Vegas over Access Point: Making Room for Thin Client Game Systems in a Wireless Home

Armir Bujari, Michele Massaro and Claudio E. Palazzi *Member, IEEE*
Department of Mathematics
University of Padua, Padua, Italy
Email: {abujari,mmassaro,cpalazzi}@math.unipd.it

***Abstract*—Cloud-based entertainment is gaining momentum. With the cost of commodity hardware lowering by the day and the consumer market penetration of in-house digital entertainment systems, a new generation of interactive services has come into being. This new technological wave has launched new content producers and providers which are rapidly adapting to match the consumer demand. In this context, thin client or cloud based gaming is attracting much attention, shifting the computational burden to the cloud while the consumer enjoys a fat video feed accessed through its thin client via the shared wireless gateway. However, this model of interaction raises new challenges that demand for specific networking solutions aimed at addressing the heterogeneous flow coexistence problem at the home wireless gateway. We propose a solution to this problem by devising a TCP Vegas-like congestion control algorithm deployed on top of the home gateway. Our solution works out of the box with the standard protocols at server, router and client level, thereby making deployment straightforward. Experimental assessment with real traffic traces shows that our solution addresses the problem effectively.**

***Index Terms*—Cloud gaming, interactivity, TCP Vegas, thin client game, wireless.**

I. Introduction

HOME entertainment is under rapid evolution. Today, with better access to broadband, the advent of digital home entertainment systems and online computing services, many of us get the bulk of their content streamed directly to our devices from the cloud. This has given spur to a two-way interaction model and personalized content, enabling new intelligent consumer services. The emergent cloud gaming paradigm, also referred to as thin client gaming, is one representative example in which the game engine resides in the provider's cloud and the outcome is streamed directly as video content to the consumer devices to be accessed through a thin client (e.g., OnLive [1]). This allows consumers to access the game without a console, making the end-device unimportant as the computing burden is shifted to the cloud [2]. Figure 1 depicts this new interaction model and emphasises the actors in play: on one side (left) is shown the home environment with the home gateway providing service access to a thin client console playing out a video stream, while on the right side is the cloud gaming system that collects the users actions and then renders, decodes and streams back the result.

Manuscript received October 12, 2014; revised October 12, 2014. Corresponding author: C. E. Palazzi (email: cpalazzi@math.unipd.it).

However, despite this powerful interaction model, cloud gaming remains in its early stages due to significant theoretical and practical challenges that inhibit its widespread deployment [3]. While the cloud computational capacity is growing by the day, the scalability of cloud gaming systems considering the stringent Quality of Experience (QoE) requirements is still an issue. Solutions have been proposed to tackle the QoE requirements producing intelligent user provisioning schemes, thus reducing the response delay on the transit network [4].

In this context, another crucial component providing us with shared access to the outside world is the wireless gateway, which brings the entertainment to the user's home, while the transport protocols remain those that have been in use for the last 30 years: the Transmission Control Protocol (TCP) for elastic (e.g., downloading) applications and the User Datagram Protocol (UDP) for real-time ones (e.g., video/music streaming, online gaming).

We consider the home gateway a potential bottleneck which, if not dealt properly, could hinder the gaming experience. The problem resides in the contrasting *modus operandi* of the underlying transport protocols being employed by the applications sharing the same wireless bottleneck. Indeed, applications with real-time requirements such as cloud gaming require fast delivery of gaming events and updates, whereas high data rates are desirable for elastic, download sessions. In contrast to classic gaming systems, cloud based games pose additional bandwidth requirements on the shared wireless downlink, with fat video chunks being streamed down to the device from the cloud [2], [5]. This further exacerbates the TCP *vs* UDP quarrel and, in particular, it jeopardizes the interactivity and responsiveness of gaming sessions, with TCP's aggressive behavior being the main cause [6].

The technical motivations and the conflicts that arise are explained in detail in Section II; yet, in synthesis, the fact is that elastic traffic is still considered to be the main traffic type run by consumers. In this context, providing reliable delivery and high throughput to this kind of applications: i) TCP and its congestion control functionality are employed at the transport layer, and ii) buffers and local retransmissions are extensively used at the MAC layer. However, these mechanisms in conjunction have been demonstrated to harm real-time applications by increasing the per-packet delivery latency.

To address this issue we could rely on delay-based TCP congestion protocols, employing packet Round Trip Time (RTT) rather than losses to prevent congestion. In this realm,

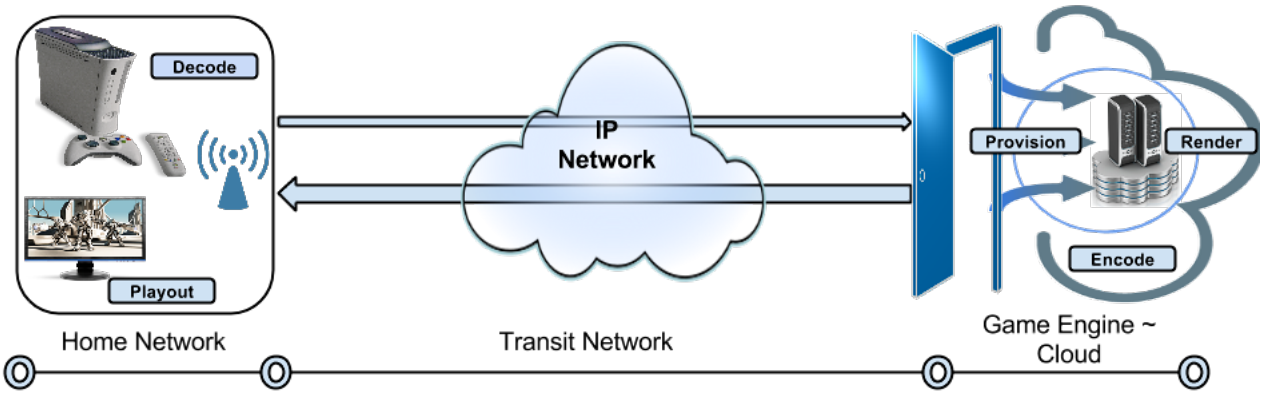


Fig. 1: Cloud gaming scenario.

TCP Vegas is the main candidate [7]. However, its adoption is hindered by the widespread deployment and use of loss-based protocols. The only way to employ TCP Vegas in practice would be to completely dismiss all loss-based TCP versions and, needless to say, this is not a feasible option.

In this article, we show how to practically exploit the benefits of delay-based protocols, by intervening only at the wireless gateway, thereby guaranteeing a factual deploy of the solution. In essence, the gateway is enhanced with an algorithm that automatically limits TCP flows through their advertised window when the channel is near saturation, thus avoiding long queues (and queuing delay) as well as packet loss, while keeping the TCP-based flows at a data rate that corresponds to a full utilization of the available bandwidth. At a high level, this approach is similar to TCP Vegas, even though the algorithms clearly have significant differences in employment, hence the name Vegas over Access Point (VoAP). As a result of our proposal, elastic applications achieve high data throughput, while real-time, gaming sessions maintain a low latency.

The rest of the paper is organized as follows. Section II discusses the technical background and issues at the basis of this work, while Section III overviews related work in scientific literature. VoAP is presented in Section IV, the experimental testbed is described in Section V and obtained results are analysed in Section VI. Finally, Section VII concludes this paper.

II. Technical Background and Problem Statement

Measurements on a real OC48 link show that the available capacity in the Internet core is generally larger than the aggregate utilized by transiting flows [8]. We can hence assume that the bottleneck is located at the edge of the path connecting a sender and a receiver: i.e., the DSL link connecting the house to the Internet or the domestic Wi-Fi. Therefore, the edge of the connection is where we have to act if we want to address congestion related problems.

Although the Internet has steadily evolved since the aforementioned study was undertaken, we deem this claim is still true nowadays. For this we refer the reader to studies like [2] where it is indirectly shown that the cloud gaming service adapts the downstream data rate depending on the constraints imposed on the network access device, thus identifying the last link as the bottleneck of the whole connection.

Applications can be grouped into two main classes depending on which protocol they use at the transport layer: TCP or UDP. The former is a protocol guaranteeing reliable and in-order delivery of data packets. These are desirable features which are exploited by elastic applications, those involving content download/transmission. A very important feature of TCP is its congestion control functionality. Through it, every TCP flow probes the link with higher and higher data rates eventually filling up the channel. At this point, packets are queued at the buffer associated with the bottleneck link until it overflows, causing packet losses. TCP retransmits the lost packets and halves its sending rate to diminish the congestion level. Finally, the regular increase of the sending rate is reestablished and the same process is repeated.

With UDP, packets are immediately sent toward the receiver with a data rate determined by the sender. UDP does not guarantee reliable and ordered delivery of packets. For this reason, UDP is usually employed by applications characterized by stringent real-time constraints that can tolerate sporadic packet losses (i.e., audio/video streaming, online games). The lack of congestion control functionalities of UDP has lead the scientific community to consider UDP as unfair toward TCP. Indeed, citing from [9]: *"Although commonly done today, running multimedia applications over UDP is controversial. [...] Thus, the lack of congestion control in UDP can result in high loss rates between a UDP sender and receiver, and the crowding out of TCP sessions - a potentially serious problem."*

Even if this may have been true several years ago, when the available bandwidth was much smaller than today, the broadband connectivity offered today tends to overturn this situation. Indeed, nowadays, larger and larger bandwidth is available at the network edges and the traffic generated by UDP-based applications (e.g., VoIP and classic online games) can generally be accommodated [10]. Even when considering bandwidth demanding UDP-based applications (e.g., real-time video streaming and thin client games) some flow control is operated at the application layer, decreasing the content quality and the required bandwidth for transmitting the media [2], [11], [12]. Yet, a problem emerges when downloading (TCP-based) applications coexist with real-time (UDP-based) ones,

the former forcing the latter to experience a scattered flow progression [13], [15]. The main cause for this problem can be found in the TCP congestion control functionality which continuously probes for higher transfer rates, queuing packets on the buffer associated with the bottleneck of the connection. If one considers that the same connection might be shared by several devices and applications thus increasing the congestion level and queue lengths, it is even more evident how packets can be delayed in queue, jeopardizing the interactivity requirements of real-time applications.

This negative situation is further worsened by factors linked to the wireless channel present at home gateways. First, the wireless medium allows the transmission of only one packet at a time and is not full-duplex as wired links. Second, as interference, errors, fading, and mobility may cause packet loss, the IEEE 802.11 MAC layer reacts through local retransmissions which, in turn, cause subsequent packets to wait in queue until the preceding ones or their retransmissions eventually reach the receiver. Last but not least, the back-off mechanism of the IEEE 802.11 introduces an increasing amount of time before attempting again a transmission.

Under these conditions, delay increments of online game packets can hit also tens of milliseconds, representing a huge waste of time when trying to deliver real-time information for entertainment services. As a reference benchmark, it is common belief that transmission delays of interactive online games should be inferior to 100 ms, with a maximum endurable value of 150 ms [3], [16].

A recent comprehensive study analysing the network traffic of the well-known thin client game system OnLive reports downstream bitrates similar to high-definition live video with frequent packets being streamed to the client, while on the other side, upstream traffic has lower data rates, comparable with classic online game upstream traffic [2]. This in- and outflow characteristics, if not dealt with properly, could worsen the game interactivity and make the session unbearable.

III. Related Works

Aimed at providing Quality of Service (QoS) capabilities to WLANs, the IEEE 802.11e has been proposed [17]. Its design allows to discriminate among various kinds of flows, assigning priorities through specific parameter settings. In particular, flows with different priorities are enqueued into different buffers, each having a specific contention time with those belonging to high priority traffic having higher chances to be transmitted before low priority ones. Yet, it is not clear how the access point (AP) could classify the incoming traffic. This also, requires the sender to mark each of its packets or flows with a priority level; unfortunately, this would imply the modification of all senders, thus strongly affecting the factual deployment of this protocol.

Solutions have been proposed to avoid the end-to-end latency increase generated by the loss-based TCP congestion control. As most of the problem resides in the protocol itself a possible approach could be to switch to a delay-based protocol. Indeed TCP Vegas is able to detect the congestion in advance using the RTT fluctuation of the packets. Essentially, it increases the transfer speed when the delay is under an α threshold and decreases it when the delay is over β ($\alpha < \beta$). This way, it is not necessary to lose a packet in order to detect congestion, rather it can be detected before it happens. As a desirable side effect, this also avoids the creation of bottleneck queues and corresponding queuing delays that would harm any interactive, real-time application. Unfortunately, TCP Vegas flows are penalized by loss-based ones, e.g., TCP New Reno and other real life TCP variants; loss-based flows generate congestion by their nature, which induces delay-based ones to slow down their transfer rate. The result is that loss-based flows will capture almost all the channel [7], [14], [18]. This incompatibility with legacy protocols made TCP Vegas not applicable.

Active Queue Management (AQM) mechanisms are a class of solutions that tackle the end-to-end latency problem on the Internet. These mechanisms require the deployment of specialized algorithms in the network path whose aim is to prevent buffers from sustaining long standing queues while also break the synchronization among flows. We can distinguish two main approaches to achieve this; they differ on the criteria used to compute the packet drop policy: buffer size [19] and delay-based [20]. The study undertaken in [20] is to the best of our knowledge the only study comparing these algorithms in the scenario of the AP bottleneck. The authors, found out that these mechanisms made TCPs feedback loop exceedingly long, leading to excessive end-to-end delays for the uplink traffic. In addition, a probabilistic or delay-based drop policy hinders TCP flows sharing the same bottleneck with the UDP ones.

On the other side, AQM could be exploited in conjunction with the Explicit Congestion Notification (ECN) mechanism available in TCP, which is able to expose congestion events to the endpoints and remedy to the issues raised by probabilistic dropping; however, as the study in [21] reports, ECN is not yet mature on the current Internet.

TCP Friendly Rate Control (TFRC) and its enhancement Mul-TFRC are TCP-friendly rate control protocols based on TCP Reno's throughput equation designed for multimedia flow transport against competing TCP flows. However, both proposals are not reactive to delay fluctuations and the way the sending rate is adjusted is expected to be more aggressive than mechanisms such as TCP Vegas [22].

A lightweight solution that does not require any modification of Internet protocols or network support was proposed in [14], [23] and named Smart Access Point with Low Advertised Window (SAP-LAW). Basically, this solution leverages on the ability of the AP to monitor all traffic passing through and, by appropriate on-the-fly modifications of the advertised window, prevents TCP-based flows from exceeding their fair bandwidth share. This avoids the classic TCP's congestion window fluctuations, stabilizing it to an appropriately computed value, which does not decrease the achieved throughput. Moreover, it avoids queue formation, hence alleviating delay problems for UDP packets and allowing the user to enjoy interactive, real-time services. However, this proposal requires an *a priori* knowledge of the bottleneck capacity and the RTT of each flow; these parameters may not be always known or

correctly computed or even stable enough to be used in a timely fashion.

Gateway Adaptive Pacing (GAP) proposed for multi-hop wireless networks is similar in spirit [24], [25]. GAP adds an artificial delay between packets based on continuous measurements of the network. In this way the packet interarrival time decreases, but so does even the throughput of TCP flows, thus finding scarce applicability in our considered scenario and, in general, in the regular one-hop case.

Studies on thin client game systems so far have been focused on traffic flow characteristics and system architecture aimed at delivering the service [2], [26]. To the best of our knowledge this is the first work addressing the problem of TCP/UDP flow coexistence in presence of concurrent cloud-based gaming flows.

IV. VoAP

In the classic TCP protocol, the actual sending rate (i.e., the sending window) of a TCP flow is determined as the minimum between the congestion window (continuously recomputed by the sender) and the advertised window (provided by the receiver via returning ACK packets). Our idea is to dynamically modify the advertised window to limit the growth of the TCP flow's sending rate. Indeed, a good tradeoff solution between throughput and low delays could be achieved by maintaining the sending rate of the TCP flows high enough to efficiently utilize the available bandwidth and, at the same time, limited in its growth so as to not overutilize buffers. Intuitively, this way the per-packet delays are reduced by the absence of queues along the route from the sender to the receiver, while the throughput is kept high by the absence of packet losses that would otherwise halve the congestion window. At the same time, we exploit an existing feature of regular TCP implementations thus limiting the modifications required only to the home AP.

The advertised window is generally imposed by the receiver; however, in our case, VoAP computes its value. We decide to install VoAP on the AP as it represents the bottleneck of the connection and all communication should pass through this device. In essence, VoAP monitors the ongoing traffic and the current queuing delay experienced by packets transiting through its buffer. If this queuing delay becomes too long to sustain interactive applications such as games, VoAP proceeds with appropriate on-the-fly modifications of the advertised window in TCP ACKs so as to limit TCP flows just below the congestion level.

In the following we discuss in detail how VoAP operates in order to determine an appropriate advertised window for the TCP flows that pass through the AP. Our algorithm addresses the aforementioned coexistence problem among heterogeneous applications, while avoiding impractical assumptions that could hinder its deployment. Below, we enumerate the desirable properties that our target algorithm should adhere to.

Property 1: It does not require any change to current network protocols, servers, routers and clients (only the AP is modified).

Property 2: It does not require any *a priori* information about the network (e.g., the capacity of the bottleneck link or flows' RTTs).

Property 3: UDP flows do not have to suffer from per-packet latency caused by TCP.

Property 4: TCP throughput must not be sacrificed.

Property 5: Congestion must be detected as soon as possible.

Property 6: Every TCP flow must be able to obtain a fair share of the available bandwidth.

Property 7: If a TCP flow does not exploit the entire the bandwidth assigned by the algorithm, the leftover must be redistributed among the other flows.

Our solution has been designed mainly for a wireless home scenario, where the bottleneck is represented by a wireless component; however, it should work even in a more general environment (including wired connectivity). Through the rest of the section we discuss how each of the properties above is ensured by VoAP. Where applicable, we also provide a pseudocode explanation of the algorithm designed to specifically ensure a certain property or behaviour.

Property 1) To satisfy the first property, we have taken inspiration from SAP-LAW, thus placing our solution in the AP. From this position we can monitor and act upon individual flows; at the same time, this strategy alleviates us the burden to perpetuate any client side modifications. Using the advertised window already present in the header of a TCP ACK, it is possible to limit the amount of transiting packets and hence control the utilization of the buffers.

Property 2) The main idea at the basis of VoAP is to monitor all TCP flows passing through the AP and measure how long packets are queued before actually being transmitted through the wireless channel. This is the only information required and it is easily available at the AP. No modification is needed at the server, router or client side to enable VoAP. Distributing the changes as a firmware update or other means on the AP-side augments the chances of an actual deployment.

Property 3) Algorithm 1 shows how the information about packet queueing time is exploited by VoAP in order to determine the appropriate value for the advertised window of TCP flows. In particular, we distinguish three cases. If the measured delay $IFQmaxDel$ is below the threshold α (line 2), the channel can be considered almost free, so the advertised window is increased, allowing the TCP to increase its transfer speed. A queuing delay between the two thresholds α and β (line 12) means that the network is well utilized but not congested; for this reason the flows are not acted upon. If the delay is over the β threshold (line 20), the channel is saturated, so the TCP flows are slowed down by decrementing their advertised windows communicated though the ACKs. In Algorithm 1, the variable $VoAPactive$ indicates whether VoAP's capability to limit TCP flows through their advertised window is active or not; the rationale behind this feature is explained when discussing **Property 4**.

The parameters α and β are expressed in milliseconds and represent the queuing delay boundaries we would like to have in our system. In particular, β is the ideal maximum queuing delay at the AP. Considering that any game packet should be delivered within 150 ms at most and that this accounts for all

delays (transmission, propagation, handling, queuing, etc.), we decided to set β equal to 15 ms [3], [16]. Instead, the value used for α is set to 5 ms and the explanation is provided when discussing **Property 4**.

The queuing delays are measured during an interval of time T (200 ms in our experimentation), after which Algorithm 1 is executed. Every modification to the advertised window is done on a per unit basis to ensure a smooth fluctuation of the TCP flows and throughput. Moreover, these modifications are performed only after T elapses. After this, the timers are reset and the delay sampling restarts.

Inside a timeframe T each flow is monitored if it is active or not, and the bandwidth is split accordingly among flows in a fair way. It is therefore necessary for T to be big enough to allow for all active flows to transmit at least one packet, but also small enough not to waste the bandwidth on a non-active flow.

In Section VI-A we report the experimental results, which demonstrate that VoAP correctly and effectively enforces this property, maintaining a low per-packet delivery delay.

Property 4) VoAP should be able to keep the data flowing through the channel while at the same time avoid the negative effects of per-packet latency. The choice for α and β values relate to how many packets can simultaneously reside in the bottleneck queue which has a direct impact on both the per-packet latency and the throughput. We discussed the β value (15 ms) as the ideal maximum delay we would like to tolerate as queuing delay, while parameter α represents a minimum queuing delay that we would actually like to have at the AP as it ensures that the AP's buffer has a set of packets ready to be transmitted at any time and no transmitting opportunity will be wasted.

From the experiments that we have made by varying α between 0 and β, the former parameter had no big impact on the performance, thus making a possible optimization study not strictly necessary. We have hence used α equal to 5 ms for the experimentation presented in Section VI as a representative value which works well (as well as other values between 0 ms and β). This is shown in Section VI-B.

VoAP should not limit the slow start phase to a linear increment. To this end, we introduce two features to our solution:

- VoAP's algorithm is not used if the congestion window of a TCP flow is in slow start (the value of $VoAPactive$ used in Algorithm 1 will be $false$);
- VoAP's algorithm acts on the TCP flow only after the β threshold is exceeded for the first time (lines 23 and 24, Algorithm 1).

To avoid ambiguities regarding the slow start phase detection, in the experimentation part what does count is the increment rate of the TCP. However, this component is exploited only at the initial stages of the algorithm thus having a very limited impact on the final outcome. The protocols perform in a similar way even when this feature is not considered and the difference in orders of magnitude between the other protocols remain invariant. In Section VI-B we report on experimental results that demonstrate how VoAP does not harm TCP's throughput.

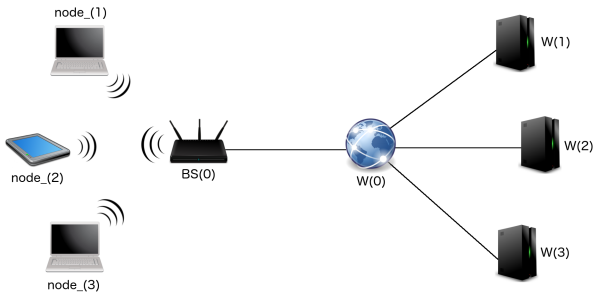


Fig. 2: The considered experimental scenario.

Property 5) When the queue is growing it is mandatory to act quickly so as to avoid losses. To fasten the delay detection, the queuing delay measurements are not only made when a packet leaves the queue, but also while a packet is still in the queue. To achieve this, we just need to read the timestamp of the first packet which is always the oldest in a FIFO queue. This procedure is described in Algorithm 2.

Property 6) A fair share of bandwidth is guaranteed for every TCP flow. To achieve this, the maximum number of outgoing packets is estimated as the sum of the advertised windows of every active TCP flow (variable $totalAwnd$ in Algorithm 1). Once the value is obtained, it is divided by the number of active flows, to assign a share of the bandwidth to each one, as can be seen in Algorithm 3. In Section VI-C we report the experimental results, which demonstrate that VoAP is fair toward concurrent TCP flows.

Property 7) Not all flows will actually try to fully exploit their assigned bandwidth share (e.g., because of small bandwidth available at their server). For this reason we provide a sharing function (Algorithm 4) to make the algorithm able to redistribute the bandwidth leftover.

If the congestion window of a flow is bigger than its advertised window, it means that it would be able to exploit a bigger share, so it is added to a list of candidates to receive additional bandwidth (lines 6 and 7, Algorithm 4) so as to avoid the underutilization of the channel. To ensure growing chances to those low-rate flows in case their conditions change, their limited share is oversized by 10%.

In Section VI-D we report the experimental results demonstrating that VoAP is efficient and redistributes unused bandwidth among TCP flows that are able to exploit it.

V. Experimental Assessment

In order to assess the goodness of our proposal we have chosen to run the simulations in a well-known and reliable simulation environment: Network Simulator 2 (NS2). Our intention is to reproduce a realistic scenario, complex enough to simulate a real home environment.

In the considered scenario (in Figure 2) there are wireless devices connected to the AP (where VoAP is deployed) and wired nodes that represent remote resources in the cloud. All the wired connections have a 100 Mbps capacity and the wireless channel is a IEEE 802.11g representing the bottleneck of the connection. The Wi-Fi AP is configured

Algorithm 1

Require: $alpha \leq beta$
1: $IFQmaxDel \leftarrow$ highest packet delay measured in the last period *T*
2: **if** $IFQmaxDel < alpha$ **then**
3: **for each** active flow *i* **do**
4: **if** $VoAPactive(i)$ **then**
5: $awnd(i) \leftarrow awnd(i) + 1$ {$awnd(i)$ represents the advertised window of the flow i}
6: $totalAwnd \leftarrow totalAwnd + awnd(i)$ {$totalAwnd$ represents the sum of all the advertise windows}
7: **else**
8: $VoAPactive(i) \leftarrow false$
9: **end if**
10: **end for**
11: **end if**
12: **if** $IFQmaxDel > alpha$ and $IFQmaxDel < beta$ **then**
13: **for each** active flow *i* **do**
14: **if** $VoAPactive(i)$ **then**
15: $awnd(i) \leftarrow cwnd(i)$
16: **end if**
17: $totalAwnd \leftarrow totalAwnd + awnd(i)$
18: **end for**
19: **end if**
20: **if** $IFQmaxDel > beta$ **then**
21: **for each** active flow *i* **do**
22: **if** $\neg VoAPactive(i)$ **then**
23: $VoAPactive \leftarrow true$
24: $awnd(i) \leftarrow cwnd(i)$
25: **end if**
26: $awnd(i) \leftarrow awnd(i) - 1$
27: $totalAwnd \leftarrow totalAwnd + awnd(i)$
28: **end for**
29: **end if**

Algorithm 2

Require: $Queue$ represents the AP's FIFO queue
1: $oldestInQueue \leftarrow Queue\text{->}\ FirstPacketTime()$ {Save the timestamp of the first packet in the Queue, that is also the oldest}
2: $now \leftarrow$ read current time
3: $delta \leftarrow now - oldestInQueue$
4: **if** $delta > IFQmaxDel$ **then**
5: $IFQmaxDel \leftarrow delta$
6: **end if**

Algorithm 3

1: $windowSplit \leftarrow totalAwnd/activeFlows$
2: **for each** active flow *i* **do**
3: **if** $VoAPactive(i)$ **then**
4: $awnd(i) \leftarrow (awnd(i) + windowSplit)/2$
5: **end if**
6: **end for**

Algorithm 4

1: $windowSplit \leftarrow totalAwnd/activeFlows$
2: **for each** active flow *i* **do**
3: **if** $VoAPactive(i)$ **then**
4: $diff = windowSplit - cwnd(i)$
5: **if** $diff < 0$ **then**
6: $needMoreBW[i] \leftarrow true$
7: $flowsInNeed \leftarrow flowsInNeed + 1$
8: $awnd(i) \leftarrow (awnd(i) + windowSplit)/2$
9: **else**
10: $diff = windowSplit - (cwin(i) + 10\%)$
11: **if** $diff < 0$ **then**
12: $awnd(i) \leftarrow (awnd(i) + windowSplit)/2$
13: **else**
14: $BWLeftover \leftarrow BWLeftover + diff$
15: $awnd(i) = cwnd(i) + 10\%$
16: **end if**
17: **end if**
18: **end if**
19: **end for**
20: **if** $BWLeftover > 0$ and $flowsInNeed > 0$ **then**
21: $split \leftarrow BWLeftover/flowsInNeed$
22: **for each** flow *i* **do**
23: **if** $needMoreBW[i]$ **then**
24: $awnd(i) \leftarrow windowSplit + split$
25: **end if**
26: **end for**
27: **end if**

with a buffer size of 250 packets in order to be coherent with real implementations [27]. The one way delay between the resources (servers) and clients is 40 ms and the flows we run in the scenario are those shown in the Table I.

To increase the reliability of our experiments we employ real traffic traces that mimic the considered applications. For instance, the video chat is based on the traffic generated by a real webcam based video chat. The gaming UDP flow employed corresponds to a thin client game system, thus requiring high data rates for the video stream from the server to the client and a thin game action stream from the client to the server. To reproduce a realistic scenario, we have adopted the traces gathered in [2], which have been collected using the OnLive thin client game system with different categories of games [1]. A characteristic of thin client game systems is the ability to adapt the quality of the streamed video feed based on network conditions. Among those reported in [2], we chose to employ the traces corresponding to flows whose data rate is best suited to our wireless home network scenario. Moreover, we consider two different game categories: first person shooter and strategic.

Among those analysed in [2] we adopt the *Unreal Tournament III* (UT) trace in its 10 Mbps limited downstream version and *Grand Ages: Rome* (Rome) in all its variants. More detailed information about the game flows and their characteristic are provided in Table II.

Before discussing the experimentation results, we point out that the approach employing an end-to-end TCP Vegas sce-

TABLE I: Network flows

Application	Protocol	From	To	Start/End (s)
Download (FTP)	TCP	W(3)	node_(3)	100/250
Thin Client - S to C	UDP	W(1)	node_(1)	1/250
Thin Client - C to S	UDP	node_(1)	W(1)	1/250
Video chat S to C	UDP	W(2)	node_(2)	50/250
Video chat C to S	UDP	node_(2)	W(2)	50/250

TABLE II: Thin client games

Name	Packet Size	Interarrival	Restrictions
UT	947 B	1.2 ms	10 Mbps downstream limited links
Rome	914 B	1.6 ms	Unlimited

nario has a performance trend similar to our VoAP. However, a Vegas-like approach is hindered by the coexistence with loss-based TCP variants sharing the same bottleneck. Instead, we contrast and provide an extended analysis of VoAP against popular TCP variants present in our devices.

VI. Results

We have discussed VoAP's desirable properties throughout Section IV. Some of these properties can actually be verified experimentally. To this end, in this section we assess through simulations **Properties 3, 4, 6, 7**. To avoid biasing our comparison towards one specific TCP version or the other, we have chosen to compare VoAP against three widely available TCP variants as noted in [28].

A. Demonstration of Property 3: Low Per-Packet Latency

One of the main purposes of VoAP is to reduce the negative effects of TCP over concurrent UDP-based real-time applications, which in our scenario translates to cloud based gaming session. In particular we want to avoid the latency peaks caused by the TCP congestion control algorithm, as stated by **Property 3** discussed in Section IV.

Figure 3 compares VoAP against three widely used TCP variants. From Figure 3a it can be seen that TCP New Reno has a stability zone at interval 20 to 80 with a highest peak amounting at 121 ms while Bic (Figure 3b) and Cubic (Figure 3c) stabilize at intervals [30, 85], [60, 80] with the highest peak arriving at 92 ms, 82 ms respectively. Instead with VoAP, the picture drastically changes and the outcome is shown in Figure 3d, with the highest peak reaching a 18 ms delay, showing an improvement of 85% when compared to New Reno and 80%, 78% with Bic and Cubic respectively.

B. Demonstration of Property 4: No Sacrifice of Throughput

The previous results would not be satisfactory if we have to sacrifice TCP in terms of throughput. We also want to allow the TCP flow to exploit the same bandwidth that it would have reached without the use of VoAP.

To this end, Figure 4 shows the bandwidth trend of the TCP flow without enforcing VoAP. From the charts it can be seen how loss induces bandwidth fluctuations consistent with the peaks exhibited in Figure 3. Using VoAP instead, the absence of latency peaks in Figure 3d is reflected by a more stable bandwidth without any loss incurred, as shown in Figure 4d.

The difference between the throughput in the experiments is below the 1% threshold. We can state that the employment of VoAP does not harm the TCP flows, henceforth **Property 4** stated in Section IV is satisfied. This result is also confirmed when employing the Rome trace (Figures 5, 6).

Figure 7 shows a complementary view, reporting the cumulative distribution function (CDF) of the queuing delay for the UT and Rome scenarios, respectively. It is clear that VoAP outperforms the regular deployment in both scenarios, incurring less delays, reaching a near 100% per-packet delay at around 20 ms.

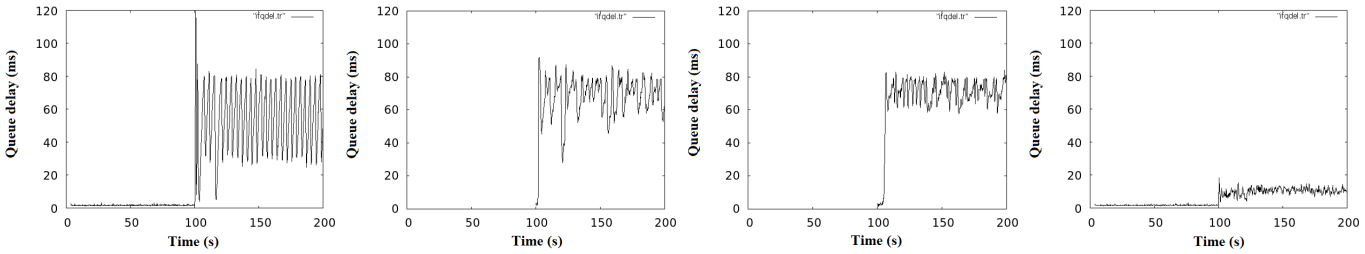
Furthermore, we studied the average throughput achieved by the TCP flow in the scenario with a concurrent UT game flow and VoAP employing different values for α in the interval 0 to 10 ms with a 1 ms step increment. The outcome is, as anticipated in Section IV, that almost any α value between 0 and β does not significantly impact on the performance. Thereby it does not necessarily require an optimization study. For completeness we report the statistical significance of the data obtained from the ten runs having a mode of 5.37 Mbps, lower peak at 5.17 Mbps, higher peak at 5.38 Mbps and a standard deviation of 0.097.

The trend of the charts shown until now regarding the TCP throughput is shaped by the use of VoAP on the AP. When considering a traditional scenario, the TCP throughput follows the well-known saw-tooth shaped fluctuations of the congestion window, which determines the actual sending window. Instead with VoAP, the sending window is shaped by the advertised window as the sending window is the minimum between the congestion window and the advertised window.

Therefore, to understand the connection behind the throughput and the per-packet delay we compare the congestion windows of regular TCP flows (Figure 8 for UT and Figure 9 for Rome) with the advertised windows generated by VoAP, which correspond to the factual sending windows in the two cases. As expected, with regular TCP and no VoAP enforced, we have a saw-tooth shaped fluctuation of the congestion window which generates queuing delays, congestion losses and throughput variations, whereas with VoAP we have a smoother progression of the advertised window which ensures a stable throughput and limits queuing delays.

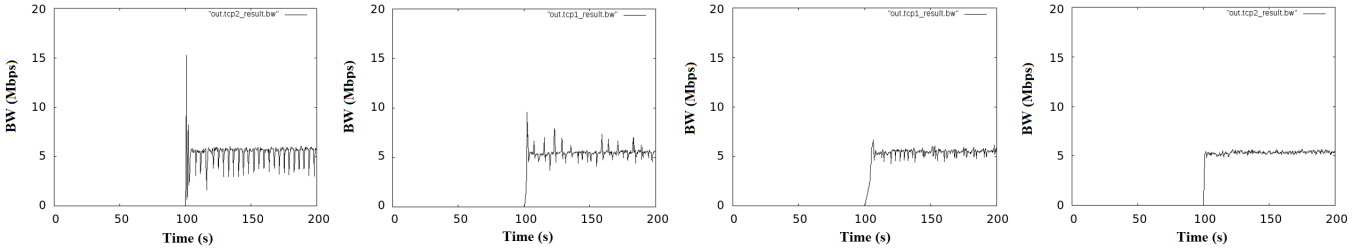
C. Demonstration of Property 6: Bandwidth Fair Share

In a common wireless home scenario there is certainly more than one TCP flow at a time. With **Property 6** discussed in Section IV we state that every flow should be able to achieve a fair share of the available bandwidth. We modified the scenario by running a TCP flow from the start and let two other flows enter the scenario after 70 s and 140 s respectively. The results are shown in Figure 10a, where it is clear how the flows' bandwidth is halved after the activation of the second flow, and is reduced to 1/3 of the original value when even the third flow is activated. The two charts corresponding to the other two flows show the same trend as they all fairly share the available bandwidth; we do not show them here to avoid redundancy and for lack of space.



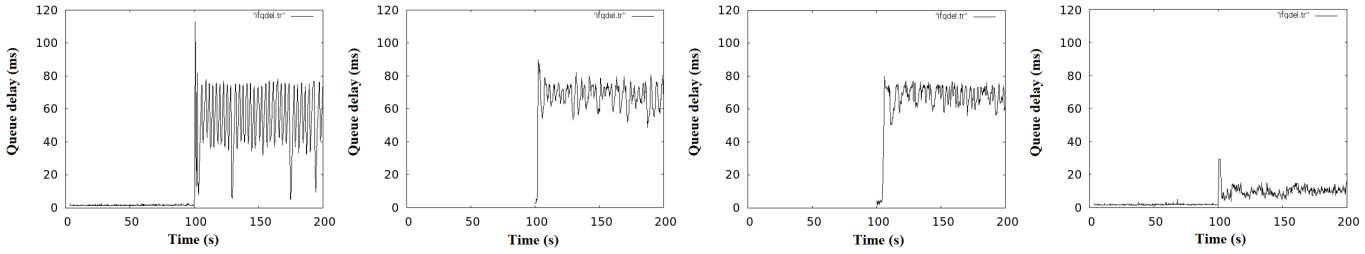
(a) New Reno (b) Bic (c) Cubic (d) VoAP

Fig. 3: Bottleneck queue delays in the UT OnLive scenario.



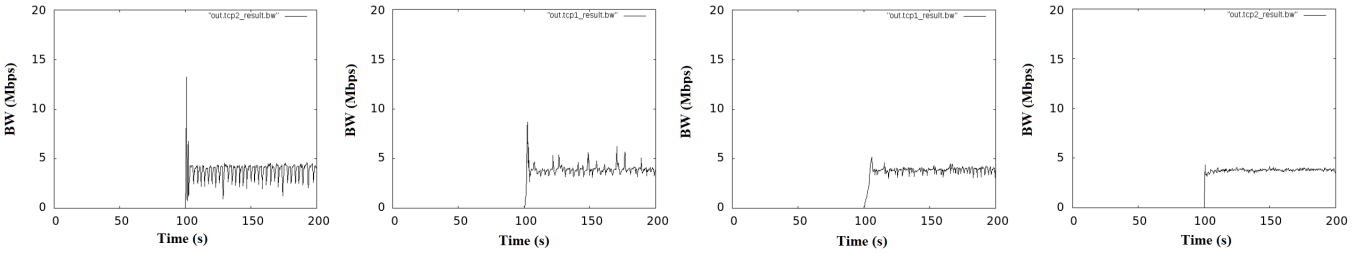
(a) New Reno (b) Bic (c) Cubic (d) VoAP

Fig. 4: Bandwidth comparison at the bottleneck in the UT OnLive scenario.



(a) New Reno (b) Bic (c) Cubic (d) VoAP

Fig. 5: Bottleneck queue delays in the Rome OnLive scenario.



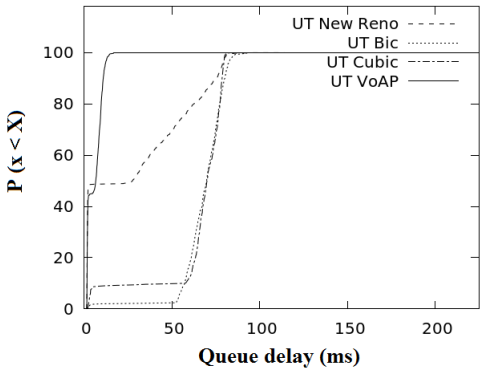
(a) New Reno (b) Bic (c) Cubic (d) VoAP

Fig. 6: Bandwidth comparison at the bottleneck in the Rome OnLive scenario.

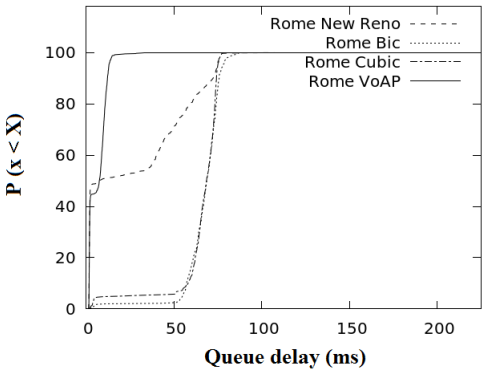
D. Demonstration of Property 7: Redistribution of Unexploited Bandwidth

As there might be some flows not fully exploiting their fair share of the available bandwidth, **Property 7** states that the possible leftover must be redistributed. To show that this property is satisfied we modify the scenario, replacing the TCP flow with the configuration detailed in Table III.

The chart depicted in Figure 10b shows how the bandwidth used by the first flow is halved when the channel is shared with a second, unrestricted flow (starting at 40 s and ending at 80 s), and how the leftover is reallocated to the first flow when a second flow does not fully exploit its fair share of the bandwidth (starting at 120 s and ending at 160 s).

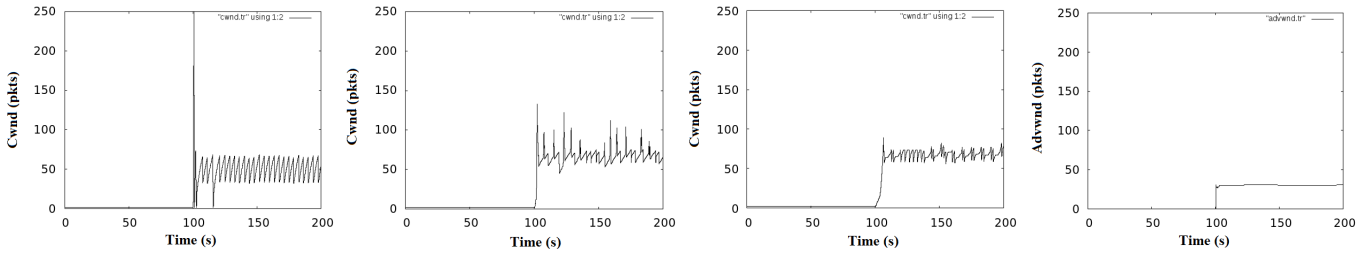


(a) UT



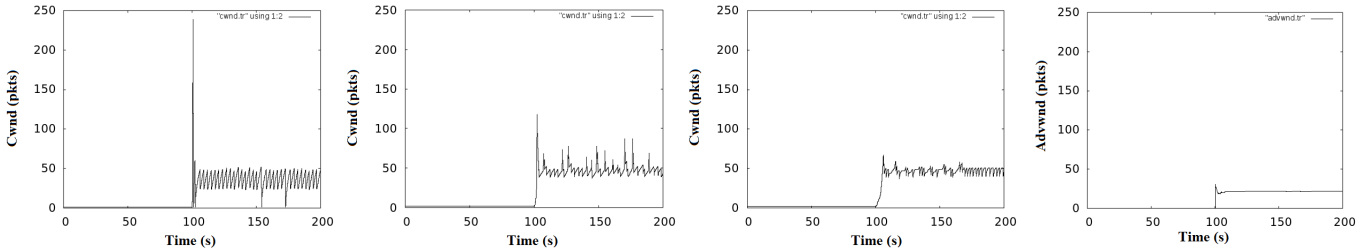
(b) Rome

Fig. 7: Queuing delay CDF.



(a) New Reno (b) Bic (c) Cubic (d) VoAP

Fig. 8: Sending window comparison with the UT OnLive flow.



(a) New Reno (b) Bic (c) Cubic (d) VoAP

Fig. 9: Sending window comparison with the Rome OnLive flow.

TABLE III: Heterogeneous TCP flow scenario

Start	Stop	Limitation
0 s	200 s	Unlimited
40 s	80 s	Unlimited
120 s	160 s	1 Mbps

E. VoAP's impact on the Upstream

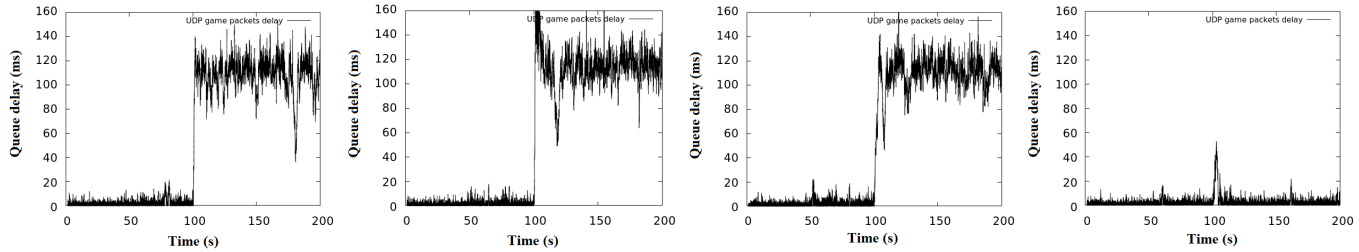
The above results clearly demonstrate that VoAP does help reduce per-packet latency while avoiding penalties in throughput. We now focus on the queuing delay in the upstream traffic. This is specifically important for the thin client game system scenario we are considering as a timely delivery of players actions has a direct impact on the responsiveness and hence on the QoE of the game play. We employ the same configuration as done before with the Rome uncontrolled scenario and the results are shown in Figure 11.

From the charts it is clear that VoAP performs better when compared to the other algorithms, with a smoother trend and less subject to oscillations. The spikes shown in the charts at around 100 ms are due to concurrent TCP traffic sharing the bottleneck: VoAP is less subject to this and able to maintain a stable regime, not introducing any delays in the packet delivery. A complementary view is given in Figure 12, which shows the CDF of the queuing delays, with VoAP able to keep nearly 98% of the game flow traffic under 12 ms of queuing time.

(a) Similar flows enter the channel

(b) Heterogeneous flows entering and leaving the channel

Fig. 10: VoAP bandwidth reallocation.



(a) New Reno (b) Bic (c) Cubic (d) VoAP

Fig. 11: Upstream bottleneck queue delays in the Rome OnLive scenario.

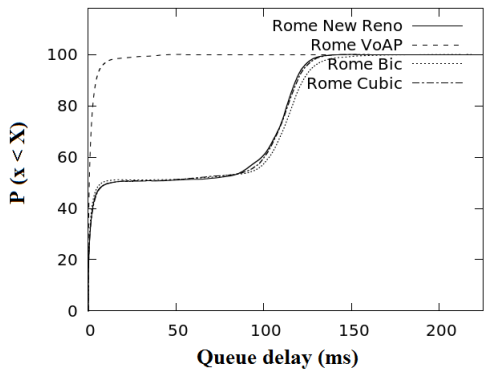


Fig. 12: Queuing delay CDF for the Rome upstream scenario.

VII. Conclusion

We considered a realistic scenario where in-home entertainment is delivered to wireless devices through a home gateway from cloud-based services. Our analysis focused on tackling the flow coexistence problem among concurrent transmissions of two kinds of different streams: TCP-based elastic (e.g., downloading) applications and UDP-based real-time (e.g., video streaming and online gaming) applications.

The contrasting *modus operandi* of the above transport protocols is further exacerbated by the downlink requirements posed by cloud based gaming, whose flow, if not properly accommodated, could be impaired by persistent TCP-based flows.

To solve this problem, we proposed the VoAP algorithm that consists of an augmented AP. Exhaustive experimental assessments employing real and synthetic flows show that our algorithm effectively allows high throughput and low per-packet delay, significantly improving the network performance and session interactivity.

As future work, we plan on building a VoAP testbed on the OpenWRT platform where we expect VoAP to still maintain its superiority. The testbed will allow us to make additional assesments of the protocol including but limited to its overhead in real environments.

References

[1] $http://www.onlive.com$ Accessed in 2015.
[2] M. Claypool, D. Finkel, A. Grant, M. Solano, "Thin to Win? Network Performance Analysis of the OnLive Thin Client Game System", in Proc. of the ACM Annual Workshop on Network and Systems Support for Games (NetGames), Venice, Italy, 2012.
[3] R. Shea, L. Jiangchuan, E. C.-H. Ngai, C. Yong, "Cloud gaming: architecture and performance," IEEE Network 27(4), 2013.
[4] D. Wu, Z. Xue, J. He, "iCloudAccess: Cost-Effective Streaming of Video Games From the Cloud With Low Latency," IEEE Transactions on Circuits and Systems for Video Technology 24(8), 2014.
[5] A. Kaiser, D. Maggiorini, N. Achir, K. Boussetta, "On the Objective Evaluation of Real-Time Networked Games," in Proc. of IEEE Global Telecommunications Conference (GLOBECOM), Honolulu, Hawaii, USA, 2009.
[6] C. E. Palazzi, G. Pau, M. Roccetti, M. Gerla, "In-Home Online Entertainment: Analyzing the Impact of the Wireless MAC-Transport Protocols Interference," in Proc. of the IEEE International Conference on Wireless Networks, Communications and Mobile Computing (WIRELESSCOM), Maui, HI, USA, 2005.

[7] S. Low, L. Peterson, L. Wang, "Understanding TCP Vegas: Theory and Practice", Princeton University, Princeton, New Jersey, 2000.
[8] H. Jiang, C. Dovrolis, "Why Is the Internet Traffic Bursty in Short (sub-RTT) Time Scales?", in Proc. of the ACM SIGMETRICS 2005, Banff, AL, Canada, 2005.
[9] J. F. Kurose, K. W. Ross, *Computer Networking: A Top-Down Approach Featuring the Internet*, 6th Ed, Addison Wesley Longman, Boston, MA, USA, 2012.
[10] M. Zhang, M. Dusi, W. John, C. Changjia, "Analysis of UDP Traffic Usage on Internet Backbone Links," in Proc. of International Symposium on Applications and the Internet (SAINT), Bellevue, WA, USA, 2009.
[11] A. Balk, M. Gerla, M. Sanadidi, D. Maggiorini, "Adaptive Video Streaming: Pre-encoded MPEG-4 with Bandwidth Scaling", Elsevier Computer Networks 44(4), 2004.
[12] I. Sodagar, "The MPEG-DASH Standard for Multimedia Streaming over the Internet". IEEE MultiMedia 18(4), 2011.
[13] M. Massaro, A. Bujari, C. E. Palazzi, "Exploiting TCP Vegas' Algorithm to Improve Real-Time Multimedia Applications", in Proc. of IEEE Consumer Communications and Networking Conference (CCNC), Las Vegas, NV, USA, Jan 2015.
[14] C. E. Palazzi, S. Ferretti, M. Roccetti, G. Pau, M. Gerla, "What's in That Magic Box? The Home Entertainment Center's Special Protocol Potion, Revealed", IEEE Transactions on Consumer Electronics 52(4), 2006.
[15] M. Gerla, D. Maggiorini, C. E. Palazzi, A. Bujari, "A Survey on Interactive Games over Mobile Networks", Wiley Wireless Communications and Mobile Computing 13(3), 2013.
[16] S. Choy, B. Wong, G. Simon, C. Rosenberg, "The Brewing Storm in Cloud Gaming: A Measurement Study on Cloud to End-User Latency," in Proc. of the IEEE Network and Systems Support for Games (NetGames), Venice, Italy, 2012.
[17] IEEE Standard for Information Technology, "Telecommunications and Information Exchange between Systems - Local and Metropolitan Area Networks - Specific Requirements Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications: Amendment: Medium Access Control (MAC) Quality of Service Enhancements", P802.11e/D13.0, 2005.
[18] L. S. Brakmo, L. L. Peterson, "TCP Vegas: End to End Congestion Avoidance on a Global Internet", IEEE Journal on Selected Areas in Communications 13(8), 2002.
[19] N. Kuhn, E. Lochin, O. Mehani, "Revisiting old Friends: Is CoDel really achieving what RED cannot?", in Proc. of SIGCOM Capacity Sharing Workshop, Chicago, USA, 2014.
[20] N. Khademi, D. Ros, M. Welzl, "The New AQM Kids on the Block: Much Ado About Nothing ?", Research Report 434, 2013.
[21] S. Bauer, R. Beverly, A. Berger, "Measuring the State of ECN Readiness in Servers, Clients, and Routers", in Proc. of ACM SIGCOMM Conference on Internet Measurement (IMC), NY, USA, 2011.
[22] D. Ros, M. Welzl, "Less-than-Best-Effort Service: A Survey of End-to-End Approaches," IEEE Communications Surveys & Tutorials 15(2), 2013
[23] C. E. Palazzi, N. Stievano, M. Roccetti, "A Smart Access Point Solution for Heterogeneous Flows", in Proc. of the International Conference on Ultra Modern Telecommunications and Workshops (ICUMT), St. Petersburg, Russia, 2009.
[24] S. M. El Rakabawy, A. Klemm, C. Lindemann, "Gateway Adaptive Pacing for TCP Across Multihop Wireless Networks and the Internet", in Proc. of the ACM International Symposium on Modeling Analysis and Simulation of Wireless and Mobile Systems (MSWiM), Torremolinos, Spain, 2006.
[25] K. Kim, D. S. Niculescu, S. Hong, "Coexistence of VoIP and TCP in Wireless Multihop Networks", IEEE Communications Magazine 47(6), 2009.
[26] Y.-C. Chang, P.-H. Tseng, K.-T. Chen, and C.-L. Lei, "Understanding the Performance of Thin-Client Gaming," in Proc. of IEEE Communications Quality and Reliability (CQR), Naples, FL, USA, 2011.
[27] T. Li, D. Leith, D. Malone, "Buffer Sizing for 802.11 Based Networks", IEEE/ACM Transactions on Networking 19(1), 2011.
[28] P. Yang, J. Shao, W. Luo, L. Xu, J. Deogun, L. Ying, "TCP Congestion Avoidance Algorithm Identification," IEEE/ACM Transactions on Networking 22(4), 2014.

Armir Bujari received his Ph.D. degree in Computer Science at the University of Padua, Italy, under the supervision of Professor Claudio E. Palazzi. He previously completed his M.S. in Computer Science, Summa Cum Laude, at the same university. His research interests include protocol design and analysis for wireless networks and delay-tolerant communication in mobile networks. On these topics, he is active in several technical program committees in international conferences and is co-author of more than 15 papers, published in international conferences proceedings, books, and journals.

Michele Massaro is a M.S. student in Computer Science and is currently completing his thesis in the MobileLab laboratory, Padua, under the supervision of Prof. Claudio Enrico Palazzi. His research interests include protocol design and analysis for wireless networks and cloud computing.

Claudio Enrico Palazzi is an Associate Professor with the Department of Mathematics, University of Padua, Italy. He was the first student enrolled in the joint Ph.D. program in Computer Science organized by University of Bologna (UniBO) and University of California, Los Angeles (UCLA). Through this program, he received his M.S. degree in Computer Science from UCLA in 2005, his Ph.D. degree in Computer Science from UniBO in 2006, and his Ph.D. degree in Computer Science from UCLA in 2007. From 2007 to 2010 he was an Assistant Professor at the Department of Mathematics of the University of Padua. His research is primarily focused on protocol design and analysis for wired/wireless networks, with emphasis on network-centric multimedia entertainment and vehicular networks. On these topics, he is active in various technical program committees in prominent international conferences and is co-author of more than 100 papers, published in international conference proceedings, books, and journals.