User Control of Quality of Experience in Home Networks using SDN

Himal Kumar
Department of Electrical Engineering
Indian Institute of Technology, Patna, India
Email: himal.ee10@iitp.ac.in

Hassan Habibi Gharakheili, Vijay Sivaraman
School of Electrical Engineering and Telecommunications
UNSW, Sydney, Australia
Email: {h.habibi, vijay}@unsw.edu.au

***Abstract*—Home networks are becoming increasingly complex, with many household devices (PCs, tablets, phones, media gateways, smart TVs) and diverse user applications (browsing, video streaming, peer-to-peer, VoIP, gaming) sharing the single broadband access link. In today's architecture the traffic streams compete for bandwidth on a best-effort basis, resulting in poor quality of experience for users. In this paper, we leverage the emerging paradigm of software defined networking (SDN) to enable the ISP to expose some controls to the users to manage service quality for specific devices and applications in their household. Our contributions are to develop an architecture and interface for delegation of such control to the user, and to demonstrate its value via experiments in a laboratory test-bed using three representative applications: video, web-browsing, and large downloads.**

I. Introduction

The home network is becoming increasingly complex. Internet-capable household devices are proliferating, ranging from computers, phones and tablets to TVs, game consoles, smart meters, and domestic appliances. These devices are running increasingly demanding applications, ranging from streaming video and tele-conferencing to gaming and large downloads. It is becoming increasingly untenable for these devices and applications to continue to share the broadband Internet access capacity on a best-effort basis. For example, if the desktop computer starts downloading a large software update while the user is streaming video to their iPad or playing a real-time game on their Xbox, quality of experience can degrade significantly, leading to user frustration. Indeed, large-scale studies are showing that the impact of access link congestion on video quality, in the form of startup delays and rebuffering events, is leading to higher user abandonment [1], and the consequent customer dissatisfaction may also be responsible for ISP churn [2].

The problem posed by increasing home network complexity may provide an opportunity for ISPs to differentiate their services by tapping into the service quality dimension. By giving end-users some means to control service quality for their traffic streams, they can let users customize it to their needs – such a feature is not only monetizable, but also increases customer satisfaction and retention. For example, the user may be allowed to configure bandwidth allocation prioritization on a per-device or per-application basis, say by giving the kids' iPad lower priority than the work-related laptop, or YouTube traffic higher priority than peer-to-peer downloads. This empowers the user, while allowing ISPs to innovate, monetize, and distinguish themselves from competitors via new mechanisms for quality support, and is hence a win-win situation for both.

It might seem at first glance that the user can improve service quality by upgrading their home gateway to support queueing features for prioritization. While this may suffice for uplink traffic, reality is that most traffic (be it video streaming or downloading) is downstream, for which queue management is required in the ISP infrastructure. Any modification therefore requires the user to contact the ISP (such as via a phone call or account management portal) to indicate their changes, often requiring manual configuration to the ISP equipment, which is both time-consuming and expensive. Alternatively, users may use specialized protocols for certain applications (e.g. TCP Nice for background transfers) – however these are limited in the applications they can serve, are too onerous on the user, and require sophistication beyond the reach of most.

We instead believe that the emerging paradigm of software defined networking (SDN) offers the ideal technological platform on which to achieve this capability. SDN allows the ISP to have a programmatic interface by which to interact with the user device or application. This allows the user to dynamically ask the ISP to slice resources for the various services, via a negotiation that is automated and real-time. While this is conceptually simple, there are several challenges that need to be surmounted: (1) how should the user-interface be designed such that it gives the user control across their devices and applications without demanding high sophistication?, (2) how should the programmatic interface be structured such that it is simple to implement and yet general enough to operate across services?, and (3) how is the effect of user control on quality of experience evaluated and quantified? This paper is our attempt at developing and demonstrating an architecture that addresses these challenges. Our specific contributions are:

- We develop the architecture of a system that allows service quality negotiation between the user and ISP, and apply it to use-cases relating to streaming, browsing, and downloading,
- We design a GUI that allows a typical user to specify their requirements on a per-device and per-application basis, and the associated API that the ISP needs to expose to facilitate this capability, and
- We prototype our system and demonstrate its value by quantifying the performance benefits when streaming,

browsing, and downloading applications that share the household Internet access link.

Our work is a first step towards allowing users to control quality of experience across their household devices and applications, and for ISPs to support such capability using SDN technology.

The rest of the paper is organized as follows: §II explains the use-cases considered in this paper. §III gives our system architecture and trade-offs. In §IV we describe our prototype implementation including the GUI, the APIs, and traffic models, and §V describes our experimental results. Relevant prior work is summarized in §VI, and the paper concludes in §VII.

II. 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 three application use-cases: *video streaming*, chosen due to its growing popularity with users, non-real-time *large downloads* chosen for their large volume, and *web browsing* chosen for their 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.

A. Streaming Video

Streaming video, 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. To enhance user quality of experience over best-effort networks, these providers use techniques such as client-side playback buffering, server-side bit-rate adaptation, and TCP instrumentation. However, large-scale studies [3], [1] confirm that video delivery quality is still lacking, with video "freeze" contributing to reduced viewing time and startup delays reducing customer retention. Since variability in client-side bandwidth is one of the dominant contributors to quality degradation, an ideal solution is to have the network "sliced" to explicitly assure bandwidth to the video stream. As mentioned earlier, this slicing has to be performed by the ISP on the user's access link in the downstream direction, and therefore requires some signalling between the user and the ISP. By exposing this signalling via an API, the ISP can monetize the quality dimension, and can obtain explicit stream attributes without requiring to infer them via expensive operations such as deep packet inspection (DPI).

B. Large Downloads

After video, bulk 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. Unlike video, large download 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 the criticality of 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. streaming video) requiring bandwidth assurances.

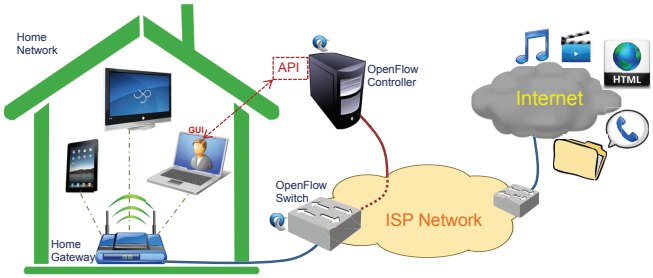


Fig. 1. Home network topology

C. Web Browsing

Though web-browsing constitutes much lower traffic volume than streaming or downloads, it has high monetary value (e.g. Google searches) for users and online retailers. Large online providers like Google and Amazon have acknowledged that even a small increase in web-page load times can result in lost conversions and reduced customer satisfaction, impacting customer loyalty. Retailers, ISPs and users therefore have an interest in ensuring that the network is responsive to short flows associated with web-page views.

III. System Architecture

We now propose a system architecture for user control of quality of experience (QoE) in the home network. We first outline the major architectural choices and trade-offs (§III-A), then describe the operational scenario (§III-B), the APIs (§III-C) and the algorithm (§III-D).

A. Architectural Choices and Trade-Offs

The aim is to slice the access link resources dynamically amongst flows in a programmatic way, so that the network is used as efficiently as possible for enhancing service performance. Unlike current practise whereby ISP use DPI and other expensive tools for classifying traffic flows, we advocate the use of open APIs. This allows home users to choose a resource requirement commensurate with the value of the service, and for the ISP to obtain application parameters (addresses/ports and bandwidth/delay requirements) directly rather than having to infer them. In this paper we focus only on slicing the access link capacity, rather than an end-to-end network resource allocation, since the latter requires coordination across network domains that is deemed a long-term goal; moreover, there is evidence [4] that network bottlenecks often lie in the access and not at the interconnects between networks. Lastly, the API we design is for home users (the primary beneficiaries of improved quality, as noted in [5]), though it can be extended for use by enterprise customers or indeed by content providers themselves.

B. Operational Scenario

Fig. 1 shows a typical access network topology. Each residence has a home gateway to which household devices

connect. The home gateway offers Internet connectivity via a (DSL, Cable, or PON) broadband link, 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 slicing functionality on the user access link.

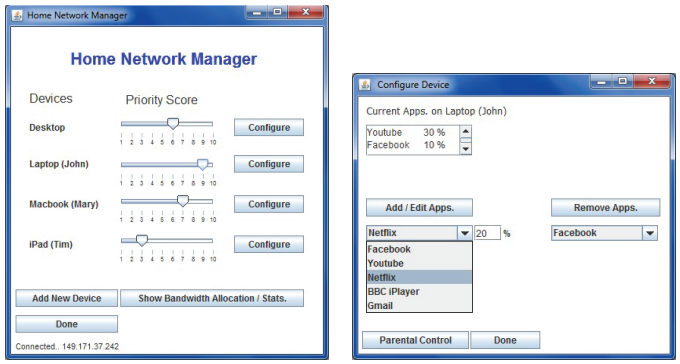
The operational flow of events is as follows. The user's input on desired device and application slice is obtained via a simple GUI (described in the next section) installed on a PC in the home. The GUI then translates these requests into the appropriate API calls exposed by the SDN controller hosted in the ISP network. The controller then executes an algorithm to verify the sanity of the request and determine an appropriate resource allocation for the request, which it then configures into the switching hardware associated with that user's access link. In what follows we describe the APIs in more detail and elaborate on the specific actions required by the user and the ISP.

C. The APIs

We develop a minimalist specification of the APIs; detailed specifications are left for future standardization. The most important API relates to bandwidth assurance, and specifies the: (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 tuple comprising the IP source and destination addresses, and potentially source and destination ports, that identify the flow (consistent with the OpenFlow specification). Note that wildcards can be used to denote flow aggregates. (d) *Bandwidth:* The bandwidth (in Kbps) that is requested by the flows. This API can be used to assure minimum bandwidth to a service like video streaming, as illustrated in our prototype in the next section. 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. Additional APIs for capabilities such as delay control (for elastic applications including large downloads) and parental control (to drop or limit traffic from specified sites) were developed but are not discussed in this paper.

D. The Controller Algorithm

When the requests come in to the SDN controller operated by the ISP, an algorithm is run to determine the feasibility of the request, e.g. ensuring that the bandwidth slices requested by applications from a residence do not exceed the bandwidth capacity on the access link to that household. If feasible, the controller installs rule(s) corresponding to the application parameters obtained from the API call into the switch serving that customer premises. This may involve setting up a new queue to which the application traffic is mapped, and ensuring a resource slice for that queue. An example is provided in the next section that describes our prototype implementation.



(a) Device level control (b) Application level control

Fig. 2. Graphical user interface: (a) device level (b) application level

IV. Prototype Implementation

We developed and prototyped our scheme on a small testbed, depicted in Fig. 3, to emulate a small part (a home with multiple clients) of a residential ISP network. The objective is to demonstrate the feasibility of our scheme with real equipment and traffic, and to evaluate (next section) the benefits of slicing for real video, web-browsing and large download streams.

A. Graphical User Interface (GUI) Design

Our first contribution is to design a GUI that allows the home user to control their household devices and applications in an easy and intuitive way without requiring much technical sophistication. Fig. 2 shows screenshots of our GUI, developed in Java, and running as an application on any client device in the user's house.

Fig. 2(a) shows the screen that allows the user to specify client device level controls. For example, this screen shows four household devices: a desktop machine (shared by the household members), a laptop (belonging to the father), a Macbook (belonging to the mother), and an iPad (belonging to the child). The super-user (say the father) who operates this control panel can configure a "priority", on a scale of $1-10$, for each device, with 10 denoting the highest priority. As will become clear later, a higher priority is associated with a larger bandwidth slice for that device. When a new device is added, the super-user needs to name the device and specify the MAC address associated with this device (this requires some technical sophistication, but can at some point in the future be obviated via automatic discovery of devices).

The user can choose a specific device and configure priorities for specific applications on that device. Fig. 2(b) shows an example screenshot whereby the laptop has been configured with two applications: YouTube has been given a 30% share and Facebook 10%. Further applications can be added, using the drop-down menu that lists popular applications such as NetFlix. Applications can also be edited or removed using this panel, and parental control (bottom button) can also be exercised to restrict access to certain sites from this device.

The device-level priorities and application shares (if specified) on each device are hierarchically used to determine the nominal bandwidth slices as follows: A device is allowed a bandwidth share that corresponds to its weight divided by the combined weights of all devices. For example, the laptop in the

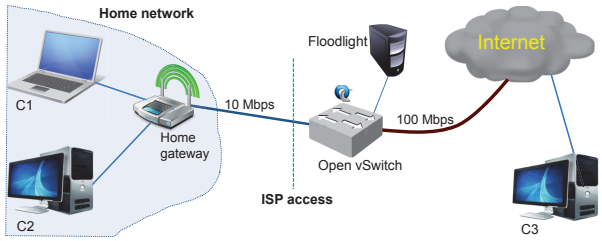


Fig. 3. Architecture of our prototype

above example will get fraction $9/(6+9+7+3) = 0.36$ of the access bandwidth allocated as a minimum, preventing it from getting starved of bandwidth by other devices in the household (e.g. kid's iPad). When a device has application-level control specified, the bandwidth assured for that application is the specified percentage of the device's bandwidth assurance; continuing with the above example, YouTube on the laptop will be assured 30% of the laptop's bandwidth fraction, i.e. an overall fraction $0.3 * 0.36 = 0.108$ of the access link capacity. Note that every device will by default obtain "best-effort" service that contains applications that have not been configured; such service will be assured bandwidth residual from the configured applications.

The configuration set by the user using the GUI are then communicated to the ISP using an API. The API we designed conveys a simple set of parameters using JSON encoding: our minimum bandwidth assurance API format is: {hello: jukebox, type: minbw, numQueue: 2, qid:1, nwsrc: 203.5.76.0/24, nwdst: 149.171.37.162/32, bw: 5500, qid: 2, nwsrc: 0.0.0.0/24, nwdst: 149.171.37.115/32, bw: 4000}. In this case the API call is requesting two queues in the ISP network, with a minimum bandwidth of (a) 5.5 Mbps on the path from YouTube server (which in Australia corresponds to 203.5.76.0/24) to client C1 (i.e. 149.171.37.162) and (b) 4 Mbps on the path from any source to client C2 (i.e. 149.171.37.115). Another example API that allows for parental control is: {hello: jukebox, type: block, nwsrc: 203.5.76.0/24, nwdst: 149.171.37.115/32, action: drop}, which blocks YouTube access for client C2. Lastly, our GUI also allows the user to see bandwidth allocation and usage statistics for each configured device and application (bottom right button in Fig. 2(a)).

B. Network and Traffic Setup

We now briefly describe the experimental setup, shown in Fig. 3, on which we implemented the functionality related to the GUI described above.

Network Topology: Two home clients (C1 and C2) are connected to the access point TP-LINK WR1043ND (i.e. home gateway). The gateway has broadband capacity 10 Mbps (achieved by disabling auto-negotiate and forcing the interface speed), emulating a DSL/cable/PON service. The gateway connects back to an OpenFlow capable Ethernet switch (emulating ISP access device). That switch connects to the Internet via 100 Mbps link and also the controller (that implements the API). The controller server is hosted in the corporate network.

Openflow switch: Our switch is a DELL PowerEdge R620 server with 8 Gigabit Ethernet ports, running the `Open vSwitch 1.10.0` (OVS) [6] on Linux Fedora 19. It supports OpenFlow protocol, provides fine-grained QoS control, and is compatible with the Floodlight controller. The switch has a default best-effort FIFO queue, and a separate queue is created for each (device or application specific) flow that makes a successful API call to the controller. Linux Hierarchical Token Buckets (HTBs) are used to assure minimum bandwidth to each queue, providing the "slicing" functionality. High-priority flow table entries programmed by the network controller override the switch's default layer-2 forwarding decision.

Network controller: We installed the Floodlight [7] OpenFlow controller for the ISP network, and developed Java modules along with static flow pusher that used the messenger class to execute the API calls using JSON. Successful API calls result in the installation of a flow table entry at the OVS to direct the traffic into the desired queue.

User clients: The home has two clients, implemented using standard computers. Client C1 represents a small-screen device (e.g. laptop/tablet/phone) on which user watches videos or does web-browsing, while client C2 represents a PC or media gateway that does large downloads.

User Traffic: Client C1 runs PowerShell scripts to automatically generate traffic representative of the average home. C1 can be in idle, browsing, or video-streaming states, the transitions being determined by a Markov chain. For browsing it opens IE and loads the first author's Facebook home-page from the Internet with a size of about 1-1.5MB. The user is assumed to read the web-page for 10 seconds, reload 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. Client C1 streams an HD video of 720p resolution from YouTube. Client C2 is either idle or downloading a large file. In our experiment, C2 downloads an ISO format image of Fedora 19 Desktop Edition (64-bit) with a size of 951 MB.

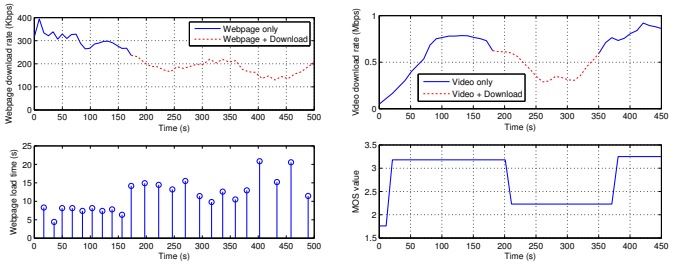
Metrics: The video streaming quality is measured in terms of Mean Opinion Scores (MOS). To automatically evaluate MOS, we rely on the technique of [8] that combines initial buffering time, mean rebuffering duration, and rebuffering frequency to estimate the MOS (with a configured playback buffer of 3 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 web-page download.

V. Experimental Evaluation

We present results from three evaluation runs: QoE improvement from device-level prioritisation, and from application-level slicing using synthetic iPerf traffic and real downloads using the IDM download manager.

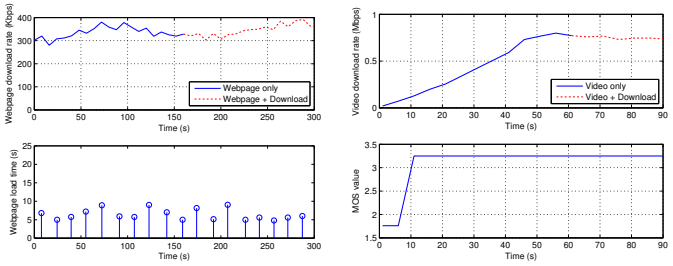
A. Device-level Slicing

To demonstrate the performance over today's best-effort networks, we conduct two experiments: in the first, client C1 is browsing continuously (by loading the Facebook web-page described above), and client C2 starts downloading a large file (the Fedora ISO image as mentioned above) about 175 seconds into the experiment. Fig. 4(a) shows the web-page download rate (upper plot) and the web-page download times



(a) Web browsing (b) Video stream

Fig. 4. Best-effort performance: (a) web browsing, (b) video streaming.



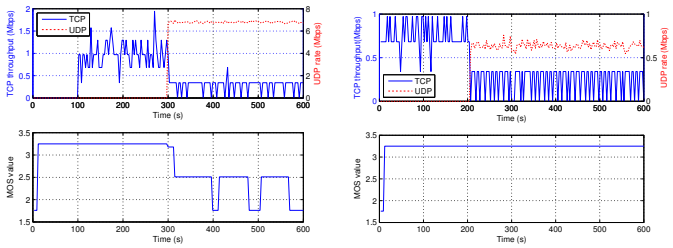
(a) Web browsing (b) Video stream

Fig. 5. Device-level slicing: (a) web browsing, (b) video streaming.

(lower plot). It is clearly seen that till time 175s, the web-page load-time is 7.34s on average, with standard deviation 1.26s. However, once the large download from client C2 commences at time 175s, the average page-load time on C1 nearly doubles to 14.11s, and the standard deviation also increases considerably to 3.30s, indicating that page-load times become more erratic due to the presence of the large download from C2. The upper plot also clearly shows a notable reduction in rate for C1 when C2 starts downloading.

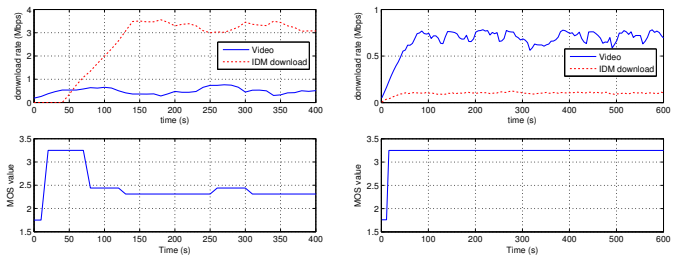
In the second experiment, again demonstrating performance in today's best-effort networks, we have a YouTube video stream playing continuously on client C1, and client C2 commences downloading a large file 175s into the experiment. Fig. 4(b) shows the video rate (upper plot) and the video mean-opinion-score or MOS (lower plot) as a function of time. It is clear that prior to the download starting, the video has a perfect MOS of 3.25 (the MOS continues to be high for a few more seconds after the download commences due to playback buffers), and thereafter drops to 2.32 for the duration of the download due to a reduction in the rate it can get from sharing the access link with the download. This clearly demonstrates how a user can be frustrated by poor QoE when downloads interfere with their video viewing.

We now employ our solution to show how the user can address this problem. Using the GUI, the user now specifies the two devices C1 and C2 with priority scores of 8 and 2 respectively. Consequently, clients C1 and C2 are ensured at least 80% and 20% of the access link capacity of 10 Mbps respectively (note that no application-level slicing is specified at the moment). The GUI then makes an API call into the ISP network via a JSON message to the network controller: {hello: jukebox, type: minbw, numQueue: 2, qid:1, nwsrc: 0.0.0.0/0, nwdst: 149.171.37.162/32, bw: 8000, qid:2, nwsrc: 0.0.0.0/0, nwdst: 149.171.37.115/32, bw: 2000}. Note that by specifying the source to be a wildcard, the API is in effect allocating a queue and minimum bandwidth on a per-client basis.



(a) Best-effort (b) Sliced

Fig. 6. Effect of iperf cross traffic on video QoE over: (a) best-effort quality, and (b) sliced, home network.



(a) Best-effort (b) Sliced

Fig. 7. Effect of IDM download on video QoE over: (a) best-effort quality, and (b) sliced, home network.

We now repeat the above two experiments with slicing enabled. Fig. 5(a) shows that with client C1 browsing and C2 starting a large download at time 160s, the average web-page load time remains relatively unaffected at around 6.63s with standard deviation of 1.44s. Fig. 5(b) shows that when client C1 is viewing video and client C2 commences a download (At time 60s), the video MOS for C1 remains stable at 3.25 (the initial low score is due to playback buffer ramp-up). Needless to say, this QoE improvement comes at the cost of slowing down the large download: in our experiments the large-download rate reduced from the range [3.13, 3.25] Mbps in best-effort queueing to the range [800, 920] Kbps with isolated queues performing slicing. This is expected and indeed desired, since the improvement in user QoE for video streaming and web browsing is well worth the elongation in download transfer time for non-critical traffic such as software updates.

B. *Application-level Slicing*

We now illustrate the ability of our tool to allow the user to perform application-level slicing for any chosen device. This is useful when the device may be performing several applications at the same time, e.g. downloading updates while also viewing videos or web-browsing. In Fig. 6 we generate a synthetic mix of TCP and UDP traffic using iPerf to demonstrate how slicing can protect video viewing. Fig. 6(a) lower plot shows that YouTube video on client C1 gets a perfect MOS of 3.25 till about 300s, at which point the iPerf UDP cross-traffic commences and clogs the link (red line in upper plot), and degrades video MOS to the range 1.75-2.5, making it unwatchable. To overcome this problem, the can open up our home network management GUI, and configure client C1 with an application slice for YouTube of 80%. The GUI calls the ISP API, resulting in creation of a separate queue for YouTube traffic (configured in our experiment as the IP block corresponding to the YouTube servers in Australia) with

8 Mbps minimum bandwidth allocation. Fig. 6(b) shows that with this configuration, the video MOS is now perfect at 3.25 without being affected by the iPerf TCP/UDP cross-traffic, while the UDP rate is suppressed to 600 Kbps.

We now repeat the above experiment with real rather than synthetic traffic. We install IDM, which is an aggressive download accelerator, on C1. Fig. 7(a) shows that the user is watching video, with perfect MOS of 3.25, till about time 40s, at which point a series of downloads are started. IDM accelerates these downloads by opening several parallel TCP sessions, rapidly ramping up its bandwidth as shown by the red curve in the upper plot. Consequently, YouTube video quality suffers, with MOS falling to below 2.5. When the user configures application specific slice for YouTube using our GUI, the resulting API call to the ISP network creates a separate queue for YouTube traffic, protecting video quality and maintaining a perfect MOS of 3.25 as shown in Fig. 7(b). Though one can argue that the user could have stopped or paused their downloads while streaming video, this requires manual action on the part of the user each time they run multiple applications; by contrast our solution allows a user to specify an application priority or slice once using the GUI, after which it takes effect each time they are performing multiple actions.

VI. Related Work

The idea of using SDN to virtualize network infrastructure for improving service quality is not new. In [9] it is suggested that the network controller detect sensitive priority flows via header inspection and then apply a rate-limiter. The NANDO framework [10] virtualizes the access network to allow multiple service providers to share infrastructure, and consumers to select the network operator to use for each service (e.g. video, voice, or data) based on QoS requirements. Home networks are sliced in [11], allowing multiple providers of services, such as smart grid metering, network management and video content providers, to share common home network infrastructure. The extension in [12] gives the home user control of how their network is sliced. Unlike many of these prior works that consider slicing to accommodate multiple service providers per household, our focus is on device or application-level slicing over the access link served by one ISP.

The work that comes closest to our is the PANE framework [13], developed in parallel to ours, that allows end-users and their applications to participate in network configuration through programmable interfaces. While their work develops APIs that can directly be incorporated into applications, our approach is different, in that it develops a GUI and corresponding API through which the user can adjust their home network performance over a slower time-scale, while continuing to use legacy devices and applications.

Other works related to the "parental control" like feature we propose include [14], that facilitates home network security management using a sliced architecture in which third party controller collects home traffic information in a centralised way, and HNDR [15] that addresses the security, performance and troubleshooting concerns of home user by an off-site controller monitoring the network activity.

VII. Conclusions

As household devices and applications proliferate and home networks become more complex, quality of experience is becoming more important, both for content providers like Google and Amazon, and for ISPs who want to reduce customer churn. This paper is an attempt to encourage ISPs to tap into the service quality dimension for differentiation and new monetization opportunities. We have shown that it is possible to develop a simple and easy-to-use GUI by which the user can prioritise their devices and applications. The GUI translates these high-level requirements and communicates them to the ISP via low-level APIs, who can then dynamically slice the access link as per the user's wishes to maintain QoE. We prototyped our system by building a Java-based GUI, implementing the API on a FloodLight SDN controller, and writing client scripts that generate real user traffic. Our experiments show that use of our tool lets users improve their web-browsing and video-streaming performance by stretching the non-time-critical traffic. We believe our tool is the first step towards more sophisticated and comprehensive home network management tools that include features for QoE, security, and much more.

References

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

[2] S. Barros and J. Beguiristain, "Capitalizing on Customer Experience," ERICSSON, White Paper, Sep. 2012.

[3] 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.

[4] Internet Society. (2012) Bandwidth Management: Internet Society Technology Roundtable Series. http://goo.gl/V3qYl.

[5] Cisco Systems, "Service Control EasyApp: Measuring Quality of Experience," Cisco Systems, White Paper, 2010.

[6] Nicira Networks. Open vSwitch. http://openvswitch.org/.

[7] Big Switch Networks. Project Floodlight. http://www.projectfloodlight.org/.

[8] 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.

[9] W. Kim, P. Sharma, J. Lee, S. Banerjee, J. Tourrilhes, S.-J. Lee, and P. Yalagandula, "Automated and Scalable QoS Control for Network Convergence," in *Proc. of USENIX INM/WREN*, San Jose, CA, Apr. 2010.

[10] J. Matias, E. Jacob, N. Katti, and J. Astorga, "Towards neutrality in access networks: A NANDO deployment with openflow," in *Proc. of IARIA International Conference on Access Networks*, Luxembourg, Jun. 2011.

[11] Y. Yiakoumis, K.-K. Yap, S. Katti, G. Parulkar, and N. McKeown, "Slicing home networks," in *Proc. of ACM SIGCOMM workshop on HomeNets*, Toronto, Ontario, Canada, Aug. 2011.

[12] Y. Yiakoumis, S. Katti, T.-Y. Huang, N. McKeown, K.-K. Yap, and R. Johari, "Putting home users in charge of their network," in *Proceedings of ACM UbiComp*, New York, NY, Sep. 2012.

[13] A. Ferguson, A. Guha, J. Place, R. Fonseca, and S. Krishnamurthi, "Participatory Networking: An API for Application Control of SDNs," in *Proc. SIGCOMM*, Hong Kong, Aug 2013.

[14] N. Feamster, "Outsourcing Home Network Security," in *Proc. ACM SIGCOMM HomeNets*, New Delhi, India, Sep. 2010.

[15] K. L. Calvert, W. K. Edwards, N. Feamster, R. E. Grinter, Y. Deng, and X. Zhou, "Instrumenting Home Networks," *SIGCOMM CCR*, vol. 41, no. 1, pp. 84–89, Jan. 2011.