Available online at www.sciencedirect.com

SciVerse ScienceDirect

Procedia Computer Science 00 (2014) 000–000

Procedia
Computer Science

www.elsevier.com/locate/procedia

11th International Conference on Mobile Systems and Pervasive Computing (MobiSPC-2014)

DAB: Dynamic and Agile Buffer-control for Streaming Videos on Mobile Devices

Ying Mao[a], Jiayin Wang[a], Bo Sheng[a]

[a]*Department of Computer Science, University of Massachusetts Boston, Boston, MA 02125, USA*

Abstract

This paper studies the video buffer control for streaming video data to mobile devices. We target on the design challenge when the wireless link quality is dynamic due to the the environmental factors or user mobility. We develop a Dynamic and Agile buffor-control scheme, called DAB, that adaptively adjusts the video buffer size based on the measurements of the signal strength (RSSI) and accelerometer on the smartphone. Our goal is to keep a smooth playback while deliver as little data as possible to the end-user in order to save bandwidth cost. We have implemented our solution on Android platform and evaluated it with experiments. Compared to traditional video buffer schemes, our solution DAB significantly improves the performance in terms of the quality of playback and the buffer efficency.

© 2014 The Authors. Published by Elsevier B.V.
Selection and peer-review under responsibility of Elhadi M. Shakshuki.

Keywords:
Mobile devices; Video streaming; Video Buffer; Signal strength; Accelerometer

1. Introduction

In the past decade, smartphones have become one of the most evolutionary devices in the history of computing. The hardware advance and the development of a large variety of mobile applications have attracted numerous users and businesses. Streaming video is always a popular Internet service for end-users including mobile users. Most of the popular stream video providers such as Youtube, Netflix, and Hulu have developed mobile apps to serve their clients. However, designing mobile apps for video streaming faces new challenges that do not exist in traditional wired Internet. One of the most critical issues is the network connection. Compared to the wired network users, a mobile user's network bandwidth is much limited. All the common connections for mobile users such as WiFi, 3G, and 4G have much lower throughput and the link qualities are heavily affected by environmental factors such as obstacles and distance to infrastructure nodes. In addition, user mobility is another unique feature for mobile users. When a user is mobile, the wireless link quality could be highly dynamic and a user can be temporarily disconnected in a certain region. Along with the movement, a user could also trigger handoff protocol to switch the associated infrastructure node. All these dynamics in a wireless network serving mobile users make the design of streaming video mobile apps more challenging. Video streaming is a real-time service and extremely sensitive to the change of network conditions. Any network jitter or delay could pause the playback of the video clip. In addition, network cost is another issue

Ying Mao. Tel.: +1-716-544-9609; Fax: +1-617-287-6433; E-mail: yingmao@cs.umb.edu ;

1877-0509 © 2014 The Authors. Published by Elsevier B.V.
Selection and peer-review under responsibility of Elhadi M. Shakshuki.

that should be considered when developing a mobile app for streaming videos. On one hand, end-users may want to consume bandwidth as little as possible when watching the video because the video delivery may incur a cost, e.g., 3G or 4G users, and additional energy consumption. On the other hand, more importantly, the service providers may want to save the network bandwidth cost while still serving the end-users. A rule of thumb to achieve that is to deliver only the necessary video data to the users. With mobile users, it is more feasible to manage to achieve this goal because the mobile streaming video apps are usually developed by the service providers. Compared to the means of accessing online video in the traditional network. E.g., via a general web browser, mobile streaming apps enable the service providers with more flexible functions to manage the data delivery from the source server to the end users.

In this paper, we target on video prefetch/buffer mechanism that is commonly used in video streaming services. We develop an efficient and dynamic video buffer control scheme that tries to keep a smooth playback with the minimum data delivered to the user by adjusting the buffer size. Our solution especially considers the change of link quality and predicts the trend of change based on the measurement of signal strengths and accelerometer. Our solution includes schemes that monitor and analyze the measurements, and an algorithm that decides the buffer size based on the measurements. We have prototyped our solution on Android smartphones and evaluated the performance with experiments. The results show that our solution significantly improves the performance in terms of playback smoothness and buffer efficiency.

2. Related Work

This work is related with research on online streaming, video streaming on smartphones and mobile quality of experience. Gill et al.[8] examine usage patterns, file properties, popularity and referencing characteristics, as well as transfer behaviors of YouTube, and compare them to traditional web and media streaming workload characteristics. While Cha et al.[5] first provide an in-depth study of YouTube and other similar User Generated Content(UGC) systems and their user's behaviors. After the study on user's behaviors, Cheng et al.[6] look at YouTube.com and the characteristics of its videos. The authors understand that YouTube has millions of videos and try to point out the problems that it's causing, like network traffic cost per bandwidth. By studying the YouTube system, Krishnappa et al.[11] propose a reordering approach for related list which leads to a 2 to 5 times increase in cache hit rate compared to an approach without reordering the related list. As a result of the increased hit rate, 5.12% to 18.19% reduction is found in server load or back-end bandwidth usage. Adhikari et al.[4] build a measurement infrastructure by using PlanetLab nodes with the goal to understand the YouTube system architecture. Recent work has also been done for other streaming platforms such as Krishnappa et al.[10] for Hulu and Adhikari et al.[3] for Netfilx.

The above papers mainly focus on measurement of streaming platforms to understand the architecture, user behavior and improvement of the system performance at server side. As the usage of smartphone blooming, more research work has been done about consuming online video on mobile devices. For instance, Finamore et al.[7] compare YouTube traffic generated by mobile devices (smart-phones,tablets) with traffic generated by common PCs (desktops, notebooks, netbooks). The approach Aquarema[15] enables application specific network resource management and thereby improves the user quality of experience. By managing the streaming process, Shen et al.[14] can minimize the sleep and wake penalty of cellular module and at the same time avoid the energy waste from excessive downloading for energy efficient smartphone video playback applications. Staehle et al.[16] present YoMo which can constantly monitor the YouTube application by comforting and detecting the stalling of the video. Ma et al.[12] provide an overview of the current mobile content delivery ecosystem and discuss the expanding role of HTTP-based mobile video delivery. Metzger et al.[13] study video stream buffering and playback control. They claim that choosing the right buffering model can make a huge difference on the quality of the playback process. However, unlike our work, these previous papers are either focus on back-end system architecture improvement, or front-end, PC smartphone, static quality of experience assurance. Our work, on the contrary, takes mobility into consideration, which is an obvious and important feature of smartphone users.

3. Dynamic and Agile Buffer Control

In this section, we present our solution DAB, a dynamic and agile buffer control algorithm for mobile devices. Our basic idea is to dynamically change the buffer size for prefetching streaming videos according to the link quality and user mobility. The objective is to provide smooth video playback with the minimum bandwidth consumption for downloading video contents.

Essentially, the optimal size of the video buffer on smartphones depends on the video quality and wireless link quality. Video quality indicates the amount of necessary video data for a smooth playback to be prefetched by the player. Apparently, higher quality videos require more data prefetched in the buffer, thus preferring a larger buffer size. Once a video clip is chosen to be player, the quality of the video specifies the minimum requirement of the buffer size, which can be considered as a given parameter for the player. In this paper, we are focusing on the second factor of wireless link quality while holding the minimum threshold of the buffer size defined by the video quality.

The link quality refers to the network bandwidth capacity for supporting video streaming. We aim to develop an algorithm to accommodate the current link quality by adjusting the buffer size. First of all, all our discussions are for the scenarios where link quality is sufficiently good to support the video rate. The buffer control is ineffective in the application if the link quality is poor because when the video content consumption is faster than the prefetch rate, the buffer will eventually become empty regardless of the buffer size causing a pause of the play. Ideally, the buffer should hold the video data just a little ahead of the current position to guarantee the smooth playback, and then prefetch new contents at the same rate as video rate. For mobile users, however, it is extremely hard to accurately keep an appropriate buffer size because the link quality is highly dynamic due to signal propagation and user mobility. Mobile device may even be temporarily disconnected (e.g., handoff between APs). Intuitively, for a static user, when the link quality is adequate, i.e., the bandwidth is much higher than the video rate, the buffer size can be set to the minimum requirement specified by the video quality. When the bandwidth is close to the video rate, we may need to buffer some more data considering small fluctuations of the link quality. For mobile clients, more importantly, we need to estimate the trend of the link quality. The basic idea is to buffer more data when the link quality is becoming worse and vice versa. In this paper, we develop a new buffer control scheme DAB (Dynamic and Agile Buffer-control), which considers the current link quality and predicts the trend of change to decide the buffer size. DAB uses the readings of RSSI (Received Signal Strength Indicator) and accelerometer from smartphones to indicate the link quality and user mobility respectively. In the rest of this section, we introduce three basic components of our solution: measurement of RSSIs, measurement of the accelerometer, and the buffer-control mechanism.

3.1. Measurement of RSSIs

Most of WiFi devices provide an interface for applications to obtain the values of RSSIs which indicate the signal strengths from the associated AP. This measurement can help represent the current link quality and predicate the trend of change in our solution. Conceptually, RSSI is an effective numeric indicator for the current link quality, i.e., a higher RSSI value indicates a better link quality. In theory, RSSI can also be directly mapped to other metrics regarding the link quality such as signal-noise-ration and bit error rate for a particular modulation. Essentailly, RSSIs at the receiver's side are supposed to be reversely proportional to the distance between the sender and receiver, i.e., a client closer to an AP should have larger values of RSSIs. In practice, however, RSSI values are not precise and quite dynamic. Besides the distance, other factors may also affect the measurement of RSSIs, such as blocking objects, wall reflection, sensitivity of WiFi antenna or nearby interferences. Table 1 shows experimental RSSI values measured at the same place. In this experiment, we continuously measure the AP's RSSIs for 100 times with an interval of 500 ms. The values in Table 1 are unstable with quite large fluctuations, especially when the client is close to the AP.

Distance	Max(dB)	Min(dB)	Avg(dB)	Std(dB)
3 meters	-40	-61	-51.980	6.635
6 meters	-55	-64	-62.939	3.651
9 meters	-68	-76	-71.384	2.838
12 meters	-74	-81	-76.263	2.368

Table 1: RSSI measurements at a given location

In our solution, inspired by[9], we use a *RSSI index* which is defined as a range of RSSI values to represent the RSSI measurement. Our intuition is to mitigate the effect of fluctuation with coarse-grained RSSI measurements. With a span of 10dB, we divide RSSI measurement into 5 intervals with index 1 to 5, *index 1*: $[-\infty,-80]$; *index 2*: [-80,-70]; *index 3*: [-70,-60]; *index 4*: [-60,-50]; *index 5*: $[-50, +\infty]$. In this paper, RSSI index is used to represent the current link quality as well as the trend of change. We monitor a series of consecutive RSSI index values and estimate the client's moving direction. We assume the client is leaving the associated AP if indexes become larger. Similarly, if the indexes get smaller, we suppose the user is moving towards the AP. More details will be introduced in Section 3.3.

3.2. Measurement of Accelerometer

Nowadays, accelerometer is a common sensor on smartphones. The accelerometer sensor can continuously measure the acceleration values on the smartphone in the directions of X (lateral), Y (longitudinal) and Z(vertical). Our solution uses the accelerometer readings as another alternative to detect the movement of a client and further estimates his moving speed to help predict the change of link quality. Specifically, we develop two schemes to analyze the accelerometer readings. The first one quickly checks if a user is static or moving. And in case the user is moving, our second scheme, which involves more computations, further derives the moving speed of the user.

Detect whether a User is Moving: To detect whether a user is moving, we only consider the average amplitude of the three directions. The faster the user moves, the greater the average amplitude is. We set a threshold τ for the average amplitude to distinguish if a user is static (sitting or standing) or moving (walking or running). Fig. 1 shows the experimental results with ten users. During the experiments, each user holds the smartphone for one minute in two different states, static and moving. From the experiment, when users are moving, the average amplitude of accelerometer is 1.034 with the highest value is 2.44 and the lowest value 0.86. When users are static, the average amplitude is only 0.1 ranging from 0.2 to 0.02. So we set the threshold τ with a heuristic value of 0.35 (m/s^2).

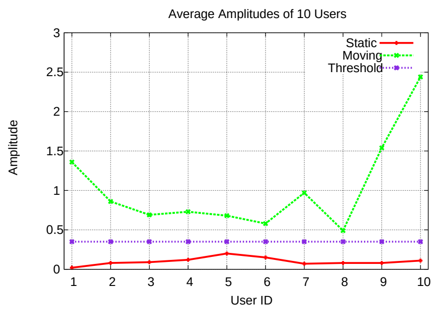


Fig. 1: Average amplitudes of 10 users for 1 minute

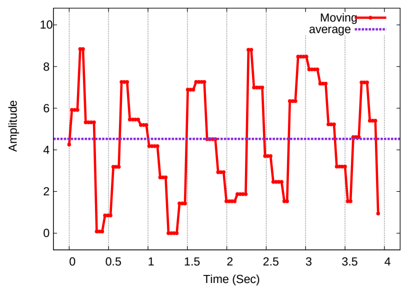


Fig. 2: Average amplitudes of a moving user

Estimate a User's Moving Speed: In order to set an appropriate size of video cache, it's not enough to just predict whether a user is leaving the access point. We also need to estimate how soon it will happen. Thus once our solution detects a user is moving, it will use accelerometer to estimate the user's moving speed. We define five discreet levels to represent the speed (level 0∼4), where level 4 represents the highest speed and level 0 indicates a static user.

Specifically, the moving speed is proportional to the stride frequency and the length of one step. Let S be the moving speed, F be the stride frequency (steps/sec) and L be the length of one step. Then S can be calculated as $S = F \cdot L$. We assume that L is nearly a constant and known as a prior knowledge. Then our focus is to calculate the stride frequency based on the accelerometer measurements. In our approach, We continuously detect the average amplitude $\bar{A}$ of accelerometer every t_m seconds and add the value to a list $\mathcal{A} = \{\bar{A}_1, \bar{A}_2, \bar{A}_3, ..., \bar{A}_N\}$. Obviously, $\bar{A}_1$ is the value of average amplitude at time t_m and $\bar{A}_N$ is the one at time $N \cdot t_m$. Fig 2 shows an example of all values in $\mathcal{A}$. The peaks of amplitudes happen on the foot strike and the troughs of amplitudes happen on the foot lifting at the highest position. So the time interval of two peaks or two roughs indicates the time interval for one step, represented as C. For example, assume A_i and A_j represent two consecutive peaks. The time interval of one step should be $C = (j - i) \cdot t_m$. Then, $\frac{1}{(j-i)\cdot t_m}$ should be the stride frequency F. Once we get the stride frequency of a user, we round it up to the nearest whole number as its corresponding speed level. From the experiments, the stride frequency of a user is seldomly larger than 4 steps/sec. So we set four speed levels for moving users in our algorithm, i.e., the level index indicates the number of steps per second.

Algorithm 1 illustrates the details of deriving the speed level (variable SL) of a user. We use avg to represent the average value of all elements in the list $\mathcal{A}$. In line 2, if avg is less than τ, the user is static and the algorithm return 0 as the value of SL. Otherwise, we use $\mathcal{P}$ and $\mathcal{T}$ to represent the sets of indexes of all peaks and troughs respectively. A_h and A_l represents the temporary highest/lowest values in the list $\mathcal{A}$ and I_h and I_l represent the indexes of them, i.e., $A_h = \mathcal{A}[I_h]$ and $A_l = \mathcal{A}[I_l]$. Lines 4 and 5 update the current highest/lowest value and mark its index. In line 6, the algorithm determines that the trend of values is going down. In this case, the current highest value will be set as a peak and the index I_h will be added to $\mathcal{P}$. Similarly, line 7 finds the troughs in the wave and the index of each trough will be added in $\mathcal{T}$. Once we get $\mathcal{P}$ and $\mathcal{T}$, lines 8∼13 are to calculate the average time interval of each two consecutive peaks/troughs. Finally, the algorithm derives the average stride frequency and calculate the user's speed level(SL) in line 14. Fig. 3 shows the examples of accelerometer readings that lead to different speed levels.

Algorithm 1 Monitor Users' Moving Speed Level

```
1: Initial: $L_h = L_l = 0, A_h = A_l = avg, \mathcal{P} = \{\}, \mathcal{T} = \{\}, D_p = D_t = 0$
2: if $avg < \tau$ then return 0
3: for $i = 1$ to $N$ do
4:     if $\mathcal{A}[i] > A_h$ then $A_h \leftarrow \mathcal{A}[i], I_h \leftarrow i$
5:     if $\mathcal{A}[i] < A_l$ then $A_l \leftarrow \mathcal{A}[i], I_l \leftarrow i$
6:     if $(\mathcal{A}[i-1] > avg)$ and $(\mathcal{A}[i] < avg)$ then $\mathcal{P} \leftarrow \mathcal{P} + \{I_h\}, A_h \leftarrow avg$
7:     if $(\mathcal{A}[i-1] < avg)$ and $(\mathcal{A}[i] > avg)$ then $\mathcal{T} \leftarrow \mathcal{T} + \{I_l\}, A_l \leftarrow avg$
8: for $m = 1$ to $|\mathcal{P}| - 1$ do
9:     $D_p + = (\mathcal{P}[m+1] - \mathcal{P}[m]) \cdot t_m$
10: $\bar{C}_p = \frac{D_p}{|\mathcal{P}|-1}$
11: for $n = 1$ to $|\mathcal{T}| - 1$ do
12:     $D_t + = (\mathcal{T}[n+1] - \mathcal{T}[n]) \cdot t_m$
13: $\bar{C}_t = \frac{D_t}{|\mathcal{T}|-1}$
14: $F = \frac{2}{\bar{C}_p + \bar{C}_t}$,    $SL = min(\lceil F \rceil, 4)$,    return $SL$
```

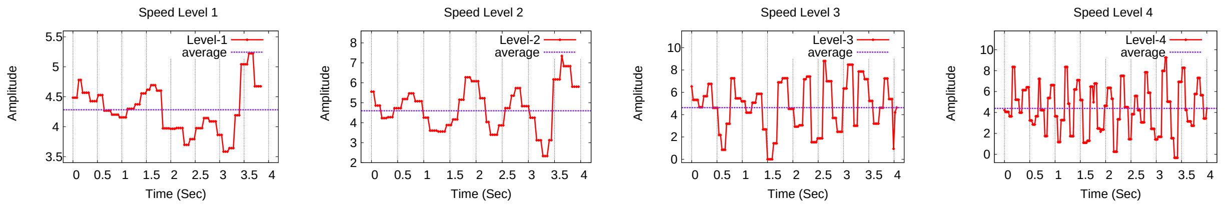


Fig. 3: Accelerometer statistics in different speed levels

3.3. Dynamic and Agile Buffer-control

In this subsection, we present our Dynamic and Agile Buffer-control (DAB) scheme which combines the measurements of RSSIs and accelerometer and adaptively adjust the buffer size on-the-fly. Comparing RSSI and accelerometer readings, we observe the following differences. First, RSSI values are more fluctuating. As we have shown in Section 3.1, there could be a large variance among the RSSIs measured at the same location. Accelerometer measurements, on the other hand, are more consistent and can accurately reflect a user's movement. In addition, accelerometer readings are barely affected by environmental factors. Second, accelerometer values can be monitored continuously while RSSIs are usually measured at discrete time points. As an internal sensor, accelerometer can be accessed any time by any program. Our solution still periodically checks the accelerometer values, but the the interval between two consecutive readings is set to be very small. RSSIs, however, are only available upon the arrivals of packets such as data packets or beacon messages from the APs. Thus RSSIs are not instantly available when the program needs it and the interval between two consecutive RSSI measurements is larger than the interval that we can set for accelerometer readings, e.g., beacon messages from an AP are broadcast with an interval of 100ms. The third difference between RSSI and accelerometer measurements is that RSSIs more directly reflect the link quality while accelerometer readings only indirectly indicate the link change. RSSIs are the measurement of wireless signals. Despite of unavoidable inaccuracy due to environmental factors and hardware sensitivity, RSSIs at a user's side represent the quality of the downlink (from the AP to the user). Conceptually, accelerometer only measures the movement of a user which is not necessarily linked to the quality of wireless signal reception. For example, a user may move back and forth around a particular location, in which case the accelerometer readings are high, but the link quality probably remain the same.

Considering the above characteristics of RSSI and accelerometer measurements, we develop an algorithm that predict the change of link quality and dynamically adjust the buffer size. Specifically, our main idea is to continuously monitor the accelerometer values (Section 3.2) and once a movement is detected, we will further measure the RSSIs (Section 3.1). The new buffer size is decided based on both measurements.

Algorithm 2 shows the details of our DAB scheme. Let B be the buffer size and given the video quality of a clip to be played, let B_{min} be the minimum buffer size required for a smooth play (as discussed in Section 3). We use $\mathcal{R}$

Algorithm 2 Dynamic and Agile Buffer-control

```
1:  Initial: $B = B_{min}, \mathcal{R} = \{\,\}$
2:  if $S > 0$ then
3:      for $i = 1$ to $m$ do
4:          Measure the RSSI value and store it in $\mathcal{R}[i]$
5:      $u = \text{argmin}_{i\in[1,m]} \mathcal{R}[i]$,   $v = \text{argmax}_{i\in[1,m]} \mathcal{R}[i]$
6:      if $u > v$ then
7:          $B = B \times S^{-\alpha} \times \beta^{-(\mathcal{R}[v]-\mathcal{R}[u])\cdot\mathcal{R}[u]}$
8:      else if $v > u$ then
9:          $B = B \times S^{\alpha} \times \beta^{(\mathcal{R}[v]-\mathcal{R}[u])/\mathcal{R}[v]}$
10: else
11:     if $B_u < 0.9 \times B$ then
12:         $c = c + 1$
13:         if $c \geq W$ then $B = 0.9 \times B$,   $c = 0$
14:     else $c = 0$
```

to represent the set of RSSI indexes (Section 3.1) and S to indicate the most recent accelerometer-based speed level (Section 3.2). Initially, $\mathcal{R}$ is empty and RSSI measurement is disabled. Our algorithm only continuously accesses the accelerometer and derive the value of S. Once $S > 0$ (line 2), which indicates a user starts moving, our solution will start to monitor the RSSIs and collect m measurements. The derive RSSI index values are stored in $\mathcal{R}$. In line 5, the algorithm finds the minimum and maximum values in $\mathcal{R}$ and stores their indexes in the variable u and v respectively. When $u > v$, we suppose the link quality is getting better; otherwise ($u < v$), the link quality is degrading. In lines 7 and 9, our algorithm adjusts the buffer size correspondingly. We use two parameters $\alpha \in (0, 1)$ and $\beta > 1$ to define two heuristic functions to change the buffer size. Intuitively, when the link quality is improving (line 7), we should reduce the buffer size towards B_{min}. The reduced amount of buffer size should be proportional to the moving speed (S) and the increase of the RSSI ($\mathcal{R}[v] - \mathcal{R}[u]$). In another work, the faster a user moves or the more improvement is on RSSI index, the smaller the resulting buffer size is. Similarly, when the link quality gets worse, our algorithm increases the buffer size considering the moving speed and the gap between the best and worst RSSI indexes. The difference on the exponents of β in line 7 and line 9 is for another design intuition that slowly increases the buffer size and quickly decreases it in the appropriate circumstances. Finally, we also consider the actual usage of the buffer during the video play. Let B_u be the size of the actual data stored in the buffer, apparently $B_u \leq B$. If B_u keeps smaller than B, which implies that the utilization of B is not full, our algorithm will reduce the buffer size B. In Algorithm 2, we use 0.9 as a threshold for checking the utilization of the buffer. We use a count variable c to record the number of consecutive occurrences of $B_u < 0.9 \cdot B$. When the value of c exceeds another threshold W, the algorithm will adaptively shrink the buffer size to be $0.9 \cdot B$ (line 13).

4. Implementation and Evaluation

4.1. System implementation and experiment setup

We implement our solution DAB on Android platform and deploy it on an assorted set of phones with different manufactures including LG, ASUS, and Samsung. Specifically, in the implementation, we use vitamio[2], an open multimedia framework for android, as our base development platform. Youtube is the video service provider in our experiments. Additionally, to better evaluate our solution, we implement two commonly used buffer mechanisms: Flip-Flop(FF) and Maximum(Max), for comparison. With Flip-Flop mechanism, the video player uses a preset value as the fixed buffer size, such as 10 seconds or 2MB. Whenever the buffer is full, it stops video prefetching. In Maximum buffer mechanism, the player simply keeps downloading data until all the video data has been buffered.

Finally, in order to thoroughly test our application in different network conditions, we attach the android phones to a router running DD-WRT[1], a Linux-based firmware. On DD-WRT, we use tc(traffic control) command to limit the maximum connection speed on certain device. To simulate the mobility, the users continuously move around within the router's communication range, backward and forward.

4.2. Performance Evaluation

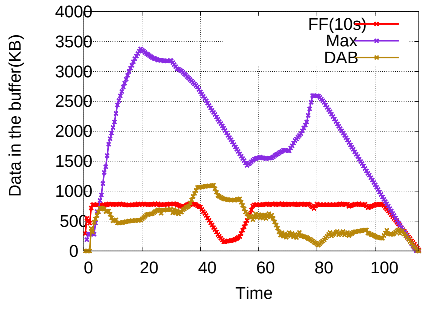
To measure the smoothness of the playback, the performance metric of our application is defined as *total stalling duration equation* $\Delta_t = card\{\ i \mid i \in [1, n],\ C_{BS_i} - C_{PS_i} = 0\ \} \times t_i$, where C_{BS_i} and C_{PS_i} are the i-th current-buffered size and current-played size. When $C_{BS_i} - C_{PS_i} = 0$, the playback is stalling. The application records those values with an interval of 500ms which is represent by t_i in the above equation. By multiplying the interval and the number of recorded stalling cases, we get the total stalling duration which is denoted by Δ_t.

Another objective of our solution is to efficiently use the buffered bytes so that a user can save bandwidth and server can reduce the traffic load. To quantify this target, we define the two parameters E_i and R_i in Eq. 1 and Eq. 2, where T_i is the recording timestamp of the i-th record, E_i is the buffer efficiency at T_i and R_i is the average buffer redundancy rate from 0 to T_i which indicates the extra bandwidth cost on this video and, obviously, the lower the better it is.

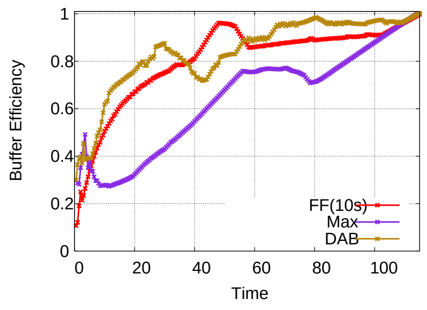
$$E_i = \begin{cases} \frac{C_{BS_i}}{C_{PS_i}}\ , & if\ C_{BS_i} \neq C_{PS_i} \\ 0\ , & otherwise \end{cases} \qquad (1) \qquad\qquad R_i = \frac{\sum_{i \in [1,n]} (C_{BS_i} - C_{PS_i})}{T_i} \qquad (2)$$

In this subsection, we present the evaluation result of our proposed solution. For Algorithm 2, we set $\alpha = 0.3$ and $\beta = 1.1$. For the compared FF method, we set the buffer size as 10 seconds. In our experiments, we use a 115-second Youtube video clip which is 9.38MB as the source. In each test of the three buffer mechanisms, users follow a similar mobility pattern(moving towards and forwards). Particularly, in our proposed solution we maintain a window of 100 consecutive accelerometer readings with an interval of 10ms and if a user movement is detected, we will start to record 5 RSSI readings with an interval of 500ms.

We first test our application under a good connection where the bandwidth on the router is 6MB from service provider. Fig. 4a plots the values of $C_{BS_i} - C_{PS_i}$ which indicate Δ_t by the number of zeros on the x-axis. From the figure, we can observe that all three buffer mechanisms perform well with no stalling ($\Delta_t = 0$).



(a) Buffered Data Size



(b) Buffer Efficiency

Fig. 4: Experiments with a good connection (6MB wired bandwidth on the router)

Fig. 4b compares the buffer efficiency E_i. We consider the commonly observed user behavior that a user often stops playing a video clip before it ends. The axis x indicates the time when a user stops the video. As we can observe from Fig. 4b, if a user stop watching at the very beginning of the video, E_i of DAB, FF and Max mechanisms are all below 50%. For example, at the 10-th second, the DAB, FF and Max's $E_1 0$ values are 63.8%, 49.5% and 27.8%. From 0-th to 38-th second, DAB outperforms FF and Max. From time 29-th to 43-th second, the efficiency of DAB decreases because during this period, the algorithm detects that the user is about to leave the transmission range, and then it allows the player to increase the buffer size in order to prefetch more data. Furthermore, the redundancy rate at the 10th second for DAB, FF and Max, $R_1 0$ are 50.7Kb/s, 87.3Kb/s and 206.4Kb/s, respectively. Our solution reduces 33.4% redundancy rate during these 10 seconds. Therefore, in this scenario with a good link connection, our solution DAB can deliver perfect quality of service to user($\Delta_t = 0$), at the meantime, we achieve high efficiency E at beginning of the video which can help user save bandwidth and reduce traffic load at the server side.

Since most users suffer from slow network connection, we also test our application with a maximum downloading speed 200Kb/s. Fig 5 presents the result of this scenario. Clearly, from the figure, FF and Max suffer from stalling during video playback. The total stalling duration (Δ_t) for DAB, FF and Max in this scenario are, 4, 12, and 3.5 seconds, respectively. Particularly, FF mechanism stalls 5 times, for 5.5s, 1s, 0.5s, 2s and 3s each time. While DAB and Max mechanism only stalls once for 4s, 3.5s. In this case, our DAB solution and Max perform similarly and much better than FF mechanism. However, considering the average redundancy rate from time 0 to 25, comparing to Max mechanism, our solution DAB reduces it by 59.7%.

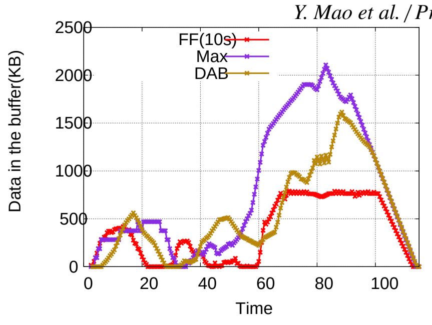


Fig. 5: Connected to 200Kb/s downloading speed

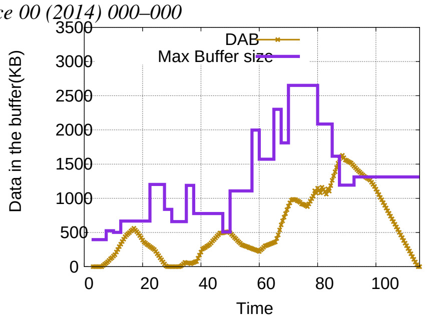


Fig. 6: Buffer size trend change during the movement

Fig 6 shows the buffer size trend of our proposed DAB along with the calculated maximum buffer size during the 115-second test. As we can see from the figure, at the beginning, DAB successfully detects the user's connection conditions and movements. The adjusted buffer size does not limit the actual bytes in the buffer. However, from 80-th to 100-th second, the calculated buffer size helps the user reduce buffer size and improve the efficiency.

5. Conclusion

In this paper, we present DAB, a dynamic buffer-control scheme that adaptively adjusts the video buffer size based on the measurement of RSSI and accelerometer. Our solution predicts the change of link quality and correspondingly changes the buffer size to help maintain a smooth playback while minimizing the bandwidth consumption. The experimental results have shown that our solution is superior to typical buffer schemes.

References

1. Dd-wrt. http://www.dd-wrt.com/.
2. Vitamio framework. http://www.vitamio.org/en/.
3. Vijay Kumar Adhikari, Yang Guo, Fang Hao, Matteo Varvello, Volker Hilt, Moritz Steiner, and Zhi-Li Zhang. Unreeling netflix: Understanding and improving multi-cdn movie delivery. In *INFOCOM*, pages 1620–1628, 2012.
4. Vijay Kumar Adhikari, Sourabh Jain, Yingying Chen, and Zhi-Li Zhang. Vivisecting youtube: An active measurement study. In *INFOCOM*, pages 2521–2525, 2012.
5. Meeyoung Cha, Haewoon Kwak, Pablo Rodriguez, Yong-Yeol Ahn, and Sue Moon. I tube, you tube, everybody tubes: Analyzing the world's largest user generated content video system. In *Proceedings of the 7th ACM SIGCOMM Conference on Internet Measurement*, IMC '07, pages 1–14, New York, NY, USA, 2007. ACM.
6. Xu Cheng, Cameron Dale, and Jiangchuan Liu. Understanding the characteristics of internet short video sharing: Youtube as a case study. *CoRR*, abs/0707.3670, 2007.
7. Alessandro Finamore, Marco Mellia, Maurizio M. Munafò, Ruben Torres, and Sanjay G. Rao. Youtube everywhere: impact of device and infrastructure synergies on user experience. In *Proceedings of the 2011 ACM SIGCOMM conference on Internet measurement conference*, IMC '11, pages 345–360, New York, NY, USA, 2011. ACM.
8. Phillipa Gill, Martin Arlitt, Zongpeng Li, and Anirban Mahanti. Youtube traffic characterization: a view from the edge. In *Proceedings of the 7th ACM SIGCOMM conference on Internet measurement*, IMC '07, pages 15–28, New York, NY, USA, 2007. ACM.
9. Daniel Halperin, Wenjun Hu, Anmol Sheth, and David Wetherall. Predictable 802.11 packet delivery from wireless channel measurements. *SIGCOMM Comput. Commun. Rev.*, 40(4):159–170, August 2010.
10. Dilip Kumar Krishnappa, Samamon Khemmarat, Lixin Gao, and Michael Zink. On the feasibility of prefetching and caching for online tv services: a measurement study on hulu. In *Proceedings of the 12th international conference on Passive and active measurement*, PAM'11, pages 72–80, Berlin, Heidelberg, 2011. Springer-Verlag.
11. Dilip Kumar Krishnappa, Michael Zink, Carsten Griwodz, and Pål Halvorsen. Cache-centric video recommendation: an approach to improve the efficiency of youtube caches. In *Proceedings of the 4th ACM Multimedia Systems Conference*, MMSys '13, pages 261–270, New York, NY, USA, 2013. ACM.
12. K.J. Ma, R. Bartos, S. Bhatia, and R. Nair. Mobile video delivery with http. *IEEE Communications Magazine*, 49:166–175, April 2011.
13. Florian Metzger, Albert Rafetseder, David Stezenbach, and Kurt Tutschku. Analysis of web-based video delivery. In *FITCE International Congress*, Palermo, 2011.
14. Hao Shen and Qinru Qiu. User-aware energy efficient streaming strategy for smartphone based video playback applications. In *Proceedings of the Conference on Design, Automation and Test in Europe*, DATE '13, pages 258–261, San Jose, CA, USA, 2013. EDA Consortium.
15. Barbara Staehle, Matthias Hirth, Rastin Pries, Florian Wamser, and Dirk Staehle. Aquarema in action: Improving the youtube qoe in wireless mesh networks. In *Baltic Congress on Future Internet Communications (BCFIC)*, 2 2011.
16. Barbara Staehle, Matthias Hirth, Florian Wamser, Rastin Pries, and Dirk Staehle. Yomo: A youtube application comfort monitoring tool. Number 467, 3 2010.