Virtualizing the Access Network via Open APIs

Vijay Sivaraman, Tim Moors,
Hassan Habibi Gharakheili, Dennis Ong
University of New South Wales
Sydney, Australia
{vijay, t.moors}@unsw.edu.au
{h.habibi, dennis.ong}@unsw.edu.au

John Matthews, Craig Russell
CSIRO ICT Centre
Sydney, Australia
john.matthews@csiro.au
craig.russell@csiro.au

ABSTRACT

Residential broadband consumption is growing rapidly, increasing the gap between ISP costs and revenues. Meanwhile, proliferation of Internet-enabled devices is congesting access networks, frustrating end-users and content providers. We propose that ISPs virtualize access infrastructure, using open APIs supported through SDN, to enable dynamic and controlled sharing amongst user streams. Content providers can programmatically provision capacity to user devices to ensure quality of experience, users can match the degree of virtualization to their usage pattern, and ISPs can realize per-stream revenues by slicing their network resources. Using video streaming and bulk transfers as examples, we develop an architecture that specifies the interfaces between the ISP, content provider, and user. We propose an algorithm for optimally allocating network resources, leveraging bulk transfer time elasticity and access path space diversity. Simulations using real traces show that virtualization can reduce video degradation by over 50%, for little extra bulk transfer delay. Lastly, we prototype our system and validate it in a test-bed with real video streaming and file transfers. Our proposal is a first step towards the long-term goal of realizing open and agile access network service quality management that is acceptable to users, ISPs and content providers alike.

Categories and Subject Descriptors

C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design

Keywords

Software-Defined Networks; Slicing; Virtualization

1. INTRODUCTION

Fixed-line Internet Service Providers (ISPs) are increasingly confronting a business problem – residential data consumption continues to grow at 40% per annum [3], increasing the cost of the infrastructure to transport the growing traffic volume. However, revenues are growing at less than 4% per annum, attributable mainly to "flat-rate" pricing [3]. To narrow this widening gap between cost and revenue, ISPs have attempted throttling selected services (such as peer-to-peer), which sparked public outcry (resulting in "net neutrality" legislation), and now routinely impose usage quotas, which can stifle delivery of innovative content and services. It is increasingly being recognised that ensuring sustainable growth of the Internet ecosystem requires a rethink of the business model, that allows ISPs to exploit the *service quality* dimension (in addition to bandwidth and download quota) to differentiate their offerings and tap into new revenue opportunities [28, 20].

Simultaneously, end-user expectations on service quality are evolving as personal and household devices proliferate and traffic types change. Real-time and streaming entertainment content (e.g. Netflix and YouTube) has replaced peer-to-peer as the dominant contributor to Internet traffic [24]. However, maintaining quality of experience (QoE) in online video viewing over best-effort networks remains a challenge. The rapid growth in the number of household devices (computers, phones, tablets, TVs, smart meters, etc.) concurrently accessing the Internet has increased peak-load and congestion on the access link, which is often the bottleneck between the (wired or wireless) residential LAN and the ISP backbone network [26]. The consequent impact on video quality (startup delays and rebuffering events) has been shown to lead to higher user abandonment, lower user engagement, and lower repeat viewership [12].

Content providers (CPs), who monetize their video offerings via ad-based or subscription-based models, are seeing a direct impact on their revenue from reduced user QoE. Though they use sophisticated techniques such as playback buffering, content caching, adaptive coding, and TCP instrumentation to improve video quality, these approaches are inherently limited and often involve trade-offs (e.g. increasing playback buffers can reduce rebuffering but increase startup delay). The frustrations associated with providing good QoE to users over a third-party access network may explain why some CPs (e.g. Google) are building their own fiberhoods, while some other CPs are merging with access network operators (e.g. NBC and Comcast). However, we believe that these proprietary solutions cannot be replicated world-wide (for cost and regulatory reasons), and open solutions are needed that allow any CP to improve the delivery of their services over any ISP access network.

CoNEXT'13, Dec 9–12, 2013, Santa Barbara, California.

http://dx.doi.org/10.1145/2535372.2535381.

Given the strong motivation for all parties (ISPs, users, and CPs) to want service quality capability in the network, one can rightly ask why it does not already exist. Indeed, user QoS/QoE has been studied extensively over the past two decades, and many researchers (including the authors) have worked to develop numerous technical solutions ranging from ATM-SVC to RSVP and IntServ/DiffServ. However, we believe that the limited success of these prior frameworks is partly because they have not satisfactorily addressed two critical aspects: (a) who exercises control over the service quality? and (b) how is it monetized? These challenges are elaborated next.

Control: Today, the control of network service quality is largely left to the ISP, who carefully hand-crafts policy and device configurations, likely via mechanisms (e.g. marking, policing, resource reservation, and queueing) from the DiffServ frameworks. Users have no visibility into the ISP's doings, and are left powerless and suspicious, wondering if "neutrality" is being violated (e.g. peer-to-peer traffic being de-prioritized). Further, exposing controls to the user also raises challenges around user expertise needed to configure and manage QoS. At the other end, CPs can exert little (if any) control over service quality in ISP networks today. They do not have access to end-to-end quality assurance frameworks (e.g. RSVP/IntServ based) since ISPs deem them either too onerous to operate or too dangerous to expose; at best CPs can indicate relative priority levels for their packets (e.g. via DiffServ code-points), but these assurances are "soft", being qualitative and subject to other traffic in the network. These concerns exacerbate further when the ISP and CP do not peer directly, i.e. connect via a transit provider. Any viable quality enhancement solution therefore has to tackle the issue of how the control is shared amongst the various players involved.

Monetization: An ISP has little incentive to deploy service quality mechanisms unless there is a monetary return. Consumers are very price sensitive, and it is unclear if sufficient consumers will pay enough for the QoS enhancement to allow the ISP to recoup costs. CPs potentially have greater ability to pay; however, current "paid peering" arrangements are based on aggregate metrics such as transfer volume or transfer rate. A CP is unlikely to pay more for "wholesale" improvement in service quality, especially if a non-negligible fraction of their traffic gets delivered at adequate quality anyway. A viable QoS solution should therefore allow the CP to make fine-grained (e.g. per-flow) decisions in an agile way so that service quality can be aligned with their business models. For example, the CP may want to deliver traffic only for certain customers or certain content at high quality, and these decisions can vary dynamically (e.g. depending on time-of-day or loss/delay performance of the network).

The above two challenges have been poorly addressed in earlier frameworks, dissuading ISPs from deploying service quality mechanisms and causing frustration for CPs and end-users. We believe that the emerging paradigm of **software defined networking** (SDN) provides us a new opportunity to overcome this old impasse. Logical centralization of the control plane under SDN helps in many ways:

1. A central "brain" for the network makes it easier for the ISP to expose (e.g. via APIs) service quality controls needed by an external party, such as the CP. We believe that a software-driven API is a far superior method for information exchange rather than interconnecting existing protocols (e.g. RSVP) to external parties, since (a) protocols often reveal information (e.g. network topology or network state) that is both private to the ISP and unnecessary for the external entity, whereas APIs can be crafted specifically for the negotiation task at hand, (b) protocols do not easily straddle transit domains, whereas APIs can be invoked by a remote entity that does not peer directly with the ISP, and (c) protocols are typically distributed across network elements and take longer to converge whereas APIs implemented at the central controller can respond rapidly to external requests. We believe that the above advantages of APIs make SDN a more suitable paradigm by which the ISP can expose and share QoS control with external entities.
2. The centralized brain in SDN is more amenable for optimal decision making. Since the SDN controller has a global view of resources, it can make informed decisions based on current availability and requests. Indeed, the resource management algorithm we develop in this paper has a provable performance bound, which is difficult to achieve in distributed systems that have limited visibility into global state.
3. Lastly, SDN provides a cross-vendor solution that does not require protocol support from the various forwarding elements. The resource partitioning can be executed by the centralised software across any forwarding element over any access technology that supports a standardized SDN interface such as OpenFlow.

At a high level, our solution encourages the ISP to "virtualize" the access network, namely partition the last-mile network bandwidth resources dynamically amongst flows, using SDN. The virtualization is driven by open APIs that are exposed to external entities (CPs in our case), who can choose to invoke it to negotiate service quality with the network on a per-flow basis. For the ISP, the API offers a monetization opportunity, while protecting sensitive internal information, and giving them the freedom to innovate mechanisms for maximizing resource usage (for example by "pooling" WiFi resources in a neighborhood). For CPs, the API provides an enforceable assurance from the access network, and the pay-as-you-go model gives them freedom to align quality requirements with their business models. For users, we equip them with a simple control into the degree to which their access network resources are pooled/partitioned, allowing them to match it to their usage patterns. While past experience has taught us that any large-scale deployment of QoS faces significant practical obstacles, we believe our solution approach has the potential to overcome the business, regulatory and administrative impediments, and offers the right set of incentives for ISPs, CPs and users to collaborate for its success.

Our specific contributions are as follows. We use video streaming and file transfers as two motivating examples, and develop a system architecture that virtualizes the last-mile access infrastructure to allow controlled and efficient sharing amongst user application streams. CPs can programmatically provision capacity on the access links to end-user devices at short time-scales to ensure consistent quality of experience for users; ISPs can realize per-stream revenues by dynamically reconfiguring their access network in an agile way; and users can choose their degree of participation

by tuning a single parameter. We develop an algorithm for optimal resource allocation in the access network, leveraging scheduling in the time dimension (that allows non-time-critical elastic traffic such as bulk transfers to be deferred in favour of time-sensitive streaming traffic) as well as in the space dimension (by pooling bandwidth from multiple wireless access points in a neighborhood). We then evaluate the efficacy of our algorithm in improving service quality via simulations of real traces of over 10 million flows taken from a large enterprise network. Lastly, we prototype our system using commercial switches and commodity access points, and demonstrate the benefits of our scheme via experiments in a test-bed emulating three residences running real applications. Our work presents a first step towards a viable and pragmatic approach to delivering service quality in access networks in a way that is beneficial to ISPs, users, and CPs alike.

The rest of the paper is organized as follows: §2 describes the use-cases considered in this paper. §3 describes our system architecture, trade-offs, and algorithm. In §4 we evaluate our system via simulation with real traffic traces, while §5 describes the prototype development and experimentation. Relevant prior work is summarized in §6, and the paper concludes in §7.

2. USE-CASES AND OPPORTUNITIES

The set of applications that can benefit from explicit network support for enhanced service quality is large and diverse: real-time and streaming videos can benefit from bandwidth assurance, gaming applications from low latencies, voice applications from low loss, and so on. In this paper we start with two application use-cases: *real-time/streaming video*, chosen due to its growing popularity with users and monetization potential for providers, and (non-real-time) *bulk transfers*, chosen for their large volume and high value to users. The APIs we develop and demonstrate for these use-cases will help illustrate the value of our approach, and can be extended in future work for other application types.

2.1 Real-Time / Streaming Video

Online video content, driven by providers such as Netflix, YouTube, and Hulu, is already a dominant fraction of Internet traffic today, and expected to rise steeply in coming years. As video distribution over the Internet goes mainstream, user expectations of quality have dramatically increased. Content providers employ many techniques to enhance user quality of experience, such as CDN selection [15], client-side playback buffering [22], server-side bit-rate adaptation [2], and TCP instrumentation [6]. However, large-scale studies [5, 12] have confirmed that video delivery quality is still lacking, with startup delays reducing customer retention and video "freeze" reducing viewing times. Since variability in client-side bandwidth is one of the dominant contributors to quality degradation, an ideal solution is to "slice" the network to explicitly assure bandwidth to the video stream. Eliminating network unpredictability will (a) reduce playback buffering and startup delays for streaming video, (b) benefit live/interactive video streams that are latency bound and cannot use playback buffering, and (c) minimise the need for sophisticated techniques such as bandwidth estimation and rate adaptation used by real-time and streaming video providers.

There are however important questions to be addressed in realizing the above slicing solution: (a) what interaction is needed between the application and the network to trigger the bandwidth reservation? (b) is the bandwidth assured end-to-end or only on a subset of the path? (c) which entity chooses the level of quality for the video stream, and who pays for it? (d) what rate is allocated to the video stream and is it constant? (e) what is the duration of the reservation and how is abandonment dealt with? and (f) how agile is the reservation and can it be done without increasing start-up delays for the user? Our architecture presented in §3 will address these non-trivial issues.

2.2 Bulk Transfer

After video, large file transfers are the next biggest contributors to network traffic. Examples include peer-to-peer file-sharing, video downloads (for offline viewing), software updates, and cloud-based file storage systems [24]. Unlike video, bulk transfers do not need a specific bandwidth, and user happiness generally depends on the transfer being completed within a "reasonable" amount of time. This "elasticity" creates an opportunity for the ISP to dynamically size the network bandwidth "slice" made available to bulk transfers, based on other traffic in the network. This can allow the ISP to reduce network peak load, which is a dominant driver of capital expenditure, and release capacity to admit more lucrative traffic streams (e.g. real-time/streaming video) requiring bandwidth assurances.

Though the idea of "shifting" bulk transfer traffic to low-load periods based on their elasticity is conceptually simple, there are challenges around (a) how to identify bulk transfer parameters such as size and elasticity? (b) how to incentivize the user/provider to permit such shifting? and (c) how to dimension the network resource slice for this elastic traffic? These are addressed in §3.

2.3 WiFi Pooling

Another opportunity that can be leveraged, particularly for bulk transfers, is that in urban areas the density of wireless access points (APs) is high – often 6 or more WiFi networks are visible at a typical location [10]. An ISP with a sufficiently high density of penetration in a neighborhood may therefore have available multiple paths, each via a different AP, by which data can be sent to a specific household device. As an example, the user's iPad may be streaming video via her home AP, while simultaneously her smart-TV or media gateway, which may be downloading large content for later viewing, could be made to do so via a neighbor's lightly loaded AP. This "off-loading" of traffic to alternate paths leads to more efficient use of access link resources, and motivates us to propose that the entire access network infrastructure be virtualized by treating it as a pool of resources, rather than limiting a household to one access link.

The ability to pool WiFi resources has many challenges, including determination of signal strengths and data rates, management of migrations of user clients across APs, and considerations around security, quota, charging, and fairness. However, several operators world-wide, such as Oi in Brazil, Ziggo B.V. in the Netherlands, Telefonica in Spain, and Comcast in the US, are surmounting these challenges as they prepare to convert their users' homes into hotspots [14] to pool their WiFi resources. We will leverage these

methods, additionally using SDN to steer traffic along the desired path in the pooled WiFi network.

3. SYSTEM ARCHITECTURE AND ALGORITHM

Motivated by the above use-cases, we now propose a system architecture for virtualizing the access network. We first outline the major architectural choices and trade-offs (§3.1), then describe the operational scenario (§3.2), and finally develop the detailed mechanisms and algorithms for virtualization (§3.3).

3.1 Architectural Choices and Trade-Offs

The aim of virtualization is to partition resources dynamically amongst flows in a programmatic way, so that the network is used as efficiently as possible for enhancing application performance or reducing cost. We briefly discuss why open APIs are needed to achieve the virtualization, what part of the network is virtualized, and who exercises control over the virtualization.

Why Open APIs: Current mechanisms used by ISPs to partition network resources require cripplingly expensive tools for classifying traffic flows (e.g. using DPI), encourage applications to obfuscate or encrypt their communications, and risk causing public backlash and regulation. Therefore, we advocate that the virtualization be driven externally via an explicit API open to all CPs. This allows CPs to choose a resource requirement commensurate with the value of the service, while letting ISPs explicitly obtain service attributes without using DPI.

What is Virtualized: Assuring application performance ideally requires end-to-end network resource allocation. However, past experience with end-to-end QoS frameworks has taught us that getting the consensus needed to federate across many network domains is very challenging. In this paper we therefore focus on the achievable objective of partitioning resources within a single domain. A natural choice is the last-mile access network as there is evidence [26, 9] that bottlenecks often lie here and not at the interconnects between networks. Our solution can in principle be adapted to any access technology, be it dedicated point-to-point (DSL, PON) or shared (e.g. cable, 3G). In this paper we focus our theoretical formulation and evaluation on point-to-point wired access technologies, wherein each subscriber has a dedicated bandwidth, and overlay it with WiFi pooling to allow sharing of bandwidth across residences. The case of shared media (cable or 3G) deserves a separate discussion around the policies needed to be fair to different users who embrace the virtualization scheme to different extents, and is left for future work.

Who Controls the Virtualization: Though the virtulization APIs can be invoked by any entity, we envisage initial uptake coming from CPs rather than consumers, since: (a) uptake is needed by fewer, since as much of 60% of Internet traffic comes from 5 large content aggregators [9], (b) CPs have much higher technical expertise to upgrade their servers to use the APIs, and (c) client-side charging for API usage can significantly add to billing complexity. For these reasons, we expect CPs to be the early adopters of the virtualization APIs, and defer consumer-side uptake to future study.

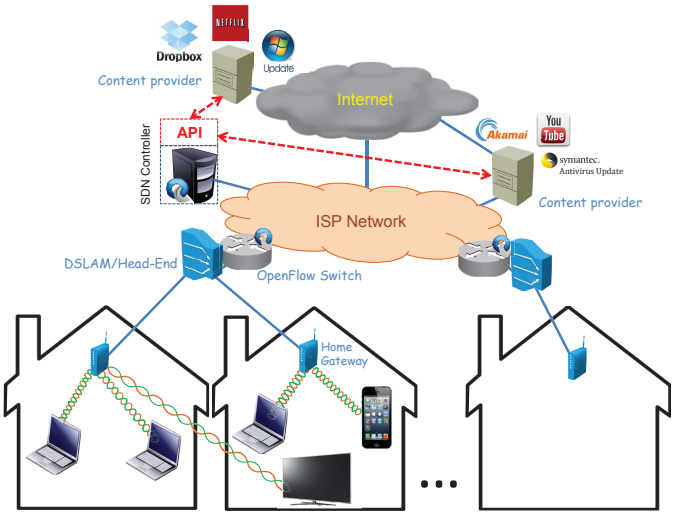

Figure 1: Network topology

The end-user still needs to be empowered with a means to control the virtualization, e.g. a user might not want her web-browsing or work-related application performance to be overly affected by streaming video that her kids watch. We therefore propose that each household be equipped with a single parameter $\alpha \in [0, 1]$ which is the fraction of its access link capacity that the ISP is permitted to virtualize. Setting $\alpha = 0$ disables virtualization, and the household continues to receive today's best-effort service. Households that value video quality could choose a higher α setting, while households wanting to protect unpaid traffic (web-browsing or peer-to-peer) can choose a lower α. Higher α can potentially reduce the household Internet bill since it gives the ISP more opportunity to monetize from CPs. Our work will limit itself to studying the impact of α on service quality for various traffic types; determining the best setting for a household will depend on its Internet usage pattern and the relative value it places on the streams, which is beyond the scope of this study.

3.2 Operational Scenario

We briefly describe the operational scenario, the reference topology, the flow of events, the API specifications, and the roles of the CP and the user.

3.2.1 Topology and Flow of Events

Fig. 1 shows a typical access network topology. Each residence has a wireless home gateway to which household devices connect. The home gateway offers Internet connectivity via a broadband link (e.g. DSL or PON), connecting to a line termination device at the ISP local exchange, which is in turn back-ended by an Ethernet switch that has SDN capability. The Ethernet switches at each local exchange connect via metro- or wide-area links to the ISP's backhaul network. The ISP network houses an SDN controller that exposes the APIs discussed below, and executes the virtualization algorithm (described at the end of this section) to reconfigure the network. The ISP network can either peer directly, or via other ISPs, to content providers that source the data that is consumed by users. Our solution works equally well when the data is sourced from CDNs or content caches within or outside the ISP network.

The operational flow of events is as follows. The user's request for content (e.g. YouTube video link click or Dropbox file transfer command) goes to the CP, who can instantly call the API into the ISP network to associate resources for this flow. If the negotiation succeeds, the ISP assures those resources for the flow, and charges the CP for it. In what follows we describe the APIs in more detail and elaborate on the specific actions required by the CP and the user.

3.2.2 The APIs

We now develop minimalist specifications of the APIs for the two use-cases considered in this paper; detailed specifications are left for future standardization.

API for Bandwidth Assurance: This specifies: (a) *Caller id:* The identity of the entity requesting the service. Authentication of some form (such as digital signature of the message) is assumed to be included, but we do not discuss security explicitly in this work. (b) *Call Type:* A type field indicates the service being requested, in this case minimum bandwidth assurance. (c) *Flow tuple:* The 5-tuple comprising the IP source and destination addresses, the transport protocol, and the source and destination port numbers, that identify the flow (consistent with the OpenFlow specification). Note that wildcards can be used to denote flow aggregates. (d) *Bandwidth:* The bandwidth (in Mbps) that is requested by the flow. (e) *Duration:* The duration (in seconds) for which the bandwidth is requested.

This API assures minimum bandwidth to a service like video streaming. Note that the flow can avail of extra bandwidth if available, and is not throttled or rate-limited by the network. Further, we have intentionally kept it simple by using a single bandwidth number, rather than multiple (e.g. peak and average) rates. The value to use is left to the CP, who knows best their video stream characteristics (peak rate, mean rates, smoothness, etc.) and the level of quality they want to support for that particular session. The duration of the bandwidth allocation is decided by the caller. To combat abandonment, the CP may choose to reserve for short periods (say a minute) and renew the reservation periodically; however, this runs the risk of re-allocation failures. Alternatively, the caller can choose to reserve for longer periods, and the APIs can be extended to include cancellation of an existing reservation. These implementation decisions are left for future standardization. Lastly, the ISP will charge the caller for providing bandwidth assurance to the stream. The pricing model is left to the individual ISP, and can be dynamic (depending on time-of-day, congestion levels, etc.). We do not know, or dare to put, a price on bandwidth – that will ultimately be determined by market forces.

API for Bulk Transfer: This includes: (a) *Caller id:* as before. (b) *Call Type:* in this case bulk transfer. (c) *Flow tuple:* as before. (d) *Size:* The volume of data to be transferred, in MegaBytes. (e) *Deadline:* The duration (in seconds) by which the transfer is requested to be completed. This API is for large data transfers that are not time critical. The elasticity can be leveraged by the ISP to reduce peak demand [20], and the monetary benefit can be passed on to the CP in the form of a lower per-byte transfer cost. In turn, the CP may pass on the discount to the user, such as by charging them less for downloading a movie for later viewing than for streaming it in real-time. The CP can either solicit the deadline parameter directly from the user (via the application's interface), or deduce it implicitly (from prior knowledge about the user), as elaborated further below.

One conceivable way by which the ISP can schedule bulk transfers is to warehouse the data, i.e. move it from the CP to the ISP, and then on to the user at a suitably scheduled time. However, this carries with it complexities (such as proxies, split connections) and liabilities (e.g. pirated data). We instead propose that the ISP dynamically adjust the network path and bandwidth made available to the bulk transfer, without having to store any content. This eliminates complexities and liabilities for the ISP, lets them innovate (and protect) their mechanisms for path selection and bandwidth adjustment, and does not require any user-client changes.

3.2.3 Changes for Content Provider and User

The changes required at the content servers are well-within the technical expertise of the CPs. They can identify a client's ISP based on the client IP address, and a DNS entry can be created for the controller advertised by that ISP. We note that the CP has full visibility of the flow end-points (addresses and ports), irrespective of whether the home uses NAT or not. For streaming video, the bandwidth requirement can be deduced from the format and encoding of the content. For bulk transfers, delay bounds can either be explicitly solicited from the user (via an option in the application user interface) or chosen based on previously acquired knowledge about the consumer (e.g. deadlines to ensure delivery before prime time viewing). Lastly, CPs are at liberty to align the API usage with their business models, such as by invoking it only for premium customers.

Subscribers are provided with a single knob $\alpha \in [0, 1]$ that controls the fraction of their household link capacity that the ISP is permitted to virtualize, adjusted via their account management portal. This parameter can be tuned by the user to achieve the desired trade-off between quality for reserved (video/bulk) flows and unreserved (browsing/peer-to-peer) flows. Where WiFi pooling is employed, we do not mandate any changes to user clients or applications to facilitate migration of traffic across access points (such as multi-homing [7] or proxying techniques [29]), instead relying on ISP-controlled migration using MAC address black-listing. All user clients (computers, TVs, phones, etc.) running any operating system can thereafter benefit from the virtualization without requiring any software or hardware changes. For bulk transfer applications, the user interface may be updated by CPs to explicitly solicit transfer deadlines from users, potentially giving users financial incentive to choose slacker deadlines.

3.3 The Virtualization Mechanism

The mechanism we develop for the ISP to execute the above APIs leverages the time dimension (§3.3.1) and space dimension (§3.3.2), and schedules resources for traffic flows as per our algorithm (§3.3.3).

3.3.1 Time Scheduling

The time "elasticity" of bulk transfers, inferred from the *deadline* parameter in the API call, is used to dynamically adjust the bandwidth made available to such flows. Upon API invocation, the ISP creates a new flow-table entry and dedicated queue for this flow in the switches along the path (though scalability is a potential concern here, we note that

a vast majority of flows are "mice" and will not be using the API). Periodically, the minimum bandwidth assured for this queue is recomputed as the ratio of the remaining transfer volume (inferred from the total volume less the volume that has already been sent) to the remaining time (deadline less the start time of the flow). Note that the flow is not throttled, and can possibly get higher bandwidth than the minimum depending on congestion state. Also, the flow bandwidth requirement is reassessed periodically (every 10 seconds in our prototype) – this allows bandwidth to be freed up for allocation to real-time streams in case the bulk transfer has been progressing ahead of schedule, and gives the bulk transfer more bandwidth to catch-up in case it has been falling behind schedule. Lastly, the dynamic adjustment of network bandwidth "slice" for this flow is largely transparent to the client and server. Call admission for all flows ensures that no deadline violations occur; rejected flows simply traverse as best-effort with no assurances (or costs) for the caller.

3.3.2 Space Scheduling

Pooling of residential WiFi resources has been shown to be feasible from a research [10] and commercial [27, 14] standpoint. Our intention is to show that the ISP can leverage such mechanisms to improve call admission success (and consequent revenue), without needing to reveal such innovations to CPs. We propose (and prototype) a solution in which the ISP centrally manages the APs (akin to an enterprise WiFi solution), and user clients are unmodified. The residential APs are configured as transparent layer-2 devices with no routing or address translation. IP address allocation to each user device is done centrally by the ISP, as is the security authentication, akin to enterprise networks. Once authenticated, a user device can then migrate (with minimal disruption) across APs (our implementation achieves this by dynamically black-listing client MAC addresses). Direct visibility of the user devices to the ISP also simplifies accounting on a per-device basis, which eliminates issues with quota stealing across households.

Fairness in bandwidth sharing can be a concern when households have different settings of fraction α that represents their contribution to the "virtual pool" of bandwidth. The concern is that a household with low α allows little of its bandwidth to be used by its neighbors, but can benefit from using bandwidth from a neighbor with high α. To counter this unfairness, we allow a household-i access to at most fraction α_i of the bandwidth in the virtual pool from its neighborhood. More specifically, the maximum bandwidth that household-i can access from its neighbors is limited to $\alpha_i * \sum_{j \in N_i} \alpha_j C_j$, where C_j denotes the broadband capacity of household-j, and the set N_i denotes the APs that comprise the wireless neighborhood of household-i. For example, if three households with overlapping WiFi coverage, each with broadband access bandwidth of 10 Mbps, set their parameters to $\alpha_1 = 0.2$, $\alpha_2 = 0.5$ and $\alpha_3 = 1.0$, then household-1 can access at most 3 Mbps from its neighbors, household-2 at most 6 Mbps, and household-3 the entire 7 Mbps from its neighbors.

3.3.3 Virtualization Algorithm

Our algorithm takes as **inputs**: (a) The bandwidth requirement for each flow – for streaming video flows, these are supplied with the API call, while for bulk transfers these are computed periodically as described in §3.3.1. Further, we aggregate the bandwidth requirements of all flows at a client (since the client is single-homed) to obtain a single bandwidth number b_i for the client, and (b) The network neighborhood of a client, namely the set of APs to which the client can connect (if WiFi pooling is enabled). For our test-bed, we obtain this information by writing a C-program on our APs running dd-wrt that opens a monitor port and scans for all client transmissions, records their signal strengths, and reports these back to our controller.

The **output** of the algorithm is an assignment of clients to APs. The **objective** function is to balance the load, or equivalently minimize the maximum (peak) load, across the access links, so that the minimum residual bandwidth at any access link is maximized to accept future traffic flows. The problem is NP-hard in general, as can be demonstrated by an easy reduction from the job shop scheduling problem as follows. Given a job scheduling problem with n jobs of variable sizes and m machines, map each job to a client with bandwidth requirement b_i equal to the size of the job, and map each processor to an AP. The mutual constraints between jobs and machines are mapped to equivalent constraints between clients and APs (representing the feasible associations between clients and APs). A polynomial-time algorithm that minimizes the maximum load across the APs will therefore minimize the makespan (i.e. total length of schedule) for the job scheduling problem, which is known to be an NP-hard problem. This reduction shows that the problem of minimizing the maximum load across access links is also NP-hard in general.

To obtain an approximate solution in reasonable time for the general case (with WiFi pooling enabled), we adapt the best-known heuristic, known as LPT (Longest Processing Time) [8], whereby we sort the clients in descending order of bandwidth requirement, and assign each in turn to the feasible AP with the largest residual bandwidth. This algorithm can be executed in linear time, and is known to be within a factor of 4/3 of the optimum [8]. The next section evaluates our virtualization algorithm via simulation of real traces comprising over 10 million flows.

4. SIMULATION AND TRACE ANALYSIS

We now evaluate the efficacy of our solution by applying it to real trace data. Obtaining data from residential premises at large scale is difficult; instead we use a 12-hour trace comprising over 10 million flows taken from our University campus network. Though the latter will differ in some ways from residential traces, we believe it still helps us validate our solution with real traffic profiles. We describe the characteristics of the data trace and the network topology, and then quantify the benefits from our virtualization method.

4.1 Trace Data and Campus Network

Our trace data was obtained from the campus web cache, containing flow level logs stored in the Extended Log File Format (ELFF). Each row pertains to a flow record, and includes information such as date and time of arrival, duration (in milliseconds), volume of traffic (in bytes) in each direction, the URL, and the content type (video, text, image, etc.). Our flow logs cover a 12 hour period (12pm-12am) on 16th March 2010, comprising 10.78 million flows and 3300 unique clients.

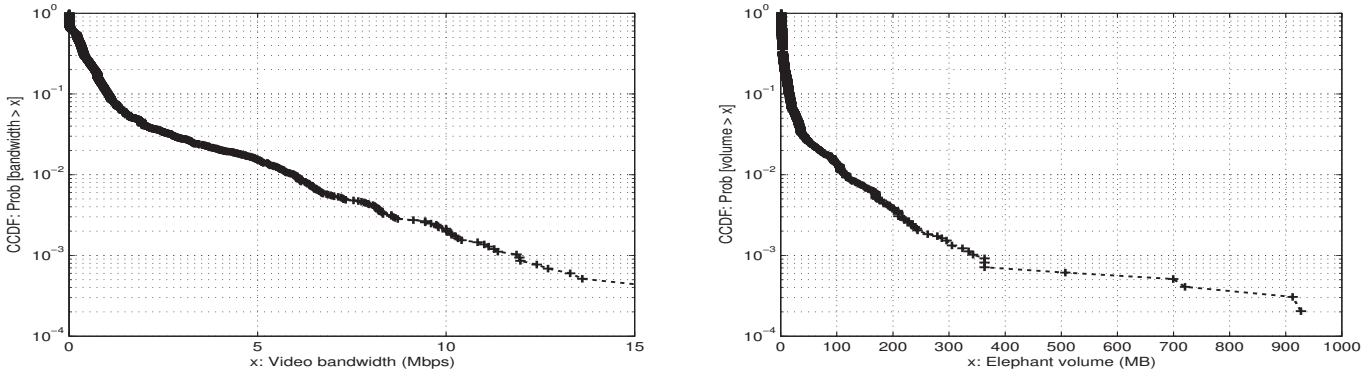

Figure 2: Campus trace CCDF of (a) video flow bandwidth and (b) elephant flow size.

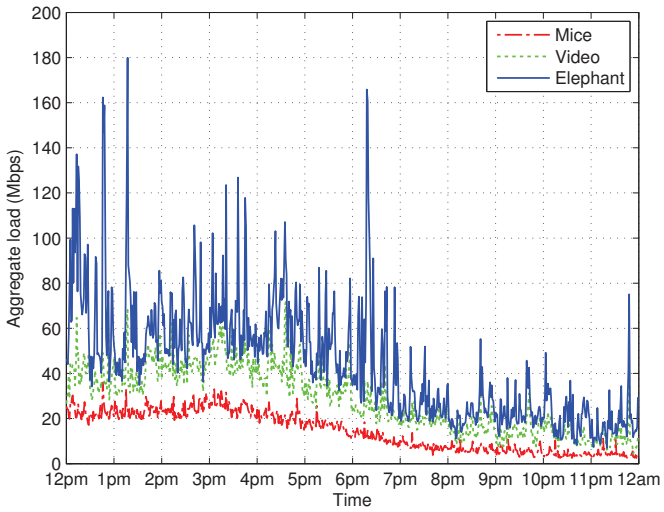

Figure 3: Aggregate load

For our evaluation we categorize flows into three types: video, mice, and elephants. Video flows are identified by their content type, and were found to be predominantly from YouTube. The remaining flows are categorized as mice or elephants based on their download volume. Mice flows are representative of web-page views, for which the user expects immediate response, and are defined as flows that download up to 1 MB (chosen conservatively to be above the average web-page size of 700 KB [1] recorded in 2010). Elephant flows are defined to be those of size over 1 MB, and assumed to be "elastic", i.e., the user can tolerate some transfer delays. Of the 10.78 million flows, we found that the vast majority (10.76 million or 99.8%) of flows were mice, while there were only 11,674 video and 9,799 elephant flows. However, in terms of volume, the three categories were roughly equal, constituting respectively 32%, 32%, and 36% of the traffic download. Note that peer-to-peer traffic does not go through the web-cache, and consequently elephant transfers are likely to be under-represented in our trace. Nevertheless, the traffic characteristics of our trace are reasonably consistent with prior observations of Internet traffic.

A time trace of the traffic volume in each category, averaged over 1-minute intervals over the 12-hour period, is shown in Fig. 3. The bottom curve corresponds to mice flows, and we found that most (93%) mice flows complete within a second, and very few (0.1%) mice flows download more than 300 KB, consistent with published findings [21].

Video traffic volume (as an increment over the mice traffic volume) is shown by the middle line in Fig. 3. To evaluate the impact of our solution on video quality, we assume that video flows have a roughly constant rate (this allows us to measure quality as the fraction of time that the video stream does not get its required bandwidth). This rate is derived by dividing the video flow traffic volume by its duration. To account for the fact that video streaming uses playback buffers that download content ahead of what the user is watching, we added 40 seconds to the video flow duration, consistent with the playback buffer sizes reported for YouTube [22]. The video flow traffic rate CCDF is depicted in Fig. 2(a), and shows that more than 98% of video flows operate on less than 5 Mbps, and less than 0.2% of flows use more than 10 Mbps. The video flow duration distribution (plot omitted) also decays rapidly – only 10% of video views last longer than 3 minutes, and only 1% are longer than 10 minutes.

The total elephant traffic volume (as an increment over the mice and video traffic) is shown by the top curve in Fig. 3. We observe several large spikes, indicating that bulk transfers can sporadically impose heavy loads on the network. In Fig. 2(b) we plot the CCDF of the file size, and find that it decays rapidly initially (about 1% of flows are larger than 100 MB), but then exhibits a long tail, with the maximum file size being close to 1 GB in our trace.

The above traffic trace is simulated over a residential network topology modeled on the campus from which the trace is taken. It consists of 10 four-storeyed apartment buildings, each building containing 30 residences. Each residence has a broadband capacity of 20 Mbps, and is assumed to be equipped with a wireless access point. The coverage overlaps of these access points was obtained from the WiFi maps available for the campus buildings. In each simulation run, clients are mapped to a randomly chosen apartment in a randomly chosen building, yielding a roughly uniform density of 11 clients per household. Clients are only allowed to connect to one access point at a time, i.e. we assume clients do not have interface virtualization capability, and the WiFi overlap maps are used to decide upon feasible client attachment points. Our simulations using these overlap maps show that a client is on average within range of 5.8 access points, which is consistent with the number observed in residential networks by earlier studies [10].

4.2 Simulation Methodology and Metrics

We wrote a native simulation that takes flow arrivals from the trace as input, and performs slot-by-slot (where a slot

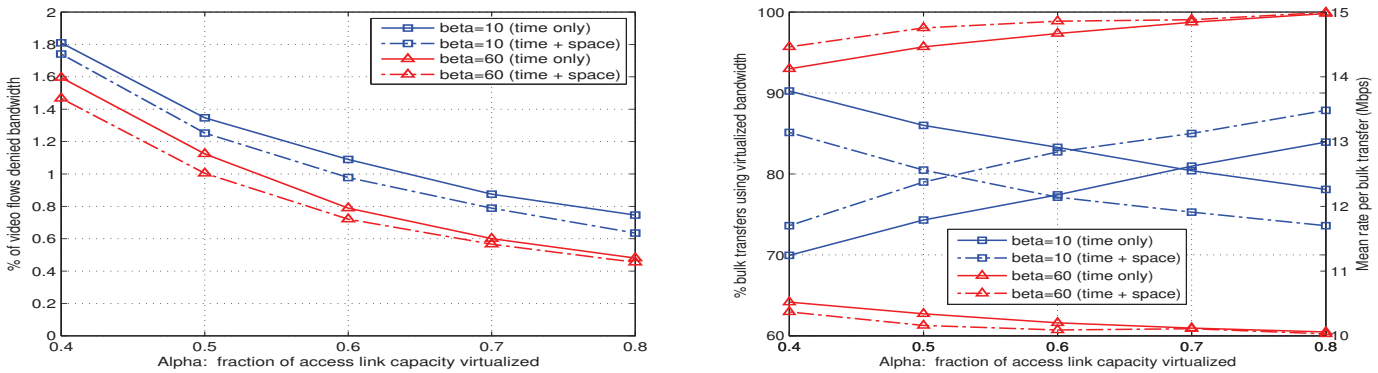

Figure 4: (a) Video allocation failures versus α, and (b) Bulk transfer allocation success (left axis, ascending curves) and mean rate (right axis, descending curves) versus α.

is of duration one second) service on each access link. All video flows, and elephant flows of size larger than 10 MB, are assumed to utilize the API. Such flows are allocated their own queue, whereas flows that do not call the API (or are denied by the API) share a best-effort queue. Within the best-effort queue, mice flows (that transfer less than an MB) are given their bandwidth first (since they are assumed to be in TCP slow-start phase), and the remaining bandwidth is fairly divided amongst the long (video and elephant) flows (since such flows are typically in TCP congestion avoidance phase). Reserved video flows are assumed to be constant bit rate (as discussed above), and assured their bandwidth. The minimum rate allocation for bulk transfer flows is updated dynamically based on their progress, and any spare capacity is shared fairly amongst the queues based on their minimum allocations. We emphasize that our scheduler is work conserving, and does not waste any link capacity if traffic is waiting to be served.

4.3 Performance Results

The slackness in delay bound chosen for the bulk transfer API calls impacts performance for both streaming video and bulk transfer flows. We define elasticity parameter β for a bulk transfer flow as the ratio of the chosen delay bound to the expected delay when the entire access link capacity is available to the flow. Though in reality each bulk transfer will choose its own elasticity (possibly via some negotiation between the user and the CP), to simplify the parameter space in this study we assume that all bulk transfer flows use the same elasticity parameter β. For our first set of results we choose $\beta = 10$, i.e., each bulk transfer requests at least one-tenth (rather than its fair share) of the access link capacity. For given β, we plot in Fig. 4 the impact of the degree of virtualization α (assumed to be the same at each household) on performance for video and bulk transfer performance.

Impact of virtualization degree α on call admissions: Fig. 4(a) shows that video allocation failures reduce as α increases, since a larger α makes more access bandwidth available for allocation. For example, when $\beta = 10$ and considering only time scheduling (solid line with box markers in the figure), it is seen that increasing α from 0.4 to 0.8 reduces allocation failures by more than half, i.e. from 1.8% to 0.75%. Similarly, Fig. 4(b) shows that bulk transfer allocation success (left axis, ascending curves) increases with α; for example, allocation success increases from 70% to 84% as α increases from 0.4 to 0.8. We also note that as α increases, the average bandwidth available to each bulk transfer flow falls marginally (right axis, descending curves in Fig. 4(b)), reflecting the fact that the virtualized capacity is shared by more (video and bulk) flows that are admitted.

Impact of bulk transfer stretch β: We now increase the elasticity factor β from 10 to 60 for all bulk transfer flows – this six-fold increase is chosen so a 10-minute download is now allowed to take an hour. This allows bulk transfers to be stretched, and is expected to create extra room for accepting more video flows. This is corroborated by the solid line with triangular markers in Fig. 4(a), showing that allocation failures drop by about 20% compared to using $\beta = 10$. For example, at $\alpha = 0.8$, allocation failures fall from 0.75% to 0.48%, representing an extra 30 video flows that can be given bandwidth assurance. It is interesting to note from Fig. 4(b) that the bulk transfer flows still receive a sufficiently high average bandwidth (over 10 Mbps); this provides an incentive for the CP to call the bulk transfer API with a relatively slack deadline to reduce transfer cost without (in most cases) significantly increasing transfer delay.

Impact of WiFi pooling: Lastly, when space scheduling is enabled, allowing clients engaged in bulk transfers to be migrated across to lighter loaded APs, there is a slight reduction in video flow rejections, as shown by the dashed lines (with box markers corresponding to $\beta = 10$ and triangular markers for $\beta = 60$) in Fig. 4(a). The benefit of space sharing is more evident for bulk transfers – Fig. 4(b) shows that space sharing (dashed lines in figure) permits a larger number of bulk transfers to be accepted, confirming that using multiple network paths enables more efficient usage of network resources.

A detailed look at video performance improvement: To understand the impact of virtualization on video quality in more detail, we plot in Fig. 5 the CCDF of the fraction of time for which a video flow does not receive its required bandwidth. It is seen that when no virtualization is employed (i.e. $\alpha = 0$), approximately 3% of flows experience some level of degradation, with around 1% of flows not receiving their required bandwidth more than half the time (i.e., being severely degraded). By contrast, with our scheme that virtualizes 80% of the link capacity, less than 0.8% of flows experience any degradation at all (with $\beta = 10$), and this number can be reduced to below 0.5% if bulk transfers elasticity is increased to $\beta = 60$.

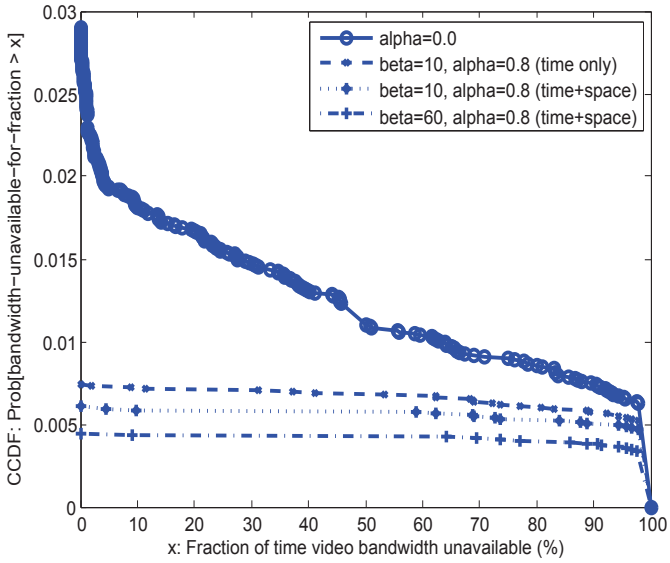

Figure 5: Video bandwidth unavailability CCDF

Summary: The key observations from our evaluation are that: (a) Even without WiFi pooling, virtualization significantly improves video performance, by virtue of being able to identify and squeeze elastic (bulk transfer) traffic, (b) The more bandwidth a user permits their ISP to virtualize (parameter α), the more video performance improves for that user, (c) The higher the slack β chosen for bulk transfers, the better video performs, (d) WiFi pooling (aka space scheduling) improves video performance somewhat, while bulk transfer rates are significantly improved (recall that video flows are not migrated but bulk transfers are).

5. PROTOTYPE IMPLEMENTATION AND EXPERIMENTATION

We prototyped our scheme in a small testbed, depicted in Fig. 6, hosted in a 18m x 12m two-level shed, to emulate a small part (3 homes, each with multiple clients) of a residential ISP network. The objectives of this experimental setup are to demonstrate the feasibility of our scheme with real equipment and traffic, and to evaluate the benefits of virtualization for real video and bulk-transfer streams.

5.1 Hardware and Software Configuration

Network Topology: The clients are within range of their home AP, as well as the other two APs. Each AP has broadband capacity 10 Mbps (achieved by disabling auto-negotiate and forcing the interface speed), emulating a DSL/cable/PON service. The APs connect back to an access switch (emulating a DSLAM, cable head-end, or OLT), which is back-ended with an OpenFlow capable Ethernet switch. This connects through a network of switches (emulating the ISP backbone network) to the controller (that implements the API) and to a delay emulator that introduces 5 ms of delay before forwarding traffic on to the servers through the corporate network (the delay emulator and corporate network together emulate the Internet).

Openflow switch: Our switch was a 64-bit Linux PC with 6 Ethernet ports, running the OpenFlow 1.0.0 Stanford reference software implementation. It supported 200 Mbps throughput without dropping packets, which is sufficient for

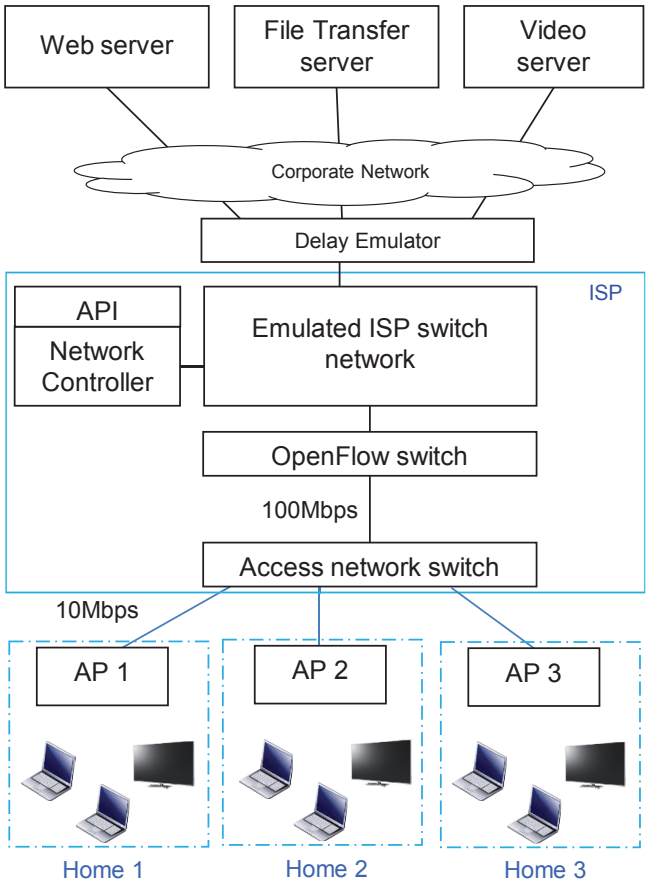

Figure 6: Network Architecture

our experiments. The switch has a default best-effort FIFO queue for each home, and a separate queue was created for each flow that made a successful API call to the controller. Linux Hierarchical Token Buckets (HTBs) assure minimum bandwidth to those flows, providing the "slicing" function.

Network controller: We used the POX OpenFlow controller and developed Python modules that used the messenger class to execute the API calls using JSON from our video and bulk-transfer servers. Successful API calls result in the installation of a flow table entry at the OpenFlow switch to direct the traffic along the desired path. We also implemented the algorithm described in §3.3 at the controller, that makes call admission decisions, and polls the switch every 10 seconds to check the volume sent for each bulk-transfer flow, computes the minimum bandwidth required to complete the transfer within the agreed time, and configures this for the HTB queue at the switch. This periodic reconfiguration of bandwidth for bulk-transfer flows involved very low transmission overhead (of the order of a few bytes per second per flow).

Video server: A Python scripted video on demand server was developed using Flup. For each user video request, the server calls the API via a JSON message to the network controller. An example is: {hello: jukebox, type: minbw, nwsrc: `10.10.7.31/32`, nwdst: `10.10.5.18/32`, proto: 6, sprt: 8080, dprt: 22400, bw: 7600}. In this case the server requests a minimum bandwidth of 7.6 Mbps for TCP on the path from `10.10.7.31:8080` (server) to `10.10.5.18:22400` (client). The server executes a new VLC (v2.0.4) instance

for each video stream, and periodically renews the bandwidth reservation until user video playback ends with TCP disconnection.

Bulk transfer server: When the bulk transfer server receives a request from a client, it calls the API at the network controller via a JSON message. An example is: {hello: jukebox, type: bulk, nwsrc: `10.10.7.31/32`, nwdst: `10.10.5.18/32`, proto: 6, sprt: 24380, dprt: 20, len: 1800000, deadline: 3600}. In this case the server requests a bulk transfer of 1.8GB by TCP on the path from `10.10.7.31:24380` to `10.10.5.18:20`. The deadline parameter indicates that the transfer can take up to 1 hour. If the controller accepts the request, the flow is given a dedicated queue, treated as "elastic", and can be charged at a lower rate.

Wireless APs: We used standard TP-LINK WR1043ND APs. The first set of experiments use the APs straight out-of-the-box, while the APs ran in layer-2 mode (i.e. routing, DHCP, and NAT disabled) with dd-wrt v24 for space scheduling experiments so that user devices could migrate across APs without changing IP addresses or disrupting active flows. Our network controller logs in to the APs via SSH to migrate clients when needed using MAC address blacklisting. To identify clients that are within radio range of each AP (and hence eligible for migration), we wrote a C program for the AP that opens a monitor port on the WiFi adaptor to put the AP in promiscuous mode, and logs all WiFi client transmission signal strengths. The AP reports the wireless neighborhood to the controller every 60 seconds – the messaging overheads of this were minimal, at just a few bytes per second.

User clients: Each home has three clients, implemented using standard computers. Client C1 represents a large-screen device (e.g. PC or TV) and client C2 a small-screen device (e.g. tablet/phone) on which users watch videos, while client C3 represents a PC or media gateway that does both web-browsing and bulk transfers. Browsing took place within Internet Explorer (IE) v10, and a web-page of 1.1 MB containing text and images is accessed. All videos were played by the VLC IE plugin.

User Traffic: Clients run PowerShell scripts to automatically generate traffic representative of the average home. Clients C1 and C2 are either idle or streaming video, and a Markov process controls the transitions, with 40% of time spent idle and 60% watching video. Client C1 streams a high bandwidth video in MPEG-1/2 format, allocated a peak bandwidth of 7.5 Mbps, and having mean rate of 5.9 Mbps averaged over 3-second interval samples. Client C2 streams a lower bandwidth video in MPEG-4v format, allocated a peak bandwidth of 2.1 Mbps and having a mean rate of 1.3 Mbps. Client C3 can be in idle, browsing, or bulk-transfer states. For browsing it opens IE and loads a 1.1 MB web-page from our web-server. The user is assumed to read the web-page for 10 seconds, reloads the web-page, and the process repeats. We disabled IE's cache so that it downloaded the full web page on every access, which lets us compare the download times for the page across various runs. For bulk-transfers the file sizes were chosen from a Pareto distribution with shape parameter 4.5, and scale parameter such that files are between 100 and 500 MB with high probability. The idle periods are log-normal with mean 10 minutes and standard deviation 2 minutes.

Metrics: The video streaming quality is measured in terms of Mean Opinion Scores (MOS). To automatically evaluate MOS, we rely on the technique of [18] that combines initial buffering time, mean rebuffering duration, and rebuffering frequency to estimate the MOS (with a configured playback buffer of 3 seconds). Our VLC IE plugin was instrumented with Javascript to measure these parameters and compute the MOS. Our client script also measured the times taken for each bulk transfer and web-page download.

App	$\alpha = 0$		$\alpha = 0.8$		$\alpha = 1$	
	mean	std	mean	std	mean	std
C1 MOS	2.87	0.44	3.10	0.31	3.25	0.01
C2 MOS	3.25	0.00	3.25	0.01	3.25	0.01
Page load (s)	2.84	0.86	3.10	1.61	4.85	3.55
FTP stretch	1.60	0.20	1.97	0.77	2.45	1.07

Table 1: Video, browsing, and ftp performance with varying virtualization factor α.

5.2 Experimental Results

We conducted tens of experiments varying the user selected parameter α that controls the degree of virtualization, and the elasticity β for bulk transfer flows. The impact of these parameters on video quality, file transfer times, and browsing delays is discussed next with time scheduling (§5.2.1) and space scheduling (§5.2.2).

5.2.1 Virtualizing Individual Access Links

In our first set of experiments, each home's access link is independently virtualized. Consequently, broadband capacity is not shared across homes, and client migrations are disallowed – this solution is easier to deploy at scale since it does not require any modifications to the home gateways, but has potentially reduced benefits as multiple paths to household clients are not exploited.

In table 1 we show how the quality for the various applications depends on the fraction α of household link capacity that is virtualized. The low-rate video (2.1 Mbps peak) on client C2 always gets a near-perfect MOS of 3.25. This is unsurprising, since a fair share of the link capacity suffices for this video to perform well in our experiments, and reservations are not necessary. The high-rate video stream (7.5 Mbps peak) on client C1 however sees marked variation in quality: disabling virtualization with $\alpha = 0$ makes the video unwatchable most of the time, with low average MOS of 2.87 (standard deviation 0.44), while complete virtualization with $\alpha = 1$ always successfully allocates bandwidth to this stream, yielding a perfect MOS of 3.25. With $\alpha = 0.8$, the average MOS degrades to 3.10 (standard deviation 0.31) since allocations fail when the other video stream is also active.

The table also shows that α has the converse effect on web-page load time: when $\alpha = 0$, the web-page loads in 2.84s on average (standard deviation 0.86s), while increasing α to 0.8 and 1 steadily increases the average time taken for page loads; furthermore, the standard deviation also increases, indicating that downloads times become more erratic as α increases. This is not surprising, since web-page downloads (and mice flows in general) will not allocate resources via the API call, and their performance suffers when bandwidth is allocated to other reserved flows. This trade-off between video quality and web-page load-time illustrates that users should adjust their household α value (via trial-and-error or other means beyond the scope of this paper) in line with

their traffic mix and the relative value they place on each traffic type.

The performance of a bulk transfer flow is measured in terms of its "stretch", i.e. the factor by which its transfer delay gets elongated compared to the baseline case where it has exclusive access to the entire access link capacity. Table 1 shows that with no virtualization, bulk transfer flows get stretched by a factor of 1.6, and the stretch increases 1.97 at $\alpha = 0.8$ and 2.45 at $\alpha = 1$. This is both expected and desired, since increasing α allows the video streams to get higher quality, which comes at the cost of stretching the elastic bulk-transfers.

5.2.2 With Space Scheduling Added

We now pool home broadband capacity and begin with the ideal case of $\alpha = 1$ at all households. Recall that we only allow migrations for clients that are engaged in bulk transfers (known via the API call), thereby protecting disruptions for video streams and best-effort (browsing) traffic. Every 10 seconds, our algorithm reassesses and reallocates bandwidth to bulk transfer flows, and can remap bulk-transfer clients to other APs in order to leverage alternate paths of higher available capacity.

Our results showed that the stretch factors for bulk transfers in the three households reduced significantly, from the original values of $(1.92, 2.19, 2.08)$ to $(1.37, 1.73, 1.44)$ when WiFi pooling is enabled. We could visually see instances where a household is doing video and bulk transfer, and our algorithm migrates the bulk transfer to an idle neighbor AP. It can be seen that virtualizing the WiFi pool makes "peak demand offload" possible. This reduces transfer time by 20-30% without impinging on other (video and web-browsing) services in the network, and is therefore beneficial to all parties.

We also conducted experiments in which users had different traffic profiles. Unsurprisingly, we found that WiFi pooling is more beneficial to power users, as it lets them tap into unused bandwidth from neighboring homes. However, this affects neither the quality for the light users, nor their Internet expense, since the ISP can account for downloads on a per-device basis that can then be billed to the appropriate household. Moreover, a user who is uncomfortable has the option to opt out of WiFi pooling or reduce their α value.

6. RELATED WORK

The body of literature on QoS/QoE is vast, and *bandwidth-on-demand* capabilities have been envisaged since the days of ATM, IntServ and RSVP. These mechanisms equip the ISP with tools to manage quality in their own network, but little has been done by way of exposing controls to end-users and content providers. Early attempts at exposing QoS to external entities include the concept of bandwidth broker for ATM networks [19], and protocols for QoS negotiation (e.g. XNRP [23]). Tools for exposing network bandwidth availability are starting to emerge, though predominantly for data center users, such as Juniper's Bandwidth Calendaring Application [25] implemented over an OpenFlow-based network. Bandwidth-on-demand for bulk data transfers between data centers has also been explored in the Globally Reconfigurable Intelligent Photonic Network [16] and NetStitcher [13], with the latter exploiting the elasticity in bulk data transfer to schedule it during diurnal lulls in network demand. Elasticity has also been leveraged by [4] to improve ISP access network performance, with time-dependent pricing explored in [11].

The works closest to ours are those that virtualize the access [17] and home [31, 30] networks. NANDO [17] allows multiple ISPs to share infrastructure, and consumers can choose the ISP on a per-service basis. This model is very attractive for public access infrastructure (e.g. in Australia or Singapore), but it remains to be seen if private ISPs will be willing to share infrastructure with each other. In [31], the home network is sliced by the ISP amongst multiple content providers. With this approach the ISP cedes long-term control of the slice to the CP (it is however unclear what policies dictate the bandwidth sharing amongst the slices), which is different from our architecture in which the ISP only "leases" well-specified resources to the CP on a short-term per-flow basis. Both models have merits and are worth exploring, though we believe our approach is likely to be more palatable to ISPs as they can retain more control over their network. Lastly, [30] gives users control of how their home network is sliced, but requires a higher level of user sophistication that we have tried to bypass.

The concept of WiFi pooling has also been studied before, with works such as [10] showing increase in neighborhood throughput from resource pooling. Note that unlike many prior approaches, we effect our solution under centralized SDN control of the APs and do not demand changes at end-user clients, a principle that is also being followed in commercial deployments we have cited earlier.

7. CONCLUSIONS

In this paper we have proposed an architecture for virtualizing the access network via open APIs to enable dynamic service quality management. Our architecture provides the motivation and means for all parties to engage: content providers can selectively choose to enhance service quality for flows in line with their business models; ISPs can monetize their access infrastructure resources on a per-flow basis without revealing network internals; and users can readily adjust the degree of virtualization to suit their usage pattern. We developed an algorithm that achieves efficient virtualization by leveraging the time elasticity of bulk transfer applications, as well as the spatial overlap of WiFi coverage in urban areas. We simulated our algorithm on real traffic traces comprising over 10 million flows to show that virtualization can reduce video quality degradations by 50-70%, for a modest increase in bulk transfer delays. Finally, we prototyped our scheme on a small testbed comprising OpenFlow-compliant switches, off-the-shelf access points, and unmodified clients, to show how the user can control the trade-off between video MOS, bulk transfer rates, and web-page load-times.

Our work is a first step towards showing how the agility and centralization afforded by SDN technology presents a unique opportunity to overcome the long-standing impasse on service quality in access networks. Needless to say, many challenges are yet to be overcome to make this a reality, such as enriching the API to include other application use-cases (e.g. low-latency gaming or virtual reality applications), extending the API end-to-end across network domains via federation, and ultimately allowing end-users direct access to APIs that let them customize their broadband experience to match the devices and applications in their household.

8. ACKNOWLEDGEMENTS

The authors thank Google for supporting this project via a Research Award, and specifically Josh Bailey who has been a close collaborator throughout this work. We also want to thank the anonymous reviews who gave us valuable feedback on our submission, and a special thanks to Nikolaos Laoutaris for his tremendous help and support during the revision process.

9. REFERENCES

[1] HTTP Archive. http://www.httparchive.org/.

[2] S. Akhshabi, A. Begen, and C. Dovrolis. An experimental evaluation of rate-adaptation algorithms in adaptive streaming over http. In *Proc. ACM MMSys*, Feb. 2011.

[3] Cisco Internet Business Solutions Group. Moving Toward Usage-Based Pricing. http://www.cisco.com/web/about/ac79/docs/clmw/Usage-Based-Pricing-Strategies.pdf, Mar. 2012.

[4] P. Danphitsanuphan. Dynamic Bandwidth Shaping Algorithm for Internet Traffic Sharing Environments. In *Proc. World Congress on Engineering*, July 2011.

[5] F. Dobrian, V. Sekar, A. Awan, I. Stoica, D. Joseph, A. Ganjam, J. Zhan, and H. Zhang. Understanding the Impact of Video Quality on User Engagement. In *Proc. ACM SIGCOMM*, Aug. 2011.

[6] M. Ghobadi, Y. Cheng, A. Jain, and M. Mathis. Trickle: Rate Limiting YouTube Video Streaming. In *Proc. USENIX ATC*, Jun 2012.

[7] E. Goma, M. Canini, A. Lopez Toledo, N. Laoutaris, D. Kostić, P. Rodriguez, R. Stanojević, and P. Yagüe Valentin. Insomnia in the Access or How to Curb Access Network Related Energy Consumption. In *Proc. ACM SIGCOMM*, Aug. 2011.

[8] R.L. Graham. Bounds on Multiprocessing Timing Anomalies. *SIAM Journal on Applied Mathematics*, 17(2):416–429, 1969.

[9] Internet Society. Bandwidth Management: Internet Society Technology Roundtable Series. http://www.internetsociety.org/sites/default/files/BWroundtable_report-1.0.pdf, Nov 2012.

[10] S. Jakubczak, D.G. Andersen, M. Kaminsky, K. Papagiannaki, and S. Seshan. Link-alike: Using Wireless to Share Network Resources in a Neighborhood. *ACM SIGMOBILE MC2R*, 12(4):798–814, Oct 2008.

[11] C. Joe-Wong, S. Ha, and M. Chiang. Time-dependent broadband pricing: feasibility and benefits. In *Proc. IEEE ICDCS*, June 2011.

[12] S. Krishnan and R. Sitaraman. Video Stream Quality Impacts Viewer Behavior: Inferring Causality Using Quasi-Experimental Designs. In *Proc. ACM IMC*, Nov. 2012.

[13] N. Laoutaris, M. Sirivianos, X. Yang, and P. Rodriguez. Inter-Datacenter Bulk Transfers with NetStitcher. In *Proc. ACM SIGCOMM*, Aug. 2011.

[14] Light Reading. Comcast Turns Homes Into Hotspots. http://www.lightreading.com/document.asp?doc_id=703027, Jun. 2013.

[15] X. Liu, F. Dobrian, H. Milner, J. Jiang, V. Sekar, I. Stoica, and H. Zhang. A Case for a Coordinated Internet Video Control Plane. In *Proc. ACM SIGCOMM*, Aug. 2012.

[16] A. Mahimkar, A. Chiu, R. Doverspike, M.D. Feuer, P. Magill, E. Mavrogiorgis, J. Pastor, S.L. Woodward, and J. Yates. Bandwidth on Demand for Inter-Data Center Communication. In *Proc. ACM HotNets Workshop*, Nov. 2011.

[17] J. Matias, E. Jacob, N. Katti, and J. Astorga. Towards Neutrality in Access Networks: A NANDO Deployment With OpenFlow. In *Proc. Int'l Conf. on Access Networks*, June 2011.

[18] R. Mok, E. Chan, and R. Chang. Measuring the quality of experience of HTTP video streaming. In *Proc. IFIP/IEEE Int'l Symp. on Integrated Network Management*, May 2011.

[19] K. Nahrstedt and J.M. Smith. The QoS Broker. *IEEE Multimedia Magazine*, 2(1):53–67, 1995.

[20] M. Nicosia, R. Klemann, K. Griffin, S. Taylor, B. Demuth, J. Defour, R. Medcalf, T. Renger, and P. Datta. Rethinking flat rate pricing for broadband services. White Paper, Cisco Internet Business Solutions Group, July 2012.

[21] S. Ramachandran. Web metrics: Size and number of resources. https://developers.google.com/speed/articles/web-metrics, May 2010.

[22] A. Rao, Y. Lim, C. Barakat, A. Legout, D. Towsley, and W. Dabbous. Network characteristics of video streaming traffic. In *Proc. ACM CoNEXT*, Dec. 2011.

[23] K. Rothermel, G. Dermler, and W. Fiederer. QoS Negotiation and Resource Reservation for Distributed Multimedia Applications. In *Proc. IEEE International Conference on Multimedia Computing and Systems*, Canada, Jun 1997.

[24] Sandvine-Intelligent Broadband Networks. Global Internet Phenomena Report. https://www.sandvine.com/trends/global-internet-phenomena/, 2012.

[25] H. Sugiyama. Programmable Network Systems Through the Junos SDK and Junos Space SDK. In *World Telecommunications Congress*, Mar. 2012.

[26] S. Sundaresan, W. Donato, N. Feamster, R. Teixeira, S. Crawford, and A. Pescapè. Broadband Internet Performance: A View From the Gateway. In *Proc. ACM SIGCOMM*, Aug 2011.

[27] Telefonica. BeWiFi. http://www.bewifi.es/, 2013.

[28] The European Telecom. Network Operators' Association. ITRs Proposal to Address New Internet Ecosystem. http://www.etno.be/datas/itu-matters/etno-ip-interconnection.pdf, Sep. 2012.

[29] N. Vallina-Rodriguez, V. Erramilli, Y. Grunenberger, L. Gyarmati, N. Laoutaris, R. Stanojevic, and K. Papagiannaki. When David helps Goliath: The Case for 3G OnLoading. In *Proc. ACM Hotnets*, Seattle, WA, USA, Oct 2012.

[30] Y. Yiakoumis, S. Katti, T. Huang, N. McKeown, K. Yap, and R. Johari. Putting Home Users in Charge of their Network. In *Proc. ACM UbiComp*, Sept. 2012.

[31] Y. Yiakoumis, K. Yap, S. Katti, G. Parulkar, and N. McKeown. Slicing Home Networks. In *Proc. ACM SIGCOMM HomeNets Workshop*, Aug. 2011.