university dorm deployment that the interaction between the APs and the controllers is fast enough.

When Alice has associated with and authenticated to her personal network, the controller creates a tunnel to Alice's personal *nexus*, which lives in the cloud, acts as her private L2 bridge, and provides basic network services, such as NAT, DHCP, and WPA key exchange. All of Alice's traffic to and from the AP is forwarded through her nexus, which implements her ACLs, parental controls, and other policies for her personal network. Her traffic is encrypted to and from the nexus so that a compromised AP cannot sniff her data. Alice is able to reach all of her devices because they connect to her nexus in the same way, regardless of where they are connected from. To share a device (Bob letting Alice use his TV), a link is created between Bob's nexus and Alice's, and it lets traffic flow between them (either L2 or L3).

5.3 User-Centric Discovery and Authentication

With personal networks, each user discovers, authenticates to, and connects to the network using her own SSID and password. To enable this, I suggest turning WiFi's discovery process into a dynamic and personalized discovery workflow. We need two innovations to make this happen. First, we need to eliminate broadcast, beacon-based discovery and dynamically expose a personalized view for each user through *unicast scanning*. Second, we need to allow users to reveal their identities and discover their own networks. As we will see, this is similar to the workflow we've seen in earlier chapters: Users express a preference (*show me my personal network*) for networks they trust, and they want to express it in a way that is secure and revocable.

5.3.1 Unicast Discovery

Traditionally, WiFi clients discover available networks in two ways: *active* scanning, where a client asks for a list of SSIDs through a probe request; and *passive* scanning, where a client overhears beacons broadcasted by the APs. It is not practical for a WiFi AP to broadcast a list of SSIDs for every user in the system—it will not scale to thousands or millions of users. Previous work has already demonstrated that broadcast beacons can consume a significant fraction of the airtime capacity [46]. To address scalability concerns, I decouple network discovery (for which I use unicast *probes*) from link monitoring and synchronization (for which I use broadcast *beacons* without SSID information). The process starts when the client sends a probe request, which results in a unicast response along with the user's personal SSID. Because discovery is unicast, we can decide which network (e.g., SSID) to expose and to whom, a critical piece of information for personalization and scaling. We need only one unicast probe and one SSID for each connected user. Unicast discovery does not consume extra airtime, and therefore, we can offer each user her own SSID without scalability concerns.

5.3.2 User Identity and Network Cookies

To discover their personal networks, users need to reveal their identity. In essence, users express a preference (*"show me my personal network"*) to networks they trust, and they want to express it in a way that is secure, is revocable, and respects their privacy. Trust, revocability, and user privacy should sound familiar by now. Let's see how network cookies can facilitate personalized discovery while protecting user privacy.

Users can reveal their identities and request their networks by attaching network cookies in their probe requests while scanning for available networks. Besides user identity, cookies introduce two basic properties in the discovery process: (i) *privacy* (users reveal their presence only to their preferred infrastructures, preventing tracking from third parties), and (ii) *revocability*—a user can stop sharing her presence with a network provider anytime (by no longer attaching the relevant cookie to her requests), much in the same way that she manages which mobile applications can access her location. They can also convey other properties through cookies. For example they can share information with the network to improve the discovery process for performance (e.g., share the type of device or operating system OS, or whether they support for 5GHz to optimize band steering).

Users can frequently update their cookie descriptors so that the 64-bit ID cannot be tracked. Alternatively, they could hide it completely by encrypting their cookies using the network operator's public key. Furthermore, descriptor attributes can describe when to use a cookie (e.g., through an expiration date, or in a geographical area where a cookie can be used).

Cookies are inserted as information elements in probe requests, existing fields used to add extra information during the discovery process (e.g., vendor extensions). [1] When the AP receives a probe request, it sends the cookie to its controller, which, in turn, looks it up in a database, and—if valid—finds the user (and hence the personal network and SSID) and replies with a probe response with this user's SSID.

This is in contrast with existing systems that use MAC addresses as user identifiers during the discovery phase (e.g., for WiFi virtualization [27, 43, 46] but also for user tracking [28]). Unlike cookies, MAC addresses are static, unique, and can be easily spoofed, and their use for identifying users raises serious privacy concerns. In response, operating systems (iOS, Linux) now randomize their MAC address during scanning to protect the user's privacy [32], making MAC-based approaches impractical. [2]

[1] Cookies can be inserted via the WiFi connection manager, a userspace utility that manages WiFi connectivity.

[2] I have personally experienced these concerns when dependencies on MAC addresses became a showstopper for an early prototype deployment of personal networks with a commercial partner.

5.4 User-Centric Reachability

While the above provides uniform discovery and authentication, a big part of our network experience is *reachability.* Users expect to be able to reach and interact with their devices from anywhere, as if they are on their own private networks (like at home). But these devices sit behind NATs and firewalls (e.g., accessing a home computer from work, or a freind's printer). To access these resources today, we need to connect all devices to the same physical network, use an intermediary in the cloud (e.g., Dropcam, Dropbox), or use a VPN (e.g., to connect to work from home).

Personal WiFi networks connect users to their resources at the network level using a cloud-based nexus. When a device associates with an AP, we create a link through a tunnel to the user's nexus. The nexus acts as a L2 bridge, connecting all devices, through which they can therefore directly talk to one another or use standard multicast/broadcast schemes. Discovery protocols that depend on L2 connectivity (e.g., mDNS), work out-of-the box. One can imagine many new features and applications exploiting such reachability: for example, a "mute application" that mutes all of the devices belonging to a user at the touch of a button. It would be much easier to do this by sending a single, broadcast packet instead of having to keep track of all devices and deploy a cloud-component through which messages are routed.

5.5 User-Centric Sharing

Besides their own devices, users often access shared resources from work (e.g., a personal desktop machine), devices shared from a friend (a TV or just access to a WiFi AP), or devices located in physical proximity (a nearby printer). Traditionally, to reach these resources, users would have to physically (or remotely) connect to the appropriate network and give away their home WiFi passwords.

In contrast, a personal network combines all resources available to a user, by composing multiple (private) networks together, and allows users to request, grant, and revoke access at a user level. Each device belongs to a single personal network (for example, Bob's TV is connected to his personal network). The device is therefore connected to a single nexus. To grant another user access to the device, we allow traffic to flow from one nexus to the other, then implement the ACL privately in the nexus to enforce access to the particular device. For example, Bob's nexus contains an ACL indicating that Alice is allowed to access his TV. If Alice sends traffic to Bob's TV using a private address in her own nexus, it will be forwarded to Bob's nexus, which will translate the address into Bob's private address space and forward the traffic to the TV. Instead of sharing her SSID and password, the owner grants access to another user for one or more devices.

When composing personal networks we need to take into account potential conflicts in private namespaces (IP address, DNS, etc). Each personal network reserves address space for remote resources. I use network translation agents to dynamically map a remote device to look like it is part of a user's personal network.

We can augment the sharing workflow with higher-level semantics for user convenience, e.g., using social networks to share resources with a group of friends, or detect physical proximity and automatically present a TV to a guest's network. We might require explicit approval for a device to be shared or make it publicly available to anyone close by.

5.6 The BeHop Prototype

To evaluate personal WiFi networks, I built BeHop, a prototype system for personal WiFi networks, and deployed it in a student dormitory at Stanford University.

5.6.1 Implementation

BeHop uses a commodity AP (dual-band, Atheros-based NetGear WNDRMACv2), which I reprogrammed with Openwrt and Open vSwitch (OVS) [39].

Using OVS, the AP forwards all WiFi management packets (probe, authentication, association requests) to a centralized WiFi controller implemented using POX [36]. The controller maintains a directory of personal networks in a database. Personal networks are indexed using different keys (username, MAC address, probe cookie ID), and for each of them, I store the user-defined SSID, the address of its nexus, and the list of APs that this user is allowed to access. When the controller receives a probe request it finds the related personal network, responds to the AP and client, and upon association, installs the client state in the AP (e.g., supported WiFi rates), and sets up an encrypted tunnel to the user's nexus.

To generate cookies, I extended *wpa_supplicant*, a WiFi connection manager for Linux and Android devices. Changes are in user space and do not require any kernel modifications. Moreover, they do not require any changes to the WiFi protocol. Cookies are inserted as information elements, a generic extension field, to exchange information during the discovery phase (such as supported features, hardware capabilities, vendor extensions, the SSID). Probe cookies introduce a negligible overhead to the discovery process: In a university dorm deployment, probe requests occupy $< 1\%$ of total airtime, and cookies add an extra 0.1%. Because cookies require modifications to the client stack, I also used MAC addresses to connect legacy devices to a personal network.

Each user nexus is implemented as an OVS bridge in a server. To link devices to their nexuses, and to link one nexus to another, I used GRE tunnels. Traffic coming from a different nexus goes through a NAT agent to bring everything under the same IP address space (our prototype only supports IP addresses for now).

The basic workflow for personal WiFi networks is shown in Figure 5.2. Physical APs remain responsible for low-level functionality (rate adaptation, retransmission, rate adaptation, beaconing). Discovery goes through a centralized controller, which maintains a database of all personal networks.

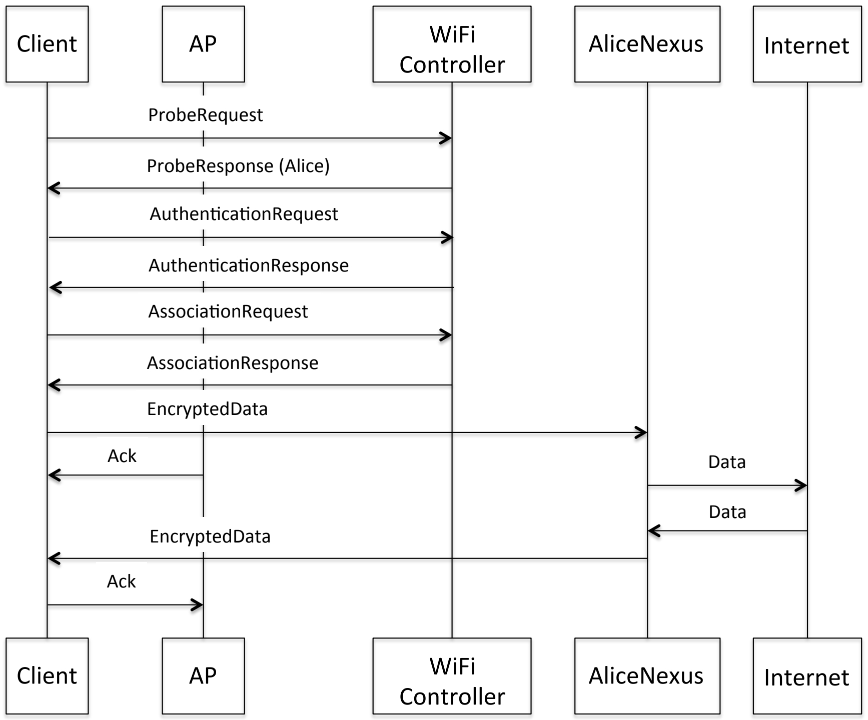


Figure 5.2: WiFi workflow for personal networks. Management packets are handled by a centralized controller. Encryption/decryption is implemented at the user's nexus, to account for non-trusted APs. AP deals with low-level tasks such as ACK generation, beacons, and rate-control.

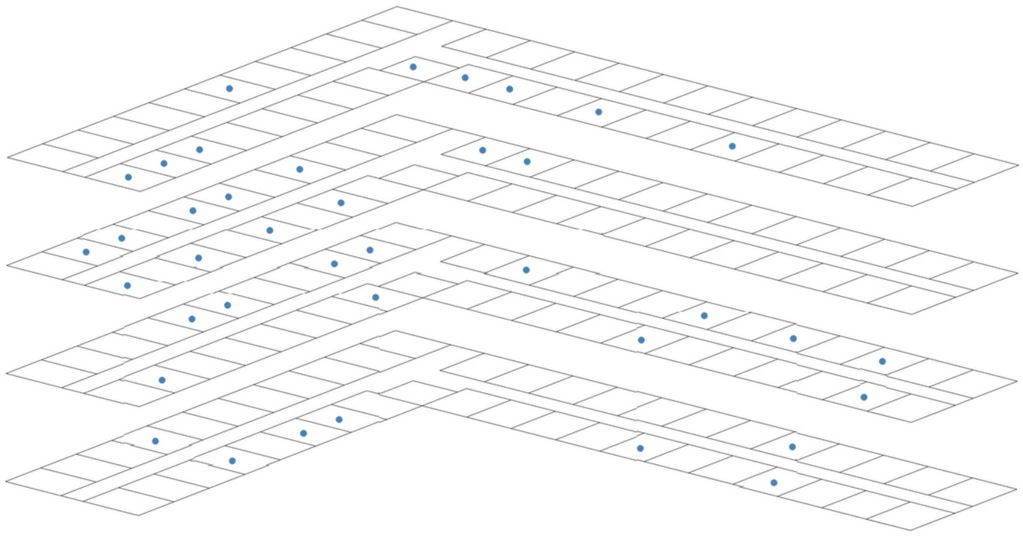


Figure 5.3: BeHop deployment in a four-story dorm at Stanford University. Each point represents an AP within a studio apartment.

WiFi management (authentication, association) can take place in the AP or in the centralized control plane. Finally, to protect users' privacy from potentially compromised APs, the key exchange and encryption/decryption of data traffic take place at the user's nexus.

5.6.2 Deployment and Evaluation

I deployed BeHop in a four-story dorm at Stanford University with 150 studio apartments (Figure 5.3). It served real traffic from 39 users and more than 90 client devices (laptops, phones, tablets) between December 2013 and December 2014. Users willing to participate each had to install an AP in her room (I provided the AP along with other participation incentives such as free Netflix access for the duration of the experiment). The building was already covered by a campus-wide network using the Cisco LWAPP technology (25 dual-band APs installed in the building). BeHop had a much denser deployment, with 50% more APs deployed. For development purposes, I also ran a smaller, three-AP network at the Computer Science Department at Stanford. [3]

To identify users, I used a combination of network cookies (for a small number of self-owned devices) and MAC addresses (to avoid changes on legacy user devices). I got a list of users and their devices' MAC addresses through Stanford's Residential Computing department. Their hesitation to provide access to MAC addresses for privacy reasons acted as an additional motivation for the

[3]For the dorm deployment, all APs connected to a wired switch, which was also used as a nexus and bridged devices from the same user together. BeHop did not provide device-sharing functionality to users, and DHCP/NAT services were provided by the upstream network at Stanford.

development of cookies as a mechanism. A similar MAC-based deployment effort with a commercial partner was blocked due to privacy concerns, further emphasizing the need for better privacy primitives.

By default, all clients can connect through any physical AP, but we can easily change this policy by giving each user access to a different subset of APs. This highlights that personalizing networks can co-exist with control primitives by owners and operators. For example, an AP owner might choose to prevent others from using her physical AP. We will revisit the co-existence of personal WiFi networks with operator control in §5.6.4. Users can access only their own devices when connected to BeHop; many found this feature to be useful, as the existing university network provided a flat L2 network, and users could discover and use devices (such as Chromecasts) from other apartments in the building.

5.6.3 WiFi Management Overhead

BeHop forwards all WiFi management to a centralized controller and therefore raises concerns about overhead and latency. WiFi management traffic generated a peak load of 630 packets/second (90% of the time it is under 200*pps* and includes traffic from all clients belonging to 150 users in the building, not just BeHop users). The WiFi APs and the controller are on the same university campus, and latency is small (< 1 ms). In a scenario with higher latency, responding to WiFi probe requests might become a concern. During the scanning process, clients probe each of 35 channels and wait for responses. I examined the scanning profiles of different devices (Figure 5.4) and found that sending a response within 25ms leads to fast network discovery, all of which should be doable by a remote controller. The diversity in scanning profiles is worth noting: Time spent and number of probes per channel, time between probes, frequency of scanning, and channel rounds per scan vary significantly for different devices. Even for the same device, scanning behavior might change based on various reasons (e.g., 2GHz or 5GHz band, screen on or off, battery status, list of remembered networks). While most devices implement a loss-tolerant discovery process, the main shortcoming of slow responses is that a device might connect to another network if it cannot detect the preferred one within a couple of scanning rounds. A faster approach would be to cache probes from specific users and to send subsequent probe responses from the local AP, although it would require keeping probe cookies unchanged for a caching window or move decoding logic to the AP. Authentication and association requests are less sensitive to latency, with a typical timeout of one second.

5.6.4 Infrastructure-Centric Performance Optimization

At first, personal networks might seem at odds with infrastructure-centric performance optimization—if networks are user-centric, how can an operator optimize an infrastructure that spans multiple independent networks? In fact, personal WiFi networks allow operators to optimize performance while exposing a different network for each individual user. For example, an operator could decide

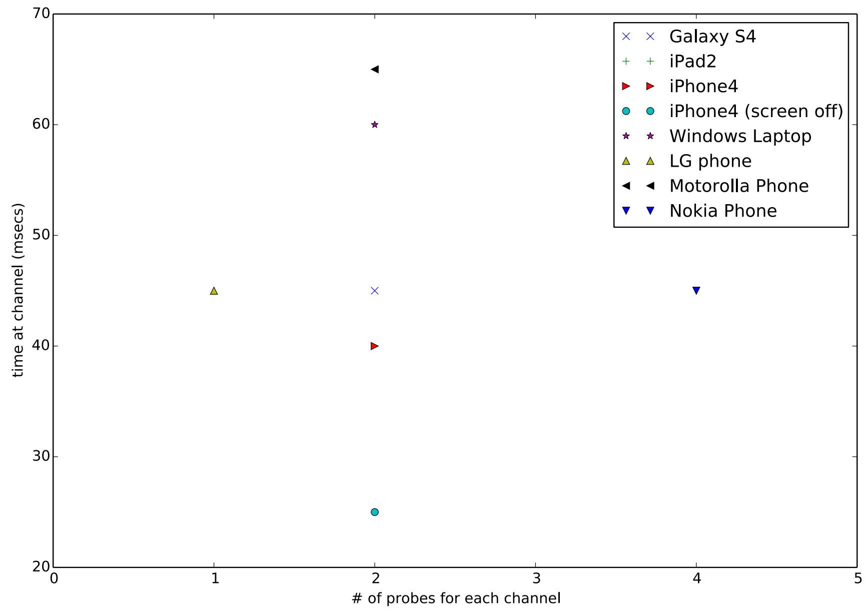


Figure 5.4: WiFi scanning profile for different devices.

which APs to present to each user and then steer them to the desired band or AP for better spectrum efficiency. Similarly, an AP owner could list which users are allowed access through her own AP.

To demonstrate how personal networks can co-exist with operator or owner control, I looked at a specific usecase: how to use the 5GHz band to improve perfomance on a dense WiFi network.

Better use of 5GHz is a common strategy for WiFi network operators to leverage less interference and wider channels available on this band. Doing this is not straightforward—many clients do not support the use of the 5GHz band and the 5GHz band penetrates walls more poorly than the 2GHz band does, making it hard to provide good coverage. As a result, we operate WiFi networks on a "dual-band" mode, where each AP is configured to serve on both bands. Due to WiFi's distributed nature, client devices decide how to connect to the network. They do not always make the best decisions. They might use the 2GHz band even when a less-loaded 5GHz connection is available, or stick to a 5GHz link that provides poor connectivity. Vendors and operators try to facilitate the use of 5GHz through WiFi management features often referred to as band steering.

To better understand the co-existence of personal networks with network-wide optimization, I used BeHop to implement a simple 5GHz steering experiment and study its performance. For the purposes of the experiment, I operated BeHop in three different modes:

BeHop-2GHz: All APs are configured for 2.4GHz-only operation, and each device chooses the AP to which to connect. This acts as the main baseline.

BeHop-Dual-Band: APs are set for dual-band operation. Clients decide the band (and AP) to which they will connect.
BeHop-Bandsteering: APs are set for dual-band operation. 5GHz-capable clients are forced to connect to the 5GHz band by responding only to 5GHz probe requests.

Setting up the desired mode of operation requires minimum effort and validates BeHop's flexibility and ease of programming. All I had to do was overwrite the handling of "probe request" packets in the controller and apply this to the desired clients (e.g. the ones who support 5GHz). Users still connected to their own networks. I did not have to make any changes in the APs. Going from 2.4GHz to band steering required fewer than 20 lines of code to program the controller.

As a reference, I also compared results with the performance of the legacy network in an identical building across the street, supporting dual-radio operation and band steering. Each experiment ran for a full day (March 18, March 22, and March 24, 2014, respectively). For measurements, I collected all packet headers after anonymization.

I used *packet delivery time* (PDT) as a metric for characterizing the performance of the network. PDT is calculated as follows: Using the wired switch behind the APs as the vantage point, I calculate the time between a TCP packet coming on the downlink, and its respective TCP acknowledgement from the client. PDT captures the actual traffic from users and accounts for everything that might degrade performance in the WiFi infrastructure: buffer build-up on the AP due to a link/channel slowdown, wireless retransmissions, channel congestion that may lead the client or AP to back off, etc.

Figure 5.5 summarizes PDT for the different scenarios. I briefly discuss the main insights:
Personal networks and network optimization: At a high level, personal WiFi networks can co-exist with network-wide management, e.g., to improve performance. Users can still connect by using their own credentials and accessing their own devices, even when connecting through third-party APs. At the same time, the infrastructure operator can apply network-wide optimizations to improve performance and spectrum efficiency.
Personal networks leverage density: Personal WiFi networks can leverage high density in existing deployments (e.g., a multi-tenant building). Personal networks are a logical abstraction, and therefore, a user can simply define her network on top of a shared infrastructure and access any AP in a building, instead of each user accessing only her own physical AP. Users get better coverage, and they can always connect to the closest AP, avoiding weak links that bring down performance for everyone. Figure 5.5 shows how BeHop (with high density) outperforms the Stanford legacy network (a significantly sparser deployment). For instance, the median PDT at the legacy network is 8.9ms, four times larger than the one from our BeHop experience. The main reason for this is the density of the physical deployment. Despite 5GHz and band steering support from the legacy network, only 30% of the overall traffic goes over it (341GB out of 1.1TB over a one-week period). Even when comparing with BeHop at 2.4GHz, the legacy network observes higher latency. Unlike

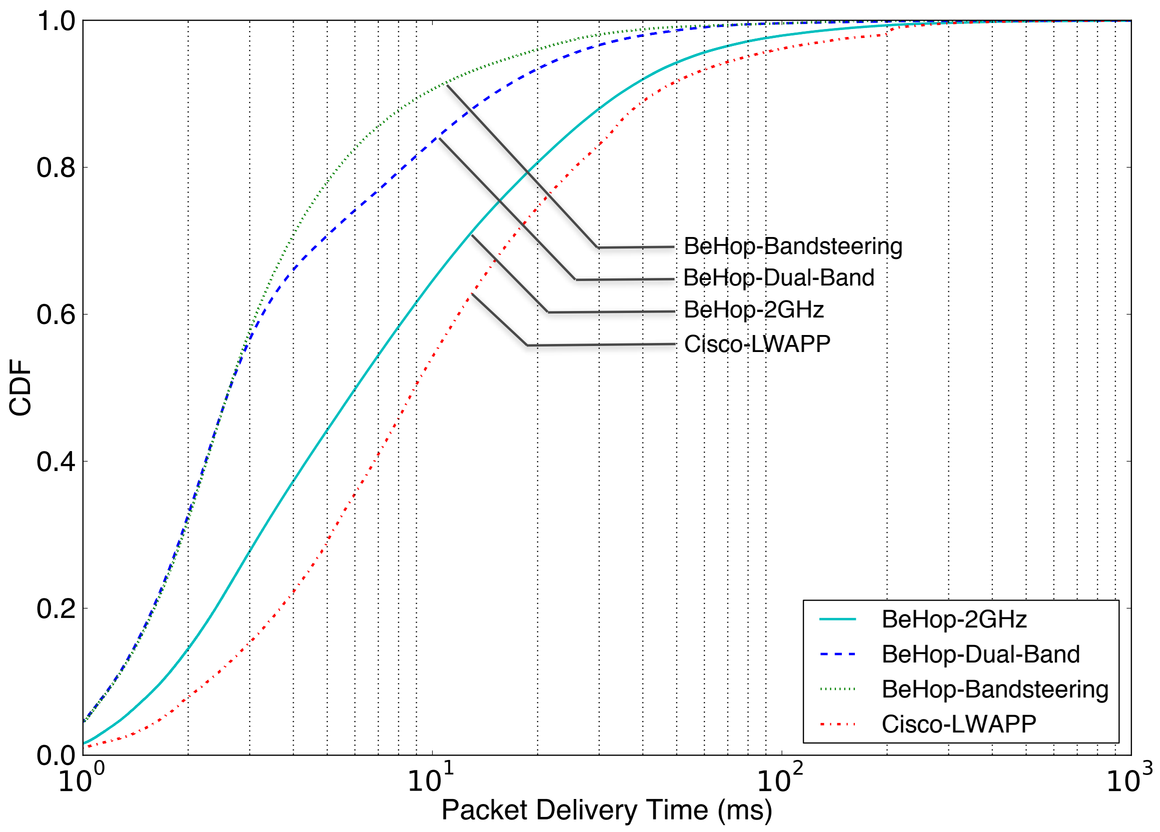
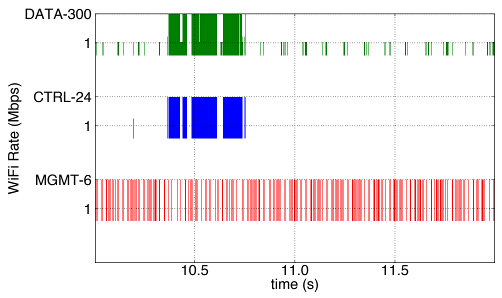


Figure 5.5: PDT for different strategies. Use of 5GHz results to faster delivery of packets.

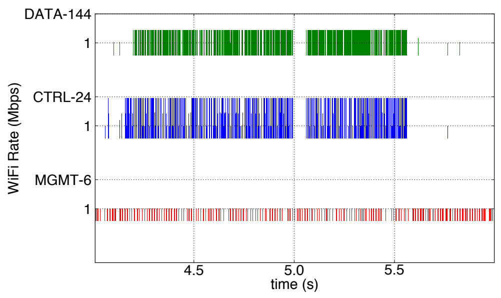
BeHop users who have APs inside their rooms (39 in total), legacy users share 25 APs spread across the entire building. This results in poor coverage for the low-penetration 5GHz band and often weak links and low WiFi rates as clients associate with APs that are far away.

2.4GHz vs. 5GHz: The use of 5GHz improves performance, reducing the median PDT from 6.2ms (BeHop-2GHz) to 2.3ms (BeHop-Bandsteering). This is not surprising. The 5GHz band has less interference and congestion, as the signal does not propagate well through walls. The lack of interference leads to more efficient use of the channel (smaller backoff, less retransmissions, higher rates) and is also enhanced by the wider 40MHz channels. To get a better understanding of why 5GHz performs better, I looked at packet-level events over a two-second period while streaming a Netflix video over 2.4GHz and 5GHz, respectively. Both streams reached the maximum data rate (3Mbps), but the impact on the channel was dramatically different between the two bands. Fig. 5.6 shows a timeline of what happens in the air. Each rectangle denotes the transmission of a packet, its width being the airtime consumed to send the packet. The use of higher rates (up to 300Mbps for data, 6Mbps for management packets) makes the 5GHz band much more efficient. Even though the maximum available rate on the 2GHz band is 144Mbps, most data packets are sent at lower rates

(78Mbps, 52Mbps), probably due to bad channel conditions resulting from load (e.g., collisions).



(a) Two-seconds zoom-in on the 5GHz band while streaming a movie from Netflix



(b) Two-seconds zoom-in on the 2.4GHz band while streaming a movie from Netflix

Figure 5.6: 2.4GHz vs. 5GHz: Higher rates, less congestion, and lower interference make the 5GHz band more efficient than the 2.4GHz. While serving the same amount of traffic, 2.4GHz approaches maximum utilization, while 5GHz remains mostly free.

Association control matters: Comparing dual-band operation with band steering, we see the importance of association control on the network. When clients are free to choose (BeHop-Dual-Band), some of them choose the sub-optimal 2.4GHz band, leading to worse performance, especially on the tail (the 90th percentile is 15ms compared to 9.4ms, when capable clients are forced to use the 5GHz band). This is an example of a client making the wrong choice in going to the more congested channel, and it highlights the effectiveness of a simple association control policy for band steering. We have seen other examples of bad decision making by clients, with the most characteristic being a client who continuously switched APs every few seconds. I defer further discussion on association patterns to future work.

5.7 Limitations of Personal WiFi Networks

Personal WiFi networks are just one step toward a fully personalized network experience. Both the architecture and prototype presented can be further improved to account for different concerns.

First, they are WiFi-specific, while in practice, we connect through different access networks (including wired and LTE). In this thesis, I tried to greedily remove all diversity each network brings and provide a baseline for how such a personalized experience should look like. One could argue instead for a narrow-waist interface for personal networks to which every technology would adjust (e.g., ignore the personalization of WiFi discovery because it is irrelevant for wired or cellular networks, and assume L3 connectivity), or completely bypass the network and bridge devices at a cloud nexus directly. I believe both approaches are valid.

The existing prototype focuses on functionality and makes the assumption that both state and traffic related to a personal network go through the cloud. This incurs overhead along with latency and bandwidth penalties, which often are not required. In reality, state distribution and dataplane decomposition will depend on trust, throughput and latency requirements, scalability, and capability to modify clients and infrastructure. We can route traffic between two devices of the same user locally if they are connected to the same trusted AP (e.g., the user's home AP), avoiding the trip to the cloud. Similarly, we can choose to decrypt and route traffic to the Internet directly for better performance. Cookie-user mapping can be mapped to the APs to avoid low-latency requirements for network discovery.

My design assumes a single entity (e.g., an ISP or a building owner) with programmatic control across all of the infrastructures. An alternative way would be to provide a directory (or multiple directories) of personal networks with a well-defined interface (e.g., probe cookie $\rightarrow$ SSID, tunnel termination mapping) and let network operators and/or AP vendors set up the rest. Building a directory of personal networks and devices associated with them can be a challenge.

Adding network cookies to probe requests is challenging in terms of deployment. WiFi connection managers in modern operating systems are proprietary. While cookies do not require any changes in the existing protocol (they are carried as standard information elements), vendors might require standardization before they support such a feature. MAC-based WiFi personalization removes such dependencies but raises privacy concerns and does not work with randomized MAC addresses.

5.8 Related Work

Personal WiFi networks build on several research and commercial ideas on multitenant datacenters and WiFi virtualization.

NXP [34] is a network virtualization platform for multitenant datacenters offering each tenant/user a private logical network. Similar to tenants, personal networks give each user her own private network. While I borrow several ideas from NXP (such as a centralized, logical bridge per user), a tenant network has significant differences from a personal WiFi network: (Virtual) devices in multitenant data centers are well defined and identified in advance, they are wired, and they belong to single owners (and therefore do not require sharing). In contrast, in a personal network, devices can dynamically connect to any AP without prior knowledge, we lack the means to identify a device during the discovery phase, and sharing is a common usecase.

Many applications use a cloud component to enable communication among different devices behind firewalls and NAT, while Google-Cloud-Messaging [29] is a generic messaging service for Android devices. Instead, I suggest enabling reachability at the network level so that applications can operate as if they are on the same network. Unmanaged Internet Protocol (UIP) [25] proposes a scalable, identity-based routing protocol (using cryptographic identities and DHT-based overlay routing) to provide global connectivity between nodes. My work goes beyond routing to L2 discovery and authentication; it leverages the cloud infrastructure to bridge devices on a per-user nexus instead of creating direct paths between hosts, and it is transparent to users who use the standard TCP/IP stack.

Network virtualization has also been extended to WiFi, where Virtual APs (VAPs) can be created on a per-client or per-user basis. [43, 46] use WiFi virtualization and focus more on optimizing WiFi performance (by centralizing association control, handoffs) instead of the user view of the network. OpenSDNW [41] uses a combination of VAP and virtual middleboxes to add user-specific network functionality (such as firewalls). Anyfi.net [27] is the closest system to personal networks: It uses unicast discovery, and all traffic from a user (including WiFi management traffic) is routed to a users's AP, therefore offering her own SSID, password, and home network view. All systems above use the MAC address as the main user identifier, which has certain security and privacy limitations. Instead, network cookies provide a simpler, more secure and revocable way of managing users' identities. I also extend personal networks to account for sharing WiFi APs and devices with guests.

Many systems have been developed to eliminate the overhead associated with authenticating to WiFi, by building large, federated networks from individual APs. FON is a community where users share their bandwidth through a second SSID and in return can access a worldwide collection of FON APs. Several ISPs maintain networks in which home APs operate as hotspots for users. In both cases, users authenticate through a captive portal. Similarly, HotSpot 2.0 is a set of standards to enable automatic discovery and authentication for WiFi services. These systems build a single, huge network which is the same for everyone, and is limited to Internet connectivity. Users can not set fine-grained ACLs, or grant access to their local devices. Personal networks build (potentially) millions of personalized networks, eliminating the need for extra accounts and passwords, and creating a personalized WiFi footprint for every user.

Chapter 6

Conclusions

6.1 Summary

The ideal network is expected to be transparent to end users and to fulfill a single, unambiguous, and objective task: Carry packets from one side to the other as quickly as possible. In practice, networks are more complicated: They decide which packets get priority over others, which traffic gets exempted from data caps, and how users authenticate and access available network resources.

As users, we have no say or visibility on how these decisions are made—they are tailored to accommodate the needs of network administrators or general user groups. This dissertation made the following statement:

Users—not ISPs or content providers—should decide how their traffic is treated by the network according to their own unique preferences, and network cookies provide the means to do this in a practical, secure, scalable, revocable, and auditable way.

Chapter 2 provided a high-level architecture for personalized network services and discussed the division of labor between various components and emphasizing the importance of the user→interface to express user preferences.

Chapter 3 focused on how to express user preferences to the network. It listed three high-level requirements for such a mechanism (*be simple yet expressive, respect the tussle, be practically deployable*) and demonstrated that existing mechanisms fall short to meet them. It then introduced network cookies, a simple abstraction to express preferences to the network in a practical, secure, revocable, scalable, auditable, and policy-free way. The next two chapters demonstrated how the combination of network cookies and user preferences enable personalized network services that can better serve the needs of individual users.

Chapter 4 studied user preferences and network cookies in the context of net neutrality. It showed that, in the absence of user choice and an appropriate mechanism for expressing which traffic gets preferential treatment from the network, existing network services are tailored to benefit only very popular content, potentially hurting innovation and raising controversy among users, ISPs, content providers, and regulators. I performed two user studies on zero-rating and fast-lane services, demonstrating the diverse and long-tailed nature of user preferences and that users are willing to express them if it is easy to do so. In contrast, any one-size-fits-all approach is unlikely to meet user needs.

Chapter 5 presented a more holistic view of user-centric networks through personal WiFi networks. With personal WiFi networks, users define their network experience once, and then their networks follow them wherever they go. Users always connect with their own SSIDs and passwords, their devices are always reachable, and they can share resources with the ones they trust. While different in nature from fast lanes and zero-rating services, personal WiFi networks pose similar requirements: scalability to serve thousands or millions of users from the same infrastructure; respect for user privacy to prevent third parties from tracking individuals; and trust revocability to let users change their minds if they want to do so. I demonstrated the feasibility, benefits, and challenges of personal WiFi networks through a prototype deployment in a student dorm at Stanford University.

6.2 Closing Thoughts

By 2025, Internet traffic will have dramatically increased; we will be always connected through multiple devices from multiple networks, ISPs will continue to invest to improve their infrastructure and cope with demand, and networks will be easy to program and customize.

We live in a time of enormous focus on how to realize this vision while ensuring that 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. The 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 in this debate, aside from lobbying our political representatives.

I expect that giving users control will eventually emerge as the only plausible way to meet our future needs, keep the Internet a level playing field, and make networks more amicable to users.

This thesis tries to make two contributions towards this direction. First, network cookies provide a neutral, policy-free mechanism for expressing 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.

Second, I advocate for a user-centric policy, where users can directly state their preferences and look after themselves. The 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 the rest.

...

Bibliography

[1] Barack Obama statement on Net Neutrality. `https://www.whitehouse.gov/blog/2014/11/10/president-obama-urges-fcc-implement-stronger-net-neutrality-rules`.

[2] Data plane development kit, http://www.dpdk.org.

[3] FCC Open Internet Rules and Order, February 2015.

[4] How does zero rating affect mobile data usage.

[5] https://support.t-mobile.com/thread/124533?tstart=0.

[6] Last Week Tonight on Net Neutrality. `https://www.youtube.com/watch?v=fpbOEoRrHyU`.

[7] Millions of people have no idea they are using the Internet.

[8] Private discussion with a major ISP provider, November 2015.

[9] Private discussion with engineers from a content provider participating in MusicFreedom, January 2015.

[10] Prohibition of Discriminatory Tariffs for Data Services Regulations, TRAI, February 2016.

[11] Sanvine Report on encrypted traffic.

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

[13] This is what the EU thinks is net neutrality.

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

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

[16] Pat Bosshart, Dan Daly, Glen Gibb, Martin Izzard, Nick McKeown, Jennifer Rexford, Cole Schlesinger, Dan Talayco, Amin Vahdat, George Varghese, et al. P4: Programming protocol-independent packet processors. *ACM SIGCOMM Computer Communication Review*, 44(3):87–95, 2014.

[17] Marshini Chetty, Srikanth Sundaresan, Sachit Muckaden, Nick Feamster, and Enrico Calandro. Measuring broadband performance in south africa. In *Proceedings of the 4th Annual Symposium on Computing for Development*, page 1. ACM, 2013.

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

[19] Luca Deri, Mario Martinelli, Tomasz Bujlow, and Alfredo Cardigliano. ndpi: Open-source high-speed deep packet inspection. In *Wireless Communications and Mobile Computing Conference (IWCMC), 2014 International*, pages 617–622. IEEE, 2014.

[20] Antonios Drossos. Guest blog: the real threat to the open internet is zero-rated content. *World Wide Web Foundation*, 2015.

[21] Paul Emmerich, Florian Wohlfart, Daniel Raumer, and Georg Carle. Moongen: A scriptable high-speed packet generator. *arXiv preprint arXiv:1410.3322*, 2014.

[22] Seyed Kaveh Fayazbakhsh, Vyas Sekar, Minlan Yu, and Jeffrey 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.

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

[24] Andrew D Ferguson, Arjun Guha, Chen Liang, Rodrigo Fonseca, and Shriram Krishnamurthi. Participatory networking: An api for application control of sdns. In *ACM SIGCOMM Computer Communication Review*, volume 43, pages 327–338. ACM, 2013.

[25] Bryan Ford. Unmanaged internet protocol: taming the edge network management crisis. *ACM SIGCOMM Computer Communication Review*, 34(1):93–98, 2004.

[26] James L Gattuso. Net neutrality: Reining in innovation. 2015.

[27] http://anyfi.net. Anyfi.net.

[28] http://euclidanalytics.com/. Euclid Analytics.

[29] https://developers.google.com/cloud messaging/. Google Cloud Messaging.

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

[31] http://www.economist.com/news/business/21654602-wi-fi-first-technology-will-be-great-consumers-disruptive-mobile-firms change. The Economist.

[32] http://www.networkworld.com/article/2361846. Networkworld article on iOS MAC randomization.

[33] Eddie Kohler, Robert Morris, Benjie Chen, John Jannotti, and M Frans Kaashoek. The click modular router. *ACM Transactions on Computer Systems (TOCS)*, 18(3):263–297, 2000.

[34] Teemu Koponen, Keith Amidon, Peter Balland, Martín Casado, Anupam Chanda, Bryan Fulton, Igor Ganichev, Jesse Gross, Natasha Gude, Paul Ingram, et al. Network virtualization in multi-tenant datacenters. In *USENIX NSDI*, 2014.

[35] Marc Zuckeberg. Free Basics protects net neutrality. `http://blogs.timesofindia.indiatimes.com/toi-edit-page/free-basics-protects-net-neutrality/`.

[36] J Mccauley. Pox: A python-based openflow controller.

[37] David Naylor, Kyle Schomp, Matteo Varvello, Ilias Leontiadis, Jeremy Blackburn, Diego R López, Konstantina Papagiannaki, Pablo Rodriguez Rodriguez, and Peter 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.

[38] Vladimir Andrei Olteanu, Felipe Huici, and Costin 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.

[39] Ben Pfaff, Justin Pettit, Keith Amidon, Martin Casado, Teemu Koponen, and Scott Shenker. Extending networking into the virtualization layer. In *Hotnets*, 2009.

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

[41] Julius Schulz-Zander, Carlos Mayer, Bogdan Ciobotaru, Stefan Schmid, and Anja 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.

[42] Justine Sherry, Chang Lan, Raluca Ada Popa, and Sylvia Ratnasamy. Blindbox: Deep packet inspection over encrypted traffic. *SIGCOMM Comput. Commun. Rev.*, 45(5):213–226, August 2015.

[43] Lalith Suresh, Julius Schulz-Zander, Ruben Merz, Anja Feldmann, and Teresa Vazao. Towards Programmable Enterprise WLANS with Odin. HotSDN '12.

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

[45] Barbara van Schewick. T-mobiles binge on violates key net neutrality principles. 2016.

[46] Yiannis Yiakoumis, Manu Bansal, Adam Covington, Johan van Reijendam, Sachin Katti, and Nick McKeown. Behop: a testbed for dense wifi networks. In *Proceedings of the 9th ACM international workshop on Wireless network testbeds, experimental evaluation and characterization*, pages 1–8. ACM, 2014.

[47] Yiannis Yiakoumis, Sachin Katti, Te-Yuan Huang, Nick McKeown, Kok-Kiong Yap, and Ramesh Johari. Putting home users in charge of their network. In *Proceedings of the 2012 ACM Conference on Ubiquitous Computing*, pages 1114–1119. ACM, 2012.