A Resource-Aware and Time-Critical IoT Framework

László Toka*‡, Balázs Lajtha*, Éva Hosszu*, Bence Formanek†, Dániel Géhberger†, János Tapolcai*§
* High-Speed Networks Laboratory, Budapest University of Technology and Economics, Hungary
† TrafficLab, Ericsson Research, Hungary
‡ MTA-BME Information Systems Research Group, Hungary
§ MTA-BME Future Internet Research Group, Hungary

***Abstract*—Internet of Things (IoT) systems produce great amount of data, but usually have insufficient resources to process them in the edge. Several time-critical IoT scenarios have emerged and created a challenge of supporting low latency applications. At the same time cloud computing became a success in delivering computing as a service at affordable price with great scalability and high reliability. We propose an intelligent resource allocation system that optimally selects the important IoT data streams to transfer to the cloud for processing. The optimization runs on utility functions computed by predictor algorithms that forecast future events with some probabilistic confidence based on a dynamically recalculated data model. We investigate ways of reducing specifically the upload bandwidth of IoT video streams and propose techniques to compute the corresponding utility functions. We built a prototype for a smart squash court and simulated multiple courts to measure the efficiency of dynamic allocation of network and cloud resources for event detection during squash games. By continuously adapting to the observed system state and maximizing the expected quality of detection within the resource constraints our system can save up to 70% of the resources compared to the naive solution.**

***Index Terms*—Internet of Things, cloud computing, cloud control, resource provisioning, adaptive, dynamic, QoS, QoE**

I. Introduction

Real time sports data analytics is an emerging field: Hawk-Eye is getting commonly used for tennis, cricket and snooker and recently goal-line technology has appeared in soccer[1]. For financial reasons these technologies are currently used only in the top domestic leagues and at major international competitions[2], because right now there is a serious engineering challenge to design such systems for affordable price. The main factor of the high cost comes from the dedicated hardware that handles the time-critical processing of big amounts of data on site. Cloud computing would provide a cost efficient solution, and we envision that in the near future these technologies will find their way into leisure sport facilities through low cost cameras and sensors around the field, backed up by service-oriented cloud-based data analytics. But for a cloud-based system two constraints are to be met: low latency for real-time applications and affordable uplink speed. In this paper we investigate the problem of connecting an IoT system to the cloud with the aim of processing time-critical high bandwidth data, sport referee systems being an use case example.

Our main idea is to make such systems resource-efficient by allowing the cloud to control the rate of data streams sent for processing from the site to the cloud. Roughly speaking the cloud predicts the location and timing of the next possible events that need to be captured. Based on the prediction some video streams are awarded more bandwidth and other cameras are switched off or instructed to operate at lower bandwidth settings. In this way the uplink is mostly occupied with data important for the analysis. For example, the camera far form the ball does not send pictures in case of sport referee systems.

For a proof of concept, we chose a relatively well-contained sport: squash. On the verge of becoming an Olympic sport, squash has an estimated player base of 15-25 million world-wide, playing on over 50.000 courts. Squash is an indoor sport, played in a relatively small room[3], only two players share the court, where the walls and racquets provide an opportunity for sensor installation. As an other benefit uniform artificial lighting and plain wall background makes image processing less resource demanding. While the rules of the game contain complex elements (like let and stroke), basic building blocks of the game are fairly simple: the time, location and attributes of ball strikes, the two players' position and movement, and the location of ball impacts on the walls and on the floor constitute the game data model.

In our testbed we concentrated on strikes, player positions and ball impacts occurring on the front wall. We equipped a single court with our smart squash system: we installed two cameras and a wireless IoT network supporting the racquet sensors, connected to our cloud through a gateway. All data was routed, processed and stored in the cloud. Based on low-bandwidth information, i.e., low-frame rate and low-resolution top camera streams and racquet sensor data, we predicted the resource needs of the ball impact-detection camera, enabling bandwidth-efficient upscaling to multiple courts.

Fig. 1 shows the functional blocks of our system. Low data rate *sensors* provide a steady stream of raw data that is processed in the cloud for contextual information. These components are not affected by the resource allocation al-

[1]As of July 2016, FIFA's quality programme website lists 96 equipped stadiums, see http://quality.fifa.com/en/Goal-Line-Technology/FIFA-certified-GLT-installations/#/index

[2]Apart from installation, analysing the data costs $3,900 each game http://www.mlssoccer.com/post/2013/04/25/mls-commissioner-don-garber-says-league-wont-adopt-goal-line-technology-2014

[3]A squash court is of size 6.4m by 9.75m with a minimum height of 5m

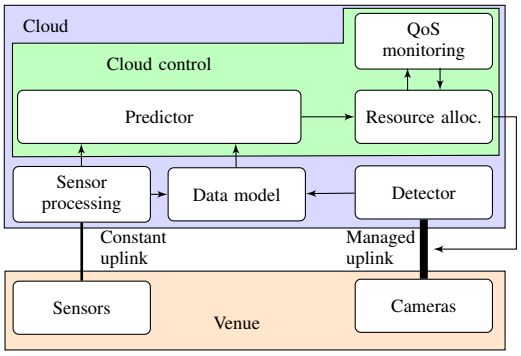


Fig. 1. System overview with functional blocks

gorithm, they are considered a constant operational cost. Within the cloud there are *detector* blocks that are responsible for processing the camera streams. These perform the most resource-intensive task: detecting events from video data. Also running in the cloud, there are processes that implement the *predictor* algorithms that forecast future events with some probabilistic confidence based on a dynamically recalculated *data model*. In the core of the proposed system, we have the *resource allocator* block which, based on the different predictions, manages uplink bandwidth and CPU resources, i.e., performs real-time optimization of resources. The resource allocation is a synchronous process, discrete time increments of 33ms are based on the frame rate of the installed cameras in our case, and resource allocation decisions are made and enforced for each time frame. Long term utility maximization is handled by the *QoS monitoring* component that ensures overall fairness and possibly service level differentiation.

Our contribution is the resource allocation algorithm that maximizes the overall utility of aggregated processing of many squash fields in the cloud. In particular we are focusing on *how to handle situation when there are not enough resources*, e.g. if the uplink does not have sufficient bandwidth for traditional multiplexing of video streams coming from many courts, and the *resource allocator* must make fast choices between the squash courts and data stream requests. In this paper we investigate how an optimal allocation can be reached.

Our contribution lies within:

- Providing maximal QoS for the users of the IoT system, given the resource constraints;
- Enabling the implementation of time critical IoT use cases, e.g., video referee for squash, due to fast optimization algorithms in the resource allocation mechanism.

The paper is organized as follows: in Sec. II we overview the related work; in Sec. III we present our resource allocation algorithm, we provide the definition of the optimization problem and we show a fast dynamic programming algorithm to compute the optimal solution; in Sec. IV we overview the experiments with our prototype on a single court, and investigate the inputs for the optimization problem; in Sec. V we present simulation of multiple courts using the model we built on collected data; in Sec. VI we conclude our work.

II. Related Work

Automating sport event data gathering includes video based player detection, tracking and interaction recognition. Challenges of tracking players on video feeds are discussed by Gerke [1] and by Liu [2], such as player identification and interaction detection on dozens of players at a time.

Video processing has been used extensively in the last decade [3]: while research focuses on player tracking and motion detection [4], HawkEye [5] is the *de facto* ball tracking and trajectory estimation system applied at competitions. An optical beacon-based motion tracking system, Ubitag [6], has been evaluated for detecting players' movements and motions. The tennis racket has been another source of data in [7], now commercialized by several manufacturers [8], [9]. Every system mentioned so far is a self-contained solution.

We have seen in the world of the Internet of Things that connected devices can benefit hugely from a cloud-based back-end. For offloading processing-intensive tasks cloud-based virtualization infrastructures have emerged [10] that handle data processing, data mediation, access control and billing [11] for IoT systems. With the wide availability of WiFi and LTE networks, and with the advent of the 5G technology [12], a new IoT device class has emerged, the streaming cameras [13], e.g., remote facility management solutions can benefit of security cameras following the IoT paradigm. Separating video capture and processing with standard interfaces enables a more flexible infrastructure, where the tenants decide which streams have to be processed for what ends, and a service center provides image processing and event handling. While sport video analytics still rely on onsite detection, the low frequency of events that need video capture makes the case for cloud based on-demand video processing.

Another related field with similar challenges is online gaming: the industry has already started to exploit the possibilities of the cloud by moving heavy computation into data centers. The clients in such gaming scenarios are only transmitting the user input to the cloud and playing out the rendered video stream [14]. The quality of the network and the shared nature of the cloud are the factors which determine the latency and as a result the Quality of Service (QoS) of cloud gaming.

For cloud gaming, the latency threshold for proper QoS is mostly defined to be 100 ms and it is further divided into network latency (80 ms) and computation latency (20 ms) in the literature. Li et al. [15] measured the wide area latency of 4 different cloud providers from 260 points in the US and except one provider, the latency is under 80 ms to the closest data center from 80% of the measurement points. Choy et al. [16] reports 70% coverage for the same 80 ms threshold using the Amazon cloud in the US. Against long tails such frameworks, e.g., Bobtail [17], can be used that allocate multiple virtual machines during the deployment, measure the latencies and shut down the underperforming instances.

The latency requirements of our application are similar, although somewhat more relaxed compared to those of cloud gaming. However, in our case as the video is streamed from

the clients to the cloud, the load travels in an uplink direction. As most access networks provide limited uplink, bandwidth and latency metrics are especially important in our case. Sunderesan et al. [18] investigated multiple Internet service providers and found that the upload throughput is fairly consistent, the last mile access latency varies between 10 and 30 ms and that even the network equipment at the user can influence the service quality. As a result the quality of the network at a particular endpoint has to be investigated before the deployment of a time sensitive application.

III. Cloud Resource Optimizer Algorithm

In this section we present our resource optimizer algorithm. For simplicity first we assume that the cloud has infinite capacity and the only bottleneck is the uplink bandwidth. Later we explain how to extend the formulas for multiple resources such as CPU core, GPU core, memory, disk, etc. Our aim is to maximize QoS, which is the number of detected events of the game, where the event detection accuracy depends on the amount of resources we allocate to the detection module.

A. Problem Formulation

For each game the cloud runs the detector process to capture the events. In our use case we defined these events to be front-wall ball impacts, as the detection of those consumes the vast majority of resources, opposed to the low-bandwidth contextual information like player position, strikes, etc. In general, the accuracy of event detection depends on the quality and quantity of the received data and the allocated cloud resources. Ideally, to analyze the game we need to capture all the events; however, an event miss may happen for many reasons and not necessarily because of lack of resources. Our goal is to minimize the number of missed events, or the price of missing events if prices to event misses are assigned.

The time is divided into time frames, and at the end of each time frame the predictor process computes a *utility function*, which is the expected number of events detected in the next time frame depending on the amount of resources it will receive (see Fig. 2 as an example). The resource optimizer aggregates these utility functions to maximize QoS: defined as a function of the number of detected events across all squash courts. An example of an aggregation of utility functions maximizes the minimum of the detection probabilities, which ensures the highest minimum quality for every court.

B. Utility Function

One of the most critical part of the scheduling is to define a utility function, denoted by $p()$, for each court. The basic utility is the probability of detecting the next event, i.e., a ball impact. By definition the utility function is 0 for zero bandwidth ($p(x) = 0$). We can assume that the utility function is an increasing step-function of the resources to be allocated, as assigning more resources should not decrease the chance of detecting an event. The largest value the utility function can have is the expected number of events in the next time frame. Fig. 2 shows an example of two utility functions for two courts. For Court 1 (Fig. 2a) the probability of a ball impact in the next period is 0.7 and no other events are possible. If we allocate less than 1 unit of bandwidth the event will be surely missed. Based on the sensor data of the racket the prediction algorithm estimates the ball impact in the left side of the wall with probability 0.65, thus we can crop the image to send only the left part, which requires 1 unit of bandwidth. Finally if we send the full picture of the wall 2 units of bandwidth is required. On Court 2 (Fig. 2b) the predictor realizes the ball was hit but the chance to reach the wall is still small, the probability of wall impact is much smaller, just 0.3. Here we can reduce the time period of sending camera pictures. If the pictures are sent through the whole time period 2 units of bandwidth should be allocated. Another option is delaying the start of transferring the camera pictures, which will result a linear increasing utility function depending on the fraction of time the pictures are sent in the time period. In this case the prediction algorithm was not able to estimate the rough location of the wall impact. Note that in practice the utility function is rarely a step function. See also Sec. IV on our practical experiences in computing the utility functions.

The utility is typically a multidimensional function of the resources, such as uplink bandwidth, CPU core, etc. This highlights an interesting trade-off between bandwidth and CPU. If we have more CPU but less bandwidth we may send low quality video images and run more sophisticated detection algorithms, and vice versa. Thus when the resources are scarce we are faced with the following interesting trade-off: should we use the high-performance algorithm on less data, or the low-performance one on high resolution video?

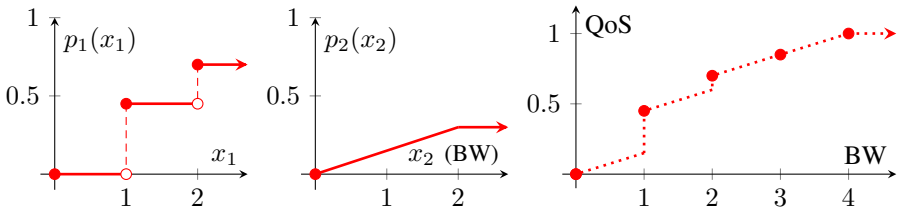
All ball-related events on distinct courts are naturally independent, thus aggregating the utility for these events is straightforward. The simplest case is to consider the total number of detected events, and to maximize the expected number of detections. In this case the utility for each time frame is the sum of the detection probabilities given the allocated resources.

C. Dynamic Programming

First let us describe the problem for n courts and B amount of allocatable bandwidth (BW). For every court we are given a list of utility functions $p_i(x)$, corresponding to the probability of detecting a ball impact on court i depending on the allocated bandwidth x. Our goal is to maximize the expected number of detected ball-impacts, or equivalently, the aggregated detection probabilites. The problem can be formulated as a mathematical program as follows.

$$\begin{aligned} \text{maximize } & \sum_{i=1}^{n} p_i(x_i), \\ \text{subject to } & \sum_{i=1}^{n} x_i \leq B, \\ & x_i \geq 0, \qquad i = 1, \ldots, n. \end{aligned}$$

We will solve the mathematical program with dynamic programming, for which we need to define states. A state of



(a) Court 1 (b) Court 2 (c) The aggregated QoS function

$$u_{1,0} = p_1(0) = 0, u_{1,1} = p_1(1) = 0.45,$$
$$u_{1,2} = p_1(2) = 0.7,$$
$$u_{2,0} = u_{1,0} + p_2(0) = 0,$$
$$u_{2,1} = \max\{u_{1,1} + p_2(0), u_{1,0} + p_2(1)\}$$
$$= \max\{0.45 + 0, 0 + 0.15\} = 0.45,$$
$$u_{2,2} = \max\{u_{1,2} + p_2(0), u_{1,1} + p_2(1),$$
$$u_{1,0} + p_2(2)\} = \max\{0.7, 0.575, 0.25\} = 0.7$$

(d) The dynamic program for $B = 2$

Fig. 2. Example on utility functions and QoS, which are the expected number of detected events as a function of bandwidth.

the problem (c, b) consists of the court index $c \in \{1, \dots, n\}$ and the amount of bandwidth b already allocated to courts $1, 2, \dots, c$, where b is in the closed interval $[0, B]$. Note that while c needs to be an integer, as it is a court index, there is no such restriction for b. The only restriction we have is that for every court the number of different scenarios should be finite. The state space is the set of feasible states, which is

$$\mathbf{S} = \{(c, b) : 1 \le c \le n, 0 \le b \le B\}.$$

The actual number of states depends on how many different values b can take. Let us define a working variable $u_{c,b}$ assigned to each state, which represents the QoS on courts $1, \dots, c$ using b units of BW. The optimal $u_{c,b}$ can be computed by solving the following recursive equations. First, for a single court the QoS equals to the utility function thus we have

$$u_{1,b} = p_1(b) \ , \qquad b = 0, \dots, B \ . \tag{1}$$

For the internal states optimal decisions are driven by the following equation:

$$u_{c,b} = \max_{x=0,1,\dots,b} \{p_c(x) + u_{c-1,b-x}\}, \tag{2}$$

for $c = 2, \dots, n$.

The final value we get is $u_{n,B}$, which gives us the QoS to expect. To process the states we start at Court 1, and compute $u_{1,b}$ for $b = 0, \dots B$, next compute the states for Court 2 $u_{2,b}$ for $b = 0, \dots B$, etc. Finally, we read out the the optimal x_i^* for all the courts in the opposite direction: it is the x for which $u_{i,B-b_i^*}$ takes its maximum, where $b_i^* = \sum_{j=i+1}^{n} x_j^*$.

Let us explain the dynamic program on the example of Fig. 2. We allocate the bandwidth in integer units, and the number of courts is $n = 2$. Let the bandwidth constraint be $B = 3$. The states are $c \in \{1, 2\}$ and $b \in \{0, 1, 2, 3\}$ which is 8 states in total. See the dynamic program on Fig. 2d till $B = 2$, and $u_{1,3} = p_1(3) = 0.7$ and the final solution is $u_{2,3} = \max\{u_{1,3} + p_2(0), u_{1,2} + p_2(1), u_{1,1} + p_2(2), u_{1,0} + p_2(3)\} = \max\{0.7 + 0, 0.7 + 0.15, 0.45 + 0.3, 0 + 0.3\} = 0.85$. See also Fig. 2c as illustration of the QoS function.

D. Multiple Resources and General Utility

As a straightforward generalization we consider multiple resources with a general utility function. Say we have m resources $B_1, B_2, \dots, B_m$. Apart from bandwidth restrictions we may have further constraints on i.e., the total number of CPU cores we may allocate for the courts.

Let x_{ij} denote the amount of resource j we allocate to court i in a given time frame. Let the set of allocations be $X = \{x_{ij}, i = 1 \dots, n, j = 1, \dots, m\}$. Let $p(X)$ be the gain if we make these allocations, and let $U(\cdot)$ be the desired utility.

The problem can be formulated as follows.

$$\text{maximize } U(p(X)),$$
$$\text{subject to } \sum_{i=1}^{n} x_{ij} \le B_i, \qquad j = 1, \dots, m,$$
$$x_{ij} \ge 0, \qquad i = 1, \dots, n, \quad j = 1, \dots, m.$$

First we have to re-define the states of the problem. A state $\mathbf{s} = (c, b_1, b_2, \dots, b_m)$ represents the amount b_j of resource $B_j, j = 1, \dots, m$ that has already been allocated when we consider court c. Similarly as before, the state space is

$$\mathbf{S} = \{\mathbf{s} : 1 \le c \le n, 0 \le b_i \le B_i, i = 1, \dots, m\}.$$

The new recursive equations that define the optimal decisions are the following. For the first court we have

$$u_{1,b_1,\dots,b_m} = p_1(b_1, \dots, b_m) \ , \qquad \begin{matrix} b_1=0,\dots,B_1 \\ \vdots \\ b_m=0,\dots,B_m \end{matrix} \ . \tag{3}$$

For the internal states optimal decisions are driven by the following equation:

$$u_{c,b_1,\dots,b_m} = \max_{\substack{x_1=0,1,\dots,b_1 \\ \vdots \\ x_m=0,1,\dots,b_m}} \{p_c(x_1, \dots, x_m) + u_{c-1,b_1-x_1,\dots,b_m-x_m}\} \tag{4}$$

for $c = 2, \dots, n$.

Theorem 1: The QoS can be computed in $O(nmB_{max}^{m+1})$ time, where $B_{max} = \max\{B_1, \dots, B_m\}$.

Proof: The maximum in (4) is performed over a $O(B_{max}^m)$ items, and there are nmB_{max} states. ∎

IV. A Dynamically Resource-Provisioned IoT System Prototype

Here we summarize our experiments with the test-bed built on a squash court for processing time-critical IoT data in the cloud. To showcase our IoT system we chose the front-wall ball impact detection service in squash courts. Providing an exact impact location and time, in addition to player positions and racquet strikes, gives several important metrics – ball

speed, ball trajectory, strike type –, and can be the basis of video referee, training, score counting applications and augmented reality games. In those latter the strict latency constraints of delivering good experience, similar to the challenges addressed in online gaming applications, are evident.

Fig. 3 shows the process chain from data gathering to the resource allocation decisions. Data is collected from several squash courts from diverse *sensors*: racquet sensors, microphones and cameras, a selected subset of available data is routed to the cloud based event detection processes. The detected events are incorporated in the predefined *data model* updating it to the current state of the system. Based on the current *data model*, a prediction is given on the expected system state in the very near future. Finally, based on the prediction utility functions are computed for each court and the *resource allocator* decides the how bandwidth and server resources are allocated for each court. The whole process is periodically repeated with 33ms-long time-frames. In the following we describe each step in more details.

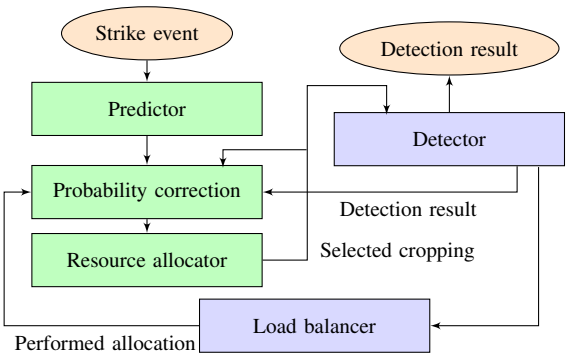


Fig. 3. Data process chain

A. Uploading and Processing Sensor Data

For this specific use case we use three different *sensors*. A motion processing unit embedded in the squash racquet transmits low bandwidth measurement data that is parsed in the cloud for strikes. A birds eye view camera uploads low frequency (3 FPS) low resolution (320p) images for player positioning. The detection of front-wall ball impacts is also implemented in the cloud and processes the video feed coming from a camera installed at the back of the squash court spotting the front wall. The cameras are connected to a RaspberryPi SBC (Single Board Computer) where video streams are generated by being compressed to H.264 [19] format. The encoded video is then multiplexed into MPEG-TS [20] (Transport Stream). The wall-facing camera streams a 720p HD (High Definition) video with 2 Mbps constant bitrate, providing sufficient picture quality for the impact detection, because the image of the front-wall is simple and stationary.

There are several ways to further reduce the bandwidth demand of an already compressed video stream. The picture quality can be reduced, i.e., by applying higher quantization value in the encoder. The resolution of the pictures can also be reduced. Frame size can also be reduced by cropping the front wall. However, detection accuracy deteriorated considerably when video quality or resolution dropped below the required minimum level. Also, increasing either quality or resolution, even well above that level, yielded no further benefit. Another way to reduce the video upload bandwidth without losing detection precision is to transmit only a small burst of the video frames in the expected time of the ball impact. In Sec. IV-B we describe how parts of the front-wall are selected by the Predictor module for recording and uploading.

B. Data Model and Event Prediction

Different from tennis or table tennis, squash is a strategic game where the next ball impact location is highly predictable using the racquet sensor data. As players share the same court they have to strike in a manner that the adversary should have the means to return the ball without major obstruction. Meanwhile players also have to avoid putting the adversary into an advantageous position: they have to choose a strike that has a great probability of success. Between these constraints a player's options are limited; furthermore, the more skilled the opponent, the less choice will be left when making the strike. Examining games of international tournaments, patterns with highly predictable outcomes emerge. Building a data model based on overall and individual strategy improves the accuracy of predictions and further reduces the amount of resources required for detection.

Our IoT system records atomic events of the gameplay: ball impacts on the front-wall, racquet strikes and player positions. These events play a key role in all sport analytics use cases. Based on the data we collect, it is impossible to predict the time window or a surface region of the ball impact on the front-wall: the predictor provides the probability of successful detection in the function of presumed resources allocated to the data collection and processing.

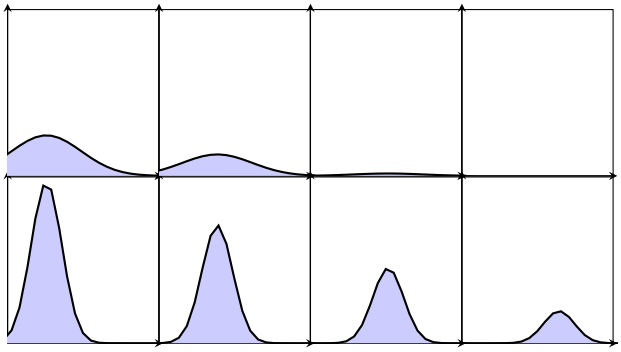


Fig. 4. The probabilities of front-wall ball impacts at different areas on the wall in the same time interval

Our implemented prediction algorithm works as follows. The prediction is triggered when a strike is detected based on racquet sensors. Racquet sensors provide some approximate information about the type of the current strike. Predictor calculates the ball's and players' trajectories from the last strike and the last ball impact. Based on the time and position

differences, and the data model, the prediction algorithm determines the type of strike the player is performing, and evaluates the probability distribution of the next front-wall ball impact in space and time. The probability distribution is given in the function of x, y and t, where x and y are the horizontal and vertical coordinates on the front wall and t is the time. Fig. 4 demonstrates the probability distribution of a straight strike made at the left wall of the court: each eighth of the front wall has its time dependent probability function, a straight ball to the bottom left area being the most probable, with an early impact time, and other impact locations having more wide spread probabilities.

C. Resource Allocator

In our testbed, the resource allocator solely focuses on the uplink bandwidth, and omits the provisioning of cloud resources. In each timeslot the uplink bandwidth allocation for camera feeds is controlled by the dynamic programming algorithm that maximizes the overall impact detection. Based on the predictions of the possible ball impacts the utility functions are computed for each court in the following way.

We divided the front wall into 8 areas by splitting the front wall horizontally in 2 and vertically in 4 (see also Fig. 4). Although impact probability is evaluated for each area of the video stream, we limit possible cropping to a single rectangular area. There are six different sizes of areas: 1/8, 2/8, 3/8, 4/8, 6/8 and full front wall. For each area size the cropping resulting in the highest detection probability is chosen.

In order to map image area to bandwidth usage, the mechanics of the video encoder have to be taken into account. The H.264 video format - as many modern video formats - uses predictively coded pictures, "P" frames, which contain only the difference information relative to a reference frame in our case the previous "I" or "P" frames. To start the video decoding there has to be a picture which contains the full information of the image, an intra-coded "I" frame, that can be decoded without information from previous frames, serving as basis for the following predictively coded "P" frames.

Fig. 5 shows an illustration of the frame types and sizes in an actual video from the front wall. As it can be seen "I" frames are larger then the predictively coded "P" frames, because there is very little difference between consecutive frames, only camera noise and possibly the ball. Due to the low latency requirements we do not use "B" frames.

The bandwidth need of a video frame is higher for a new stream than for the continuation of a previous stream: if a particular cropping was requested in the last timeslot, only a "P" frame has to be transmitted for the same associated area. However for any other requested area - even if it is a subset of the previous crop - a new "I" frame is required instead of a "P" frame. Based on our measurements, the bandwidth needs of an "I" frame are five times that of a "P" frame.

D. Cloud Deployment

System components (as depicted in Fig. 1) in the cloud communicate with components at the squash centers, i.e., at the edge of the IoT system: while the *detector* receives racquet, environmental and video sensor inputs, the *resource allocator* sends the resource allocation instructions to the courts periodically. For the communication UDP datagrams are used to avoid the connection setup time for each request and the unpredictable delays due to buffering. Regarding the networking performance of our cloud-based system, the average intra data center throughput is measured to be 500–800 Mbps, while the inter data center throughput is 100–300 Mbps in public cloud systems [15], [21]. In our case, sufficient throughput is key for uploading video streams, however, as the numbers show, a single cloud instance is able to receive more than 50 video streams continuously.

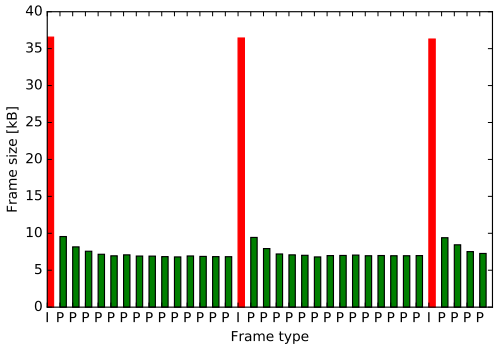


Fig. 5. Frame types and sizes in the front wall video

The cloud provides a scalable infrastructure for applications; however, it must be kept in mind that the scaling of a virtual machine based deployment can be executed on minute time scale. As a result the system for time critical applications have to be over-provisioned in order to be protected against sudden changes in the number of requests to serve. The cloud-hosted system components are designed to be stateless in a sense that prediction and detection requests from the same court can be served by any active instance. This way simple load balancers available in public clouds can be used, and the system can be operated on cloud scale. The only requirement on the load balancer is to use sticky sessions to route the packets of a single video burst to the same instance. The states are stored in the data model, in an in-memory database also provided by the cloud which is periodically loaded for the resource allocation. We use the UDP load balancer component of the Azure Cloud to distribute the load between the running instances.

The shared nature of the cloud implies that deployment of time-critical applications is challenging. The limitations have to be kept in mind during the system design and the evaluation of a particular deployment. The minimum time required to prepare the IoT edge for capturing a ball impact is the sum of the Round Trip Time (RTT) from the squash center to the cloud, the time the *predictor* and the *resource allocator* take and the length of one time-frame in the system. While the latency is reported to be satisfying in the related work on our time scale, it varies between providers and also heavily

depends on the Internet connectivity of the client sites. We evaluated the latency towards the West Europe data center of the Microsoft Azure Cloud from 3 geographically distributed sites, one being the squash center where we built the smart squash court. We were using medium sized virtual machines, as these are expected to perform significantly better than small instances [22]. The results in Fig. 6 show that the RTT is stable between 20 and 30 ms, depending on the selected site. The packet loss was under 0.005% with the used small packet sizes.

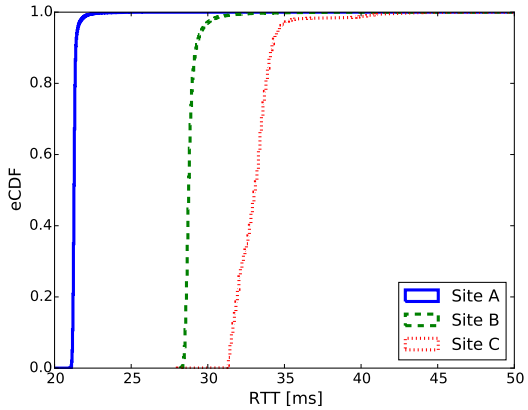


Fig. 6. RTT to the Azure cloud from 3 sites

The most time-critical part of our system is the decision loop after the strike, as this is the time when both uplink delay and prediction runtime have to be considered. When the player strikes the ball, sensor data is uploaded to the cloud with a delay of 10 to 20ms. Prediction can take up to 20ms if an active processing instance is promptly available. In the worst case a whole time-frame, i.e., 33ms, may pass until the resource allocation algorithm is triggered, and additional 10ms are required for the dynamic program to complete. With the downlink delay being between 10-20ms, the average reaction time is 72ms, while worst-case results reach 103ms. This is still sufficient for most of the strikes, even strong straight shots take 142ms on average between the strike and wall impact.

The service has to be monitored in order to quickly detect performance degradation. In order to continuously monitor the performance of the system, we send back a status packet at the end of the prediction to the sites. In Fig. 7 we plot the measured time differences between the racquet strike and the received status packet with 10–15 parallel requests coming from each of the 3 sites. We show the results only for site C which has the lowest quality network towards the data center, the other sites have better RTT characteristics. The dotted black line shows the case when there is only a single active server in the load balancing group which is unable to serve the amount of predictions properly. By simply activating more servers, the prediction time becomes significantly more stable.

The delay of the impact detection itself is lower: delay values fall between 20ms and 68ms depending on network conditions and the capture lag. The delay is composed of the capture lag of 0-33ms, the uplink delay of 10-20ms and the

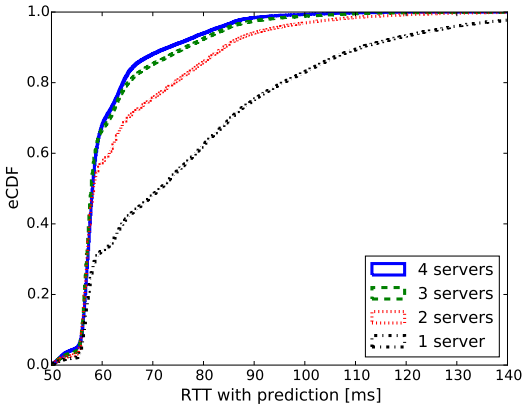


Fig. 7. Prediction latencies from site C

15ms in the detector process spent on image recognition. Due to the UDP based transport we did not measure any significant additional delay in this uploading process.

V. Simulation of Resource Allocation Algorithm

In this section using the measurements of the single squash court we simulate the performance of the resource allocation algorithm for multiple courts.

A. Assumptions

For tractability we measure bandwidth in the unit of P, i.e., the size of a "P" frame covering one eighth of the front wall transmitted over a time-frame. As displayed in Sec. IV-A, the size of an "I" frame for the same area equals five times the size of a "P" frame, and the size grows linearly with the surface of the captured area. Hence, transmitting a "P" and an "I" frame of the full front-wall in a time-frame takes $8P$ and $40P$ respectively. P is equivalent to roughly 200 kbps of throughput at our chosen capture speed of 30 frames per sec.

Based on hundreds of hours of captured and analyzed games on our smart court [23], we built a *data model* and generated strikes and front-wall ball impact events and predictions for a hypothetical squash venue featuring 20 squash courts. We then emulated squash games on those courts, and recorded bandwidth allocation decisions for different total available bandwidth values. For a venue connected to the Internet through a consumer grade subscription the uplink speed typically falls below 20Mbps at the time of writing this paper in 2016, therefore we run simulations with total uplink capacities ranging from $5P$ to $100P$, $100P$ being equivalent to 20Mbps.

The applied *data model* contains realistic inter-wall-impact times and yields predictions with various confidence levels for specific time intervals and wall surface parts. Depending on the assumed total uplink capacity, the *resource allocator* schedules video uploads for courts with the most probable ball impacts, and we consider a detection successful only if the given court received the required amount of uplink at the time of the ball impact and the camera was observing the wall surface part where the ball impact is emulated.

B. The Naive and the Multiplexed Approaches

The naive approach to capturing every ball impact on the court continuously streams the video capture of the whole front wall. An "I" frame has to be scheduled in roughly every 15 frames, resulting in an average bandwidth of $10.13P$ per time-frame per court. When this bandwidth is available, detection accuracy is perfect. Below this value QoS is compromised: either bandwidth must be reallocated among courts, or the observation of selected courts are omitted altogether.

A more refined alternative to the naive is the "multiplexed" approach in which a prediction-aware multiplexer prioritizes the video feed from the courts based on their predicted detection probability in every time-frame. In time-frames when the uplink needs of all courts exceed the available bandwidth, only the highest priority courts receive the required upload bandwidth. Note that in the "multiplexed" approach the ball impact location on the wall is not taken into account in resource allocation decisions. In order to improve the performance per resource ratio, continuous streams are preferred against new streams that require a large new "I" frame.

C. Numerical Results

We defined the performance, i.e., QoS, of the ball impact detection service as the ratio of the number of successful detections over the number of ball impacts detectable with unlimited resources throughout the emulated session. In other words we ignore those ball impacts which are missed by the *detector*, and we account for only those missed ball impacts where the insufficient amount of resources hinders the detection. As said above, a detection is successful when an image of the area containing the simulated ball impact is transmitted at the time of the ball impact. We measured the average quality of service for the 20 courts with the aforementioned amounts of available bandwidth. First we provide the QoS and the resource requirements of the naive approach, then we describe the tradeoff between the QoS versus exploited resources when applying multiplexing with predictions and our proposed optimization method with predictions.

In order to guarantee maximal QoS, i.e., 100% detection of ball impacts, the naive approach uses an uplink bandwidth of $20 \times 10.13 = 202.6P$ per time-frame which amounts to 40Mbps. Provisioning less uplink than this amount the QoS is linearly worse: either whole courts fall out of the detection scope, and/or certain time periods are blacked out in some courts. Fig. 8 shows how the multiplexed and optimized solutions perform compared to the naive approach. These other approaches benefit from the prediction results and based on those they utilize bandwidth more efficiently. At the x axis value 80 both advanced methods reach 98% accuracy. This uplink capacity is equivalent of transmitting "P" frame streams from 10 courts in parallel, as a "P" frame of the full wall uses $8P$ of bandwidth

The reason for this high QoS at such a low cost in uplink resource is due to the fact that the squash ball is designed to lose more than 75% of its kinetic energy on impact, therefore the ball spends more than half of the inter-strike time bouncing back from the front wall. Collected, analyzed and continuously incorporated the gathered knowledge about this, and more, in the *data model*, our system, and specifically the *predictor* module is able to pinpoint the time of the next ball impact with relatively high accuracy once the strike has been detected. This makes uploading video data unnecessary for most of the time between strikes: for some time right after the strike, and naturally for a long time after the ball impact. We argue that such resource savings are possible in most IoT use cases where the resource-consuming detection of events can be triggered by low resource cost data input and predictions based on the well-tailored data model.

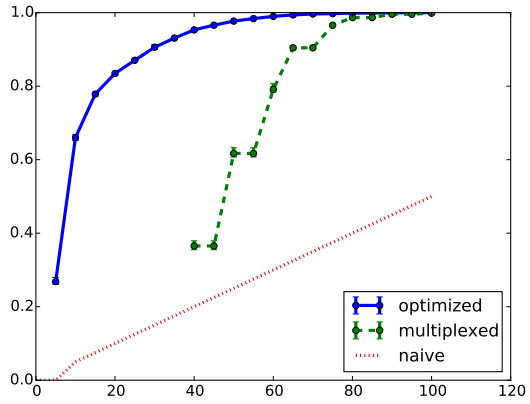


Fig. 8. Normalized QoS vs. uplink capacity for the naive, multiplexed and optimized approaches

Thus ball impact predictions are fairly accurate when strike style, estimated ball speed and player positions are taken into account. This enables prediction-aware solutions to manage concurrent bandwidth requests more efficiently. With the "multiplexed" approach the QoS degrades rapidly if uplink bandwidth is scarce, and the reason for this is twofold. First, the cost of switching between courts is high due to the initial "I" frame that has to be transmitted, so the multiplexer favors streams that are already being uploaded even if the probability of detection is lower for them (the marginal utility of the lower probability is higher due to the smaller cost). Second, the multiplexer does not split the wall of courts into lower bandwidth streams even though predictions come with highly accurate ball impact locations.

Only a more complex optimization algorithm, e.g., a dynamic programming based solution, is capable of managing the many possible bandwidth needs and the related utilities of uploading video of different areas of the front wall. Here are the results of our "optimized" approach. Because of good predictability, $5P$ of bandwidth, equivalent to "I" frames of only an eighth of the front wall, yields around 24% detection efficiency. Having $10P$ bandwidth allows for up to 8 concurrent streams of the selected eighths (an initial "I" frame then 14 subsequent "P" frames), boosting QoS to a stunning 67%. Broadly speaking $10P$ lets the *resource allocator* cover the hot spots after each shot. From this point higher bandwidth is used to cover less probable areas, with $100P$ enabling the

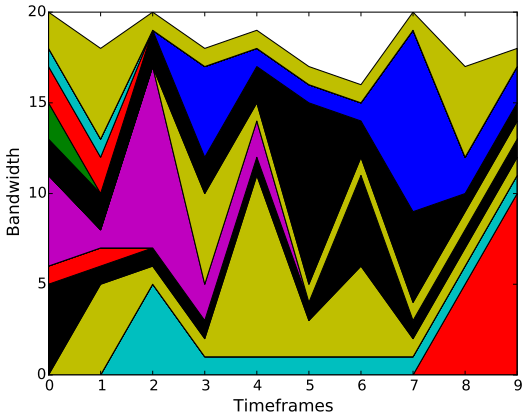


Fig. 9. Upload bandwidth allocated to courts in the optimized approach

coverage of nearly every time-frame and area when and where the ball has the physical chance for impact on the front wall.

To better understand resource allocation with our "optimized" approach applying dynamic programming, we included an extract of allocations with available bandwidth $20P$ for 20 courts for 10 consecutive time-frames. Fig. 9 displays how volatile the allocation can be despite the large cost of "I" frames that are required to change camera capture settings. This adaptability coupled with the predictability of the game results in a low resource hunger at great QoS levels.

VI. Conclusions

In this paper we present the architecture of our cloud-based IoT testbed that maximizes utility when high-bandwidth video streams have to share a narrow uplink channel. We deployed this system to solve a sport analytics use case, namely that of squash front-wall ball impact detection. Using our system, we captured and analyzed hundreds of hours of game footage and built a data model of ball impact probabilities in different contexts. Using this data model we ran simulations for an imaginary venue hosting 20 simultaneous squash sessions.

With the help of our modular cloud-based application we used three different methods for solving conflicting bandwidth requests. First we evaluated the naive approach that divides resources evenly between all the courts that are selected for analysis. Then we proposed two resource allocation methods that benefit of the context-based prediction engine. The multiplexer compressed streams to 45%, while maintaining a 97% QoS. The dynamic program based solution has not only achieved a fascinating 30% compression rate for the same QoS, but also degraded gracefully when even less bandwidth was available, up to the point where, with only 10% of the necessary bandwidth we measured more than 80% event detection. This allows for a significant reduction in uplink bandwidth and processing power in the cloud, and creates an efficient multiplexing among the resources used to detect events in squash games played in parallel.

We focus our future efforts on creating a system that not only provides optimal quality of service in each time interval, but also takes the interdependence of the time-frame into the decision problem. In our current work predictions concerning the future does not influence the optimization at the present. Past allocations had an organic effect on current decisions through the utility function, but using predictions for future detection probabilities may further reduce the amount of missed events.

Acknowledgments

This research was supported by Ericsson. We thank our colleagues from the SmartActive project who provided insight and expertise that greatly assisted the research that led to the submission of this paper.

László Toka was partially supported by the János Bolyai Research Scholarship of the Hungarian Academy of Sciences.

References

[1] S. Gerke *et al.*, "Identifying Soccer Players using Spatial Constellation," in *ACM KDD Workshop on Large-scale Sports Analytics*, 2015.
[2] T.-Y. Liu *et al.*, "Effective feature extraction for play detection in american football video," in *IEEE MMM*, 2005.
[3] M. Nieto *et al.*, "An automatic system for sports analytics in multi-camera tennis videos," in *IEEE Advanced Video and Signal Based Surveillance*, 2013.
[4] X. Wei *et al.*, "Predicting shot locations in tennis using spatiotemporal data," in *IEEE DICTA*, 2013.
[5] "Hawk-eye ball tracking," http://www.hawkeyeinnovations.co.uk.
[6] K. Conroy *et al.*, "Enrichment of raw sensor data to enable high-level queries," in *Database and Expert Systems Applications*. Springer, 2010.
[7] D. Connaghan *et al.*, "Multi-sensor classification of tennis strokes," in *IEEE Sensors*, 2011.
[8] "Smart Tennis Sensor for Tennis Rackets," http://www.sony.com/electronics/smart-devices/sse-tn1w.
[9] "Zepp tennis — analyze & improve your serve & stroke," http://www.zepp.com/en-us/tennis/.
[10] H.-L. Truong *et al.*, "Principles for engineering IoT cloud systems," *IEEE Cloud Computing*, vol. 2, no. 2, pp. 68–76, 2015.
[11] S. Nastic *et al.*, "Provisioning Software-Defined IoT Cloud Systems," in *IEEE FiCloud*, 2014.
[12] M. R. Palattella *et al.*, "Internet of Things in the 5G Era: Enablers, Architecture, and Business Models," *IEEE Journal on Selected Areas in Communications*, vol. 34, no. 3, pp. 510–527, 2016.
[13] A. N. Ansari *et al.*, "An Internet of things approach for motion detection using Raspberry Pi," in *IEEE ICIT*, 2015.
[14] Z. Xue *et al.*, "Playing high-end video games in the cloud: A measurement study," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 25, no. 12, pp. 2013–2025, 2015.
[15] A. Li *et al.*, "Cloudcmp: Comparing public cloud providers," in *ACM IMC*, 2010.
[16] S. Choy *et al.*, "The brewing storm in cloud gaming: A measurement study on cloud to end-user latency," in *IEEE NetGames*, 2012.
[17] Y. Xu *et al.*, "Bobtail: Avoiding long tails in the cloud," in *USENIX NSDI*, 2013.
[18] S. Sundaresan *et al.*, "Broadband internet performance: A view from the gateway," in *ACM SIGCOMM*, 2011.
[19] "Advanced video coding for generic audiovisual services," ITU-T Rec. H.264 and ISO/IEC 14492-10 (MPEG-4 AVC), 2003.
[20] "Generic coding of moving pictures and associated audio information: Systems," ITU-T Rec. H.222 and ISO/IEC 13818-1, 1996.
[21] V. Persico *et al.*, "Measuring network throughput in the cloud: The case of Amazon EC2," *Computer Networks*, vol. 93, pp. 408 – 422, 2015.
[22] G. Wang *et al.*, "The impact of virtualization on network performance of Amazon EC2 data center," in *IEEE INFOCOM*, 2010.
[23] L. Toka *et al.*, "The difference between leisure and competitive squash," in *ACM KDD Workshop on Large-Scale Sports Analytics*, 2016.