Optimized Device Centric Aggregation Mechanisms for Mobile Devices with Multiple Wireless Interfaces

Sanaa Sharafeddine[a,*], Karim Jahed[a], Marwan Fawaz[a]

[a] *Department of Computer Science and Mathematics, Lebanese American University (LAU), Beirut, Lebanon*

Abstract

Wireless broadband technologies and services are witnessing exponential growth to meet the demands of mobile users. State-of-the-art wireless networks are evolving with enhancements spanning all protocol layers and all network components from radio access to core network nodes. This has been coupled with a tremendous transformation of end user mobile devices towards multi-purpose smartphones and tablets with multi-core processing power, extendable memory storage, large battery capacity, and support for a wide range of wireless connectivity options. A standard smartphone currently can support short range Bluetooth and WiFi-Direct connectivity, local area WiFi connectivity, and long range 2G/3G/4G mobile connectivity. This naturally provides opportunities for data aggregation utilizing multiple wireless interfaces simultaneously to enhance device and network performance. In this work, we present the design, implementation, and testing of optimized device-centric data aggregation mechanisms for both file downloading and video streaming applications. The main novelty of the proposed mechanisms is their device-centric design that makes them practical and feasible without any changes to wireless standards; moreover, they are scalable to support any number of wireless interfaces whereas previous related work has dealt with devices having two interfaces only. To demonstrate the effectiveness of the proposed mechanisms in terms of performance gains and practical feasibility, we develop an experimental testbed using Android devices and perform extensive testing for several network scenarios.

Keywords: Bandwidth aggregation, heterogeneous networks, traffic offloading, WiFi-cellular inter-operation, multi-homed devices, WiFi/cellular inter-operation

*Corresponding author
Email address: sanaa.sharafeddine@lau.edu.lb (Sanaa Sharafeddine)

1. Introduction

Global mobile data traffic grew more than 60% in 2014 exceeding 2.5 exabytes per month [1]. By 2018, the annual demand for Internet traffic is expected to exceed 1.5 zettabytes, with nearly half of it generated by mobile devices. This trend is expected to continue supported by advances in wireless broadband technologies, smartphone devices, and mobile applications. This growth is putting mobile operators under immense pressure to handle the demands without negatively impacting the quality experienced by their subscribers. As a result, cellular operators are upgrading their network infrastructure based on advanced LTE/LTE-A features standardized in new 3GPP releases. Carrier aggregation (CA) is brought forward as one of the most efficient features of LTE-Advanced to cope with the rapid increase of mobile data traffic. Carrier aggregation may provide an overall bandwidth of 100 MHz by aggregating up to five component carriers and thus may lead to five-fold bit rates; this makes it specially attractive to LTE operators with non-contiguous spectrum blocks [2, 3]. Actual drive testing confirmed the dramatic improvement of bit rates due to CA functionality [2]. Dual connectivity is introduced to extend carrier aggregation by allowing one device to utilize the radio resources of a small cell and a macro cell simultaneously; this leads to improved end user throughput and mobility support [4]. High acquisition costs of licensed spectrum pose, however, a major challenge to network operators in improving their network performance and thus 3GPP proposed utilizing unlicensed bands using license assisted access to achieve significant gains. The challenges in the coexistence of LTE-Unlicensed and unlicensed systems such as WiFi in unlicensed bands are still not resolved [5]. This high demand for significant bit rate improvements has also triggered serious efforts to initiate standardization activities for the next generation 5G cellular technology with plans for 5G deployments starting in 2020.

Additional key enhancement techniques include heterogeneous network integration and operation with bandwidth or link aggregation [6, 7, 8]. This can lead to significant performance gains as it benefits from the wide area coverage of cellular systems, the dense deployments of IEEE802.11x WiFi access points, and the high end capabilities of end-user mobile devices (smartphones and tablets). 3GPP techniques include access network discovery and selection (ANDSF) mechanisms to mobile devices with aim to offload data from cellular to WiFi; in addition to activities focused on tight integration through which WiFi access points will be connected to cellular core network elements that facilitates smooth mobility and multiple connectivity; this is evolving to radio access network (RAN) level integration that will allow dynamic RAN-based coordinated resource management between both networks. A major complexity is that WiFi access points need to be controlled either by cellular operators or their partners that is not easy to achieve taking into account the existing WiFi landscape with diverse and ad hoc deployments [9, 10].

In this work, we propose a set of dynamic self-adaptive device-centric algorithms for link aggregation in heterogeneous wireless networks customized

to emerging use cases with multiple wireless interfaces and to both file downloading and video streaming mobile applications. The novelty of the proposed mechanisms is their seamless operation with respect to the available wireless networks (e.g., WiFi and 3G/4G cellular) without the need for any proxy server deployment, network assistance, or protocol stack modifications. Yet, we show that they optimized performance by utilizing the multiple interfaces effectively taking into account the application's quality of service requirements. We focus on link aggregation instead of link selection as it has the potential to provide significant gains to mobile users which in turn can be mapped to performance gains to the network operators. To our knowledge, this is the first work that considers full device-centric link aggregation over more than two wireless interfaces and with support to both real-time and elastic traffic types; the practical motivation for considering multiple interfaces is based on recent advances in content distribution with device-to-device cooperation among users in close proximity with respect to each other, e.g., see [11, 12]. This paper builds on our previous work [13], where we have discussed device-centric link aggregation for two wireless interfaces only, with focus mainly on file downloading applications. The main contributions of this work include i) proposing practical dynamic self-adaptive and fully device-centric link aggregation mechanisms, ii) supporting multiple ($N \geq 2$) wireless interfaces, iii) supporting elastic and stream-type traffic, and iv) developing appropriate testbeds that implement the proposed algorithms and show optimized performance with two and more wireless interfaces.

In order to test the performance of the proposed algorithms, we follow an experimental methodology based on testbed development and performance evaluation for an extensive set of scenarios with various network and design parameters. We believe that demonstrating the feasibility of practical implementation on standard Android-based smartphones and demonstrating notable gains under realistic operational conditions are additional novel contributions of this work; this helps bridge the gap between theoretical derivations and algorithmic design on one hand, and engineering implementation on the other hand.

Section 2 presents related literature and highlights the main contributions of our work as compared to existing work. Section 3 presents the system model details and the general link aggregation framework. Section 4 explains in details the proposed link aggregation algorithms for use cases with two interfaces and multiple interfaces covering both elastic file downloading applications and on-demand video streaming applications. The testbed design and implementation is then discussed in Section 5, with results and analysis summarized in Section 6 for a wide range of scenarios with various network and design parameters. Finally, conclusions are drawn in Section 7.

2. Related Literature

In the last few years, literature proposes different techniques to jointly utilize co-existing heterogeneous networks. These can be broadly classified into:

i. network-level techniques that aim at optimizing resources between both networks with network-level coordination and multiuser traffic steering [14, 9, 10, 15, 16]; ii. device-level techniques that aim at optimizing the performance of each device without network level coordination. Due to technical and business challenges of realizing network-assisted heterogeneous network integration in practice, we focus on device-level techniques for effectively utilizing co-existing networks. Research efforts in this direction make use of existing WiFi and cellular interfaces to download content faster either by dynamically selecting the interface that has better quality (known as link selection) [17, 18, 19] or by dynamically splitting traffic over both interfaces proportionally to the quality of each interface (known as link aggregation, bandwidth aggregation, or traffic splitting) [20, 21, 22, 23]. Currently, state-of-the-art smartphones do not take advantage of the coexistence of WiFi and cellular interfaces; only one interface can be active at a time, with priority normally given to WiFi irrespective of link qualities. Most proposed techniques require modifications, or at least some sort of support, at the already-mature layers of the network protocol stack [24, 25]. In addition, many require the deployment of a proxy server between mobile devices and destination Internet servers; in these approaches, the traffic is routed through an intermediate proxy server that implements the proposed bandwidth aggregation support, and that communicates with the destination servers, e.g., see [26]. Proxy-based solutions, however, entail redundant traffic, increase to the end-to-end delay, and congestion situations, which can negatively impact the users' quality-of-service. Some other work requires customized end-to-end communication session including sender and receiver sides and thus cannot be used readily with existing content servers. In [27], the authors propose an energy-aware bandwidth aggregation framework for video over heterogeneous wireless networks. This work considers only two radio access networks (e.g. LTE and WiFi) and requires the deployment of a complete framework where working modules need to be implemented at both sender and receiver sides.

There are few works that assume traffic scheduling takes place in the mobile device without a proxy server and without any network support/modification; however, they require the implementation of link estimation modules to obtain weights for dividing the traffic dynamically between both interfaces, [20, 28, 7, 29, 22, 19]. For example, in [7], the authors present an interesting approach for bandwidth aggregation systems with two options for link quality estimation: either using multiple legacy servers that are geometrically dispersed or using an own deployed server to estimate the WiFi and cellular link qualities. The estimates are then used to determine packet distribution weights over the two interfaces in order to maximize throughput. In [22], the authors propose a dynamic link aggregation scheme for file downloading with continuous link quality estimation based on received data; sequential and parallel implementations are studied using NS-3 network simulations. In [19], the authors present a novel hybrid approach where users take network selection decisions based on partial network assistance via adaptive network-level parameters in addition to locally available information related to quality and needs; they demonstrate performance gains using simulations for different traffic classes. In [30], the authors

propose user-centric multihoming with LTE and WiFi connectivity. They consider two strategies, a simple one that splits traffic according to the peak rate of each interface without any knowledge about the current system load, and a second one that takes into consideration current load information as broadcasted by the network operator. The authors show that the network-assisted approach achieves better performance than the one based on network peak rates. Both approaches, however, evaluate the proportion of the file that has to be downloaded on each interface at the beginning of the transfer without accounting for any network changes that may occur and the actual bit rates.

The novelty of this work as compared to existing literature is that we consider fully device-centric link aggregation mechanisms over more than two wireless interfaces. Our algorithms are self-adaptive and dynamic with respect to the effective bit rate that is received by each supported wireless interface. We proposed algorithms that are customized to file-downloading as well as streaming applications and validated the practicality and performance gains of those algorithms by developing appropriate experimental testbeds with realistic scenarios. In the case of two wireless interfaces, we developed a mobile application that implements the algorithms and evaluates their performance for file downloading and video streaming over cellular and WiFi networks. In the case of devices with more than two wireless interfaces, we developed an emulator that downloads and streams actual content from existing servers using multiple interfaces. The bit rates of the interfaces follow measured traces of wireless connections in order to provide as realistic scenarios as can be.

3. System model

Given a mobile device equipped with multiple wireless interfaces, our goal is to develop optimized device-centric link aggregation mechanisms that efficiently utilize all interfaces over heterogeneous networks to enhance the quality of service for accessing content from origin servers over the Internet. The system model is depicted in Fig. 1a that shows a device equipped with multiple wireless interfaces (I_1, I_2, ... I_n) that are simultaneously used to to receive content from a given server, Fig. 1b shows an example using WiFi and cellular link interfaces.

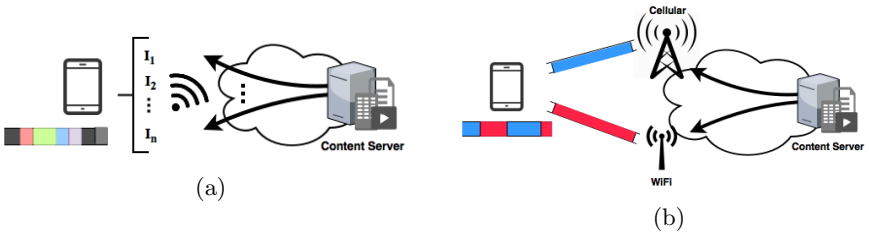


Figure 1: System model (a) Multiple interfaces; (b) Cellular/WiFi link aggregation.

Current mobile devices only support limited number of wireless interfaces but the work with multiple interfaces can be readily applied in a fog network

scenario composed of a cluster of devices, in proximity to each other with very high device-to-device short range bit rates. Devices within the cluster jointly utilize all their wireless interfaces to help one device download content with higher quality (see Fig. 2) [31, 32].

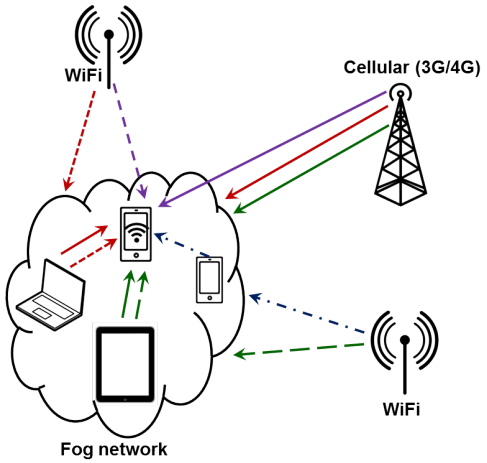


Figure 2: System model example with link aggregation across a cluster of devices cooperating to download content to a given device.

Link aggregation aims to bond the bit rates of multiple wireless interfaces to emulate one super interface capable of delivering superior quality-of-service under varying network conditions. Boosting the network throughput is one of many gains of link aggregation; however, other use cases include energy consumption by minimizing wireless interface activity duration, fault-tolerance by ensuring that a backup interface is always available in case of failure, and load balancing by distributing traffic across multiple heterogeneous networks.

We chose to build our framework on top of the HTTP protocol rather than as a lower-layer service for two reasons. First, using HTTP facilitates a design that does not require any modification to the server hosts, network nodes, or client protocol stack. The only assumption we pose is for the content server to support the HTTP "Partial GET (206)" method; this assumption is satisfied by default on most web servers as "Partial GET" requests are essential to support resuming downloads. The "Partial GET" request adds a "Range" header field to the client's request and, thus, allows the client to retrieve a subset of the content on the server rather than the whole data [33]. The second reason is its popularity for mobile services; an increasingly large number of mobile applications now utilize HTTP for connections with cloud services, due to the variety of features the HTTP protocol supports and its simple stateless operation.

In this work, we differentiate between two broad categories of network services depending on the users' interaction with the application servers. The first category includes content that needs to be fully fetched from the server before it

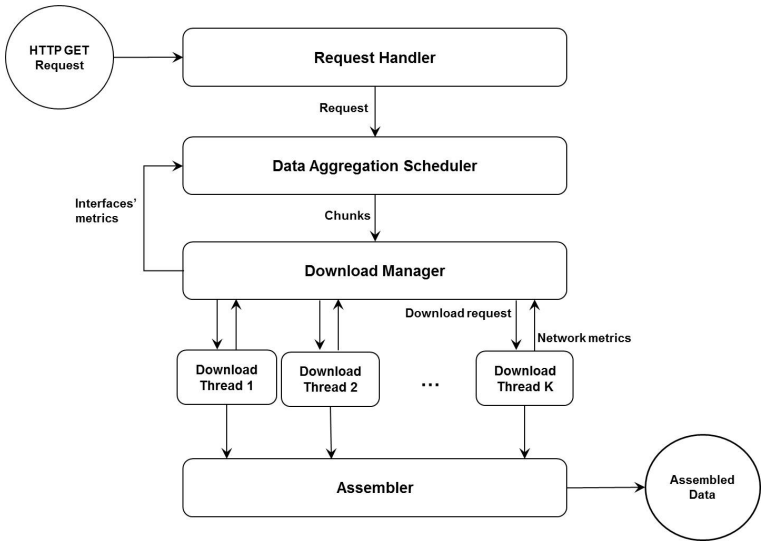


Figure 3: Proposed link aggregation framework implementation modules for mobile devices.

can be put into use such as file downloads and database queries. The second category includes streaming content, where the user can actively engage with the content while it is being downloaded from the server; one example is video-on-demand streaming where the user is playing the video while subsequent frames are still being downloaded. Differentiating between these two service classes is important due to their varying quality requirements. From a user point of view, the most important metric when dealing with file downloads is download time; the sooner the content is fully fetched the sooner it can be put into use. On the other hand, the most important metric when dealing with streaming content is buffering time, which can lead to stalls that negatively impact the user's experience.

We propose a general architecture for device-centric link aggregation that can be tailored to varying service classes and that can be fully realized in practice using state-of-the-art wireless technologies and mobile devices. A key distinguishing novelty of the proposed architecture is its device-centric property and the capability to achieve significant gains without requiring any changes to the server side and without requiring any support from the network operators side. The proposed framework architecture is shown in Fig. 3, and is composed of five main modules:

Request handler: This module acts as an interface between the application layer and the data aggregation intelligence at the mobile device.

Data aggregation scheduler: The scheduler implements the scheduling algorithm intelligence that is responsible for the optimized distribution of requested data over several available wireless interfaces. The scheduler monitors the quality metrics related to the various wireless interfaces,

queries the server for general information about the content (e.g., total content size and service type), and makes real-time decisions on which data chunks should be downloaded over which interface based on the proposed scheduling algorithms (see Section 4 for more details).

Download manager: The download manager is responsible for building the partial get requests for the chunks supplied by the scheduler, and to feed them to the appropriate download threads.

Download thread: Each download thread is bound to a single network interface. All download threads run simultaneously and execute HTTP GET requests supplied by the download manager. The downloaded data is then passed on to the assembler.

Assembler: The assembler receives downloaded chunks from the various download threads in parallel, and re-orders them to generate the complete data stream to be accessed by the end user.

In terms of performance requirements, the proposed framework aims at maximizing the total download rate or, equivalently, minimizing the total download time by utilizing multiple interfaces as efficiently as possible. The optimal solution is to download the content in equal time durations over all interfaces in parallel [21]. Given K interfaces, the instantaneous combined bit rate can then be represented as follows:

$$R_{\text{total}}(t, \Delta t) = \sum_{k=1}^{K} R_k(t, \Delta t), \tag{1}$$

where $R_{\text{total}}(t, \Delta t)$ is the total download rate in time slot t of duration Δt, and $R_k(t, \Delta t)$ is the download rate on interface $k (k \in 1, 2, ..., K)$. The amount of data downloaded per interface D_k can be related to the slot duration as follows:

$$D_k = R_k(t, \Delta t) \cdot \Delta t \tag{2}$$

The total download time T_{total} for a given content size of N bytes will vary depending on the transmission rates over the wireless interfaces during the download period; yet, its value will always be less than or equal to the time required to download the content on one of the interfaces alone. In general, the total download time can be expressed as follows:

$$T_{\text{total}}(t, \Delta t) = \sum_{s=1}^{S} (\Delta t_s + V_s), \tag{3}$$

where S is the total number of slots needed to download the content, s is the slot index, and V_s is the overhead delay per slot due to HTTP request-response round trip time (RTT) delay and lower layer protocol overheads (e.g., TCP congestion control or MAC layer retransmission mechanisms). In general, the total number of slots S varies depending on the content size, the slot duration,

and the interfaces' bit rates. The total effective download bit rate can then be calculated as follows:

$$R_{\text{eff}} = N/T_{\text{total}} \tag{4}$$

4. Proposed Link Aggregation Mechanisms

In this section, we present the details of the proposed device-centric link aggregation mechanisms, with the first part focused on the file downloading service class and the second part focused on the video streaming service class.

4.1. Link Aggregation for File Download Service Class

Typically, a file download process begins when a client issues an HTTP GET request specifying the file path. If the file is available, the server would reply with an HTTP OK response with the file content attached to the message body. At the TCP level, the server would start streaming the file content in sequential packets starting with byte 0 till byte $N-1$ for a file of length N bytes. A key idea behind the proposed framework is to utilize the *Range* header field of the HTTP Partial GET request to simultaneously download different parts of the data stream over multiple wireless interfaces. The segments are then re-assembled to correctly reconstruct the content for delivery to the application layer. The main algorithmic challenge here is to determine how the file should be partitioned and distributed between all interfaces to maximize performance efficiency.

The simplest solution is to divide the file into K equal segments to be distributed over the K interfaces in parallel. However, this approach would lead to favorable outcome only when all interfaces have similar bit rates, since otherwise the faster interfaces would become idle waiting for the slower interfaces to finish downloading their assigned parts. Intuitively, one can see that an optimal solution always utilizes all interfaces to their full bit rate potential throughout the download session. Another solution would be to assign each interface a portion of the content that is proportional to its estimated average bit rate in order to balance the download time durations on average, e.g., this is assumed in previous related works with two interfaces [20], [7]. Even though this leads to enhanced download bit rate, it can deviate from the optimal solution for scenarios with high bit rate fluctuations around the mean over time, e.g., in network scenarios with varying load and mobility. Moreover, this requires continuous estimation and monitoring of the average bit rate leading to increased complexity and computational overhead.

In this section, we divide the content into two subsections: one focused on heterogeneous network scenarios with two wireless interfaces only (e.g., WiFi and cellular) and the second focused on the more general network scenario with K wireless interfaces. Results with discussions and insights are then presented in Section 6.

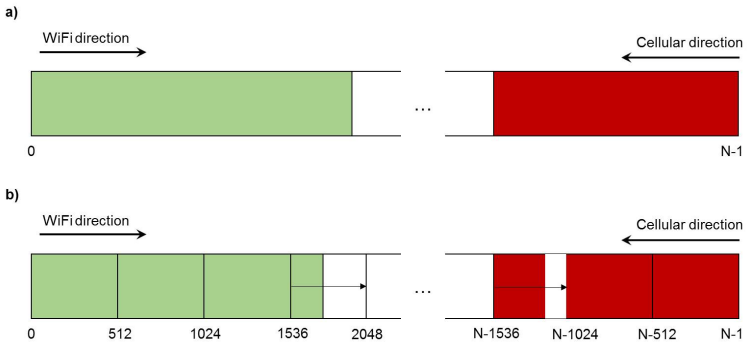


Figure 4: Example of the proposed link aggregation algorithm for file downloading with two interfaces, e.g., WiFi interface and cellular interface.

4.1.1. File Download with Two Wireless Interfaces

For the special case of two wireless network interfaces and for file downloads (i.e., non streaming services), we propose an optimized link aggregation algorithm that is fully device-centric and does not require estimation of link-quality metrics. Our approach is based on downloading the file at the same time on both interfaces, each from one end of the file in opposite directions. For instance, for a WiFi/cellular scenario, the WiFi interface will start the download from byte 0 going up while the cellular interface will start the download in the reverse direction from byte $N-1$ going down (see Fig. 4*a* for an example schematic). The download on both interfaces is interrupted the moment the assembler in the device detects that they downloaded a common byte; at this point, we are certain that all the data have been downloaded and the assembler can fully reconstruct the file. It is clear that the point at which both interfaces meet in the download progress defines the optimal dynamic partitioning of the data into two parts during that download session. Moreover, since both interfaces downloaded the data continuously using their best-effort bit rate without interruption, the file is downloaded in nearly minimum time.

In practice, implementing this strategy on a mobile device is challenging due to the fact that the HTTP standard does not support streaming the data in reverse. Since our goal is to design a fully device-centric approach without any modifications to the content server, we solve this problem by segmenting the data stream into reasonably-sized chunks. This way one interface starts downloading chunks from the beginning of the stream while the other downloads chunks from the end of the stream (see Fig. 4*b* for an example where the data stream is divided into 512-byte chunks). The only complexity that arises from this method is determining a suitable chunk size. A small chunk size leads to a delay and processing overhead from the HTTP requests which can lead to an increase in download time. On the other hand, a chunk size that is large might negatively affect the optimality of the download time as one interface can become idle (after all the chunks have been requested) while the slower interface

is still completing the download of its last chunk. The proposed link aggregation strategy is summarized in Algorithm 1.

In this section, we present an optimized algorithm for file download over K interfaces regardless of the varying network conditions per interface (see Algorithm 2 for the details). It is important to note that Algorithm 1 cannot be applied to cases with $K > 2$ interfaces, since it cannot be known apriori how much data will be downloaded on each interface due to the device centric approach and the varying network conditions over time.

The basic idea of the proposed algorithm is to assign the efficient interfaces that become idle to assist the slower interfaces in downloading their allocated chunks. Consider an example with three interfaces I_1, I_2, and I_3. Initially, the file is divided into three parts, each of length $N/3$, where N is the length of the file in bytes. The scheduler then assigns one part to each interface. Unless the interfaces have identical bit rates, at least one interface would eventually finish downloading its part and become idle while waiting for other parts to finish downloading.

Algorithm 1 Link aggregation algorithm for file downloading using two wireless interfaces.

```
1: Given: Two interface I_1 and I_2
2: Given file size N bytes
3: procedure TwoIfacesSched(N, chunksize)
4:     start = 0
5:     end = N − chunksize
6:     while start < end do
7:         if IsIdle(I_1) then
8:             if start >= end then
9:                 start = end − 1
10:            end if
11:            Schedule(I_1,[start, start + chunksize])
12:            start = start + chunksize + 1
13:        end if
14:
15:        if IsIdle(I_2) then
16:            if end <= start then
17:                end = start + 1
18:            end if
19:            Schedule(I_2,[end, end + chunksize])
20:            end = end − chunksize − 1
21:        end if
22:    end while
23: end procedure
```

4.1.2. **File Download with Multiple Wireless Interfaces**

Fig. 5a shows a scenario where I_3 finishes first while I_1 and I_2 are lagging behind. Rather than waiting, I_3 is assigned to assist the slowest interface (I_1 in Fig. 5a). For that, the scheduler will further divide I_1's remaining part into two equal-length subparts. Let the division point in the stream be byte i. Then, as shown in Fig. 5b, I_1 would continue downloading its part but only up to byte i while I_3 will start downloading the second subpart starting at byte $i+1$. This process continues until all the file has been downloaded (see Fig.s 5c and 5d). Ignoring the HTTP request/response RTT delays and lower protocol delays, this solution is guaranteed to lead to a download time that is close to optimal. In Section 5, we present implementation ideas to minimize the impact of the HTTP request/response RTT delay on overall download time.

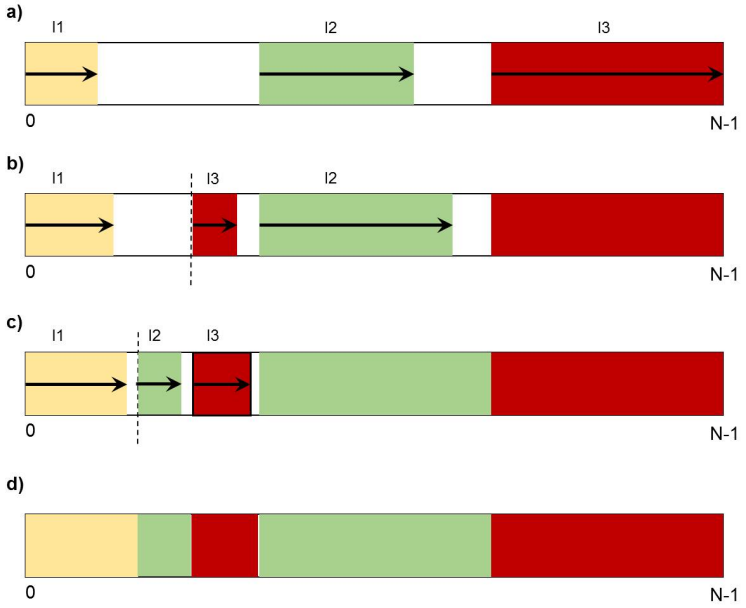


Figure 5: Example of the proposed link aggregation algorithm for file downloading with K interfaces, assuming $K = 3$ (red, green, and yellow colors reflect the data downloaded on interfaces I_3, I_2, and I_1, respectively).

4.2. Link Aggregation for Video Streaming Service Class

For multimedia services such as video streaming, the total download time is not the most important metric in evaluating the quality of service. From the user's point of view, what matters is for the video to start playing as soon as possible and for the playback to be smooth and lag-free. In fact, if we use the multi-interface file download algorithm presented in the previous section for video streaming, we would not be utilizing the full potential of all interfaces to improve the quality of service although the total download time would remain

Algorithm 2 Link aggregation algorithm for file downloading using K wireless interfaces.

```
1: Given K interfaces I_k, 1 ≤ k ≤ K
2: Given file size N bytes
3: procedure NIfacesSched(N, K)
4:     plength = N/K
5:     for k = 1 to K do
6:         chunk = [plength × (k − 1), plength × k − 1]
7:         Schedule(I_k,chunk)
8:     end for
9:
10:    while not AllIdle() do
11:        for k = 1 to K do
12:            if IsIdle(I_k) then
13:                j = GetSlowestInterface()
14:                [s, e] = GetPart(I_j)
15:                m = ⌊(s + e)/2⌋
16:                chunk_1 = [s, m]
17:                chunk_2 = [m + 1, e]
18:                Schedule(I_j, chunk_1)
19:                Schedule(I_k,chunk_2)
20:            end if
21:        end for
22:    end while
23: end procedure
```

minimum. This is since some interfaces would be downloading parts of the file that are not essential to ensure continuous playback in the very near future; video frames that reside at the beginning of the stream must be prioritized over frames that belong to the end of the stream.

A scheduling algorithm optimized for video streaming would focus first on downloading the content parts that contain the subsequent frames essential for non-interrupted playback. In general, the design of link aggregation algorithms for video streaming applications is challenging due to the strict quality of service requirements of video traffic. The following are some challenges that need to be taken into account during the design process:

- At the beginning of the download session, the scheduler does not have any performance metrics to use in choosing the correct interfaces to download the first few video chunks needed to launch the video playback.

- At some point during the download session, a slow interface could be available while faster ones could be busy. If the scheduler chose to assign a chunk to the slow interface, it might lead to stalls in the video playback in the near future. On the other hand, if the scheduler chose to postpone

scheduling the chunk, the slow interface would sit idle rather than being fully utilized.

- Several factors heavily affect the bit rate in wireless networks including mobility, interference, and load. Therefore, it is common for the bit rate to fluctuate notably even in static scenarios. While at some point the scheduler might have chosen the fastest interface to download a video chunk, the interface's performance might easily drop while still downloading that chunk leading to stall events.

One possible link aggregation strategy would be to segment the video stream into several reasonably-sized chunks. The scheduler would then assign the next chunk in the stream, sequentially, to the first available interface without performing any link quality estimation; this continues till all chunks have been downloaded. Similar to the file downloading case, the chunk size would then be an important parameter to be optimized since small sizes can lead to request-response overhead delays and large sizes can lead to lower effective download rate and higher underflow time intervals, which would impact negatively the video's quality of experience. Since the scheduler does not have prior knowledge of the capability of each interface and since link conditions vary over time, an urgently-needed chunk could easily be assigned to an under-performing interface; in this case, the video would stall although subsequent chunks might have been already downloaded.

Therefore, we propose a dynamic device-centric link aggregation algorithm for video streaming that is based on real-time estimation of the effective download bit rate over the various available wireless interfaces using available performance metrics, which are logged over time; the link quality prediction accuracy is more likely to be accurate over reasonably short time intervals. To this end, we divide the streaming process into S time slots of fixed length Δt. The length of the time slot is set to the initial buffering time, which corresponds to a short wait time that the user is willing to tolerate before the playback starts. We design a new mechanism that assigns to each interface a chunk size that is proportional to its estimated link quality (see Algorithm 3 for the details). At the start of the download session, the scheduler does not posses any link quality knowledge. To avoid delaying the video playback, we opt not to perform any pre-download link quality estimation. Rather, we chose to assign the first chunk of fixed predetermined length to all interfaces. The first chunk size is chosen to be small enough not to cause too much redundant data to be downloaded. Having a smooth start during a video streaming session is directly linked to better perceived quality of experience by end users. An example of the proposed link aggregation mechanism for video streaming is shown in Fig. 6.

Whenever an interface finishes downloading a chunk, the scheduler will evaluate its performance during the past time interval based on locally logged information, which includes downloaded data size and download time duration. The scheduler would then calculate the size of the next chunk to assign to interface

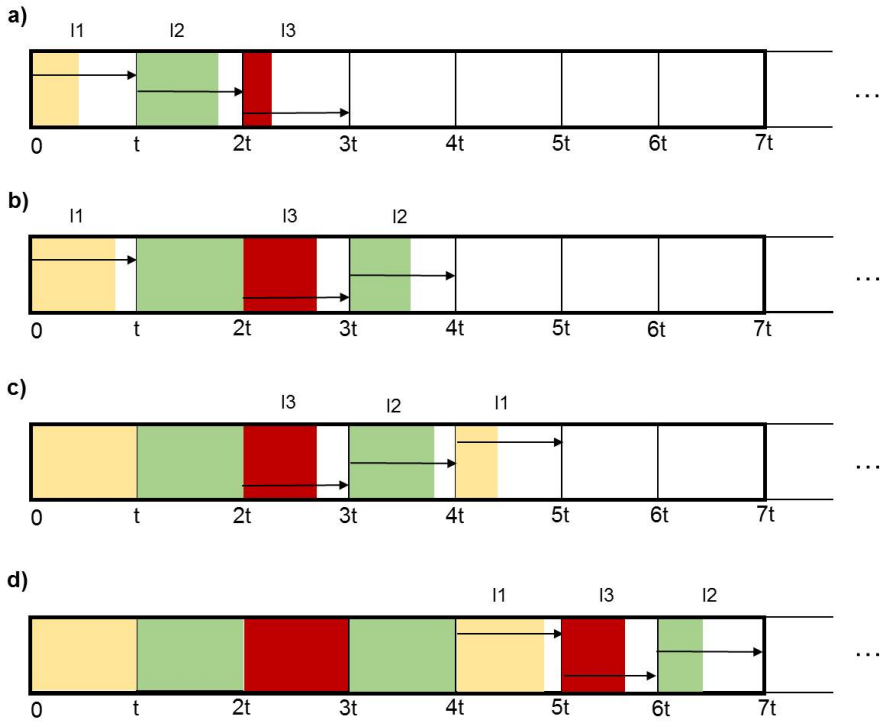


Figure 6: Example of the proposed link aggregation algorithm for video streaming with K interfaces, assuming $K = 3$ (red, green, and yellow colors reflect the data downloaded on interfaces I_3, I_2, and I_1, respectively).

k during time slot t as follows:

$$L_k(t) = Q_k(t-1) \cdot \Delta t, \tag{5}$$

where $Q_k(t-1)$ is the measured bit rate of interface k during time slot $t-1$ and Δt is the time slot duration. The interface starts downloading the chunk but may not receive it completely in Δ_t time due to possible decrease in its current bit rate. In this case, a faster interface will join in to help in downloading the remaining data of the assigned chunk.

5. Testbed Implementation

Since the current generation of mobile devices (smartphones, tablets) typically supports only two long range interfaces (3G/4G cellular and WiFi), our testbed implementation on Android smartphones is limited to the link aggregation algorithms with two interfaces supporting both file downloading and video streaming applications. To evaluate the performance in scenarios with more than two interfaces, we implemented an emulator to test the proposed

Algorithm 3 Link aggregation algorithm for video streaming using K wireless interfaces.

```
1: Given K interfaces I_k, 1 ≤ k ≤ K
2: Given file size N bytes
3: Given slot duration Δt
4: procedure NIfacesStreamingSched(N, K, Δt)
5:     offset = 0
6:     for k to K do
7:         chunk = [offset, 256000]
8:         Schedule(I_k,chunk)
9:     end for
10:    offset = offset + 256001
11:
12:    while not AllIdle() do
13:        for k to K do
14:            if IsIdle(I_k) then
15:                p_k = GetPerformance(k)
16:                size = p_k · Δt
17:                chunk = [offset, size]
18:                Schedule(I_k,chunk)
19:                offset = offset + size + 1
20:            end if
21:        end for
22:    end while
23: end procedure
```

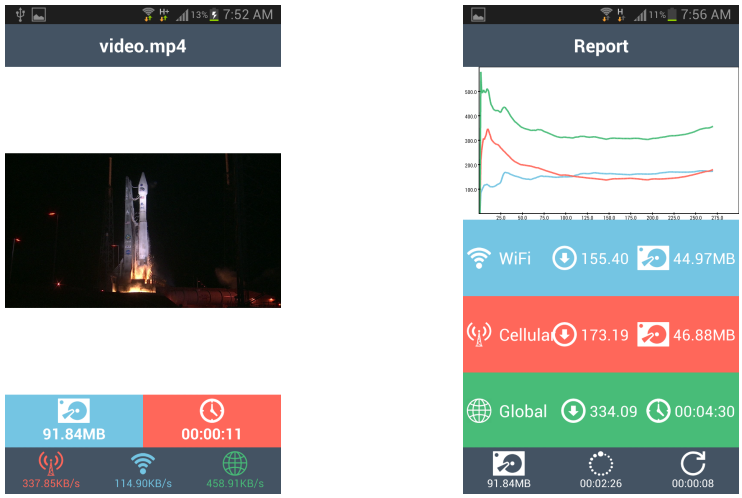
algorithms on an arbitrary number of interfaces, while still mimicking real-life practical settings. For both implementations, the system architecture follows the general design shown in Fig. 3.

5.1. Implementation on Android Devices with Two Interfaces

We designed and implemented a two-interface link aggregation mobile application for Android devices with complete device-centric operation on standard smartphones. The authors in [28] attempted to implement a device-centric link aggregation mobile application for Android devices; however, they developed an application that runs on an Android emulator on a virtual machine with support to two WiFi interfaces via two network cards connected to the machine and, thus, their implementation does not support WiFi/cellular link aggregation on smartphones.

Our mobile application takes an HTTP URL address as input and executes the algorithms presented in Section 4 to download/stream the content simultaneously over both WiFi and cellular interfaces. The application processes one HTTP request for a given content from a web server at a time. Upon receiving a request, the first step is to query the server for the content parameters.

The most important parameters are the "Content-length" and "Content-type" header fields, which specify the data size and its type, respectively. Using the type information, the application decides whether to download the content using the file downloading or video streaming algorithms. For video streaming services, extra header fields such as the duration, number of frames, and frame rate are also parsed from the file headers. Fig. 7 shows two selected screenshots from the developed mobile application during a video streaming session with the proposed WiFi/cellular link aggregation.



(a) (b)

Figure 7: Screenshots from the developed Android mobile application for WiFi/cellular link aggregation. (a) Video streaming view showing video player in addition to total size, elapsed time, and download speed on each interface in addition to aggregation; (b) Report showing download rate variations over the total download duration in addition to summary statistics.

The scheduler takes care of dividing the data stream into chunks depending on the algorithm implementation (fixed chunk sizes for file downloading applications, fixed or variable chunk sizes for video streaming applications). The scheduler monitors the state of two download threads each with a socket bound to one of the interfaces (WiFi and cellular). Whenever an interface is idle, the scheduler feeds it one chunk to download. For file downloading, the scheduler always chooses the first unprocessed chunk in the chunks list for download over the WiFi thread, whereas the last unprocessed chunk is chosen for download over the cellular thread. The downloaded data is continuously supplied to the assembler, which takes care of reconstructing the data stream in correct order for file downloading and playing the video frames in real time for video streaming.

The application takes any HTTP URL and either downloads or streams the content, while using both interfaces according to the details discussed in Section 4. The most challenging part in the implementation was overcoming Android's networking policy that allows only one interface to be active at a time with priority given always to WiFi when both networks are available and active. Eventually, we were able to force-enable both interfaces simultaneously through an undocumented function that raises the mobile interface for a short period of time while WiFi is still active. A simple timer calls this function every period of time to keep the mobile interface up. Having both interfaces up, we added a new route to the HTTP server through the mobile interface by modifying the device's routing table.

Next, two download threads are launched, one for each interface. A socket in each thread will bind the IP address of its respective interface. Any communication through that socket will go through that specific interface. The download threads share a single chunk list. The threads will iteratively consume chunks from the list in the order specified by the proposed link aggregation mechanism. When a chunk is downloaded, it is queued for assembly. For file downloading, the assembler thread polls downloaded chunks from the queue and writes them to a file in the correct order. For video streaming, Android's default video view connects over HTTP to a local proxy running on the device. The proxy fetches the video headers and feeds them to the video view. The download over both threads then proceeds normally. In this case, however, the assembler writes the downloaded chunks to the local proxy that will tunnel them to the video view rather than writing them into a file.

A main performance setback is HTTP request/response RTT overhead associated with the frequent per-chunk HTTP requests issued by the download threads. A 100 MB file would need around 390 chunks to fully download using a chunk size of 512 KB. Suppose the link latency is around 100 ms, then the total overhead would be around $100 \times 390 = 39000$ ms or 39 seconds. This overhead will notably diminish the benefits of link aggregation over two interfaces. We addressed this problem using the following implementation: rather than maintaining a single HTTP connection with the server, each download thread maintains two concurrent and persistent HTTP connections. At any time, the thread would be downloading a certain chunk through one "main" connection and ready to issue GET requests over the second "supplementary" connection. The scheduler, already monitoring the download progress of each interface, would assign the next chunk to the interface right before it finishes downloading the current chunk. When the chunk is received by the download thread, its GET request is issued over the supplementary connection but the body of the response is not read until the current chunk finishes downloading.

5.2. Emulator Implementation for Mobile Devices with Multiple Interfaces

To evaluate the performance of the proposed multiple-interfaces link aggregation algorithms, we implemented an emulator that will mimic the behavior of a mobile device with multiple wireless interfaces. The idea is to have the content hosted, locally, on the same machine running the emulator in order to

exploit the high bandwidth provided by the local loop-back interface. Through this, we gain access to a network link between the emulator and the web server that is bounded only by the hard-disk's read speed.

The emulator is supplied as input a configuration file describing each of the "virtual" interfaces. The description includes the interface name in addition to the bit rate profile over time. Given the file or video HTTP URL (hosted locally), the emulator proceeds with the download using the scheduling mechanisms presented in Section 4.1.2 for file downloading and in Section 4.2 for video streaming. Just like the Android implementation version, the emulator would launch a download thread for each of the K interfaces, and the scheduler would again feed the chunks to the download thread according to the scheduling algorithm in effect. Moreover, an assembler thread would also be running in parallel to receive the downloaded chunks from the download threads and reconstruct the data stream. The only difference here is that the download rate of each download thread (read rate from the network socket), is intentionally throttled to mimic the download behavior of a real wireless interface as defined in the configuration file.

After each network-read operation, each download thread will update its own (throttled) bit rate. The emulator supports two modes for bit rate emulation: arbitrary or measured. In arbitrary mode, a download thread bound to a virtual interface k will always update its rate using an arbitrary value picked from the range $[\text{avg}_k - \text{dev}_k, \text{avg}_k + \text{dev}_k]$ where avg_k and dev_k are the average bit rate and deviation around it for interface k, respectively, as supplied in the configuration file. On the other hand, in measured mode, each interface will update its bit rate from a sequential list of actual measured instantaneous bit rates; these have been obtained by running typical download sessions on a smartphone and recording the instantaneous bit rate over time.

The emulator facilitates the evaluation of the proposed link aggregation algorithms in a controlled environment, which is necessary for reliable comparisons between different design options. We have validated the accuracy of our emulator design by comparing its performance against the Android based testbed implementation using experiments with two wireless interfaces. To this end, we measure the download time for a number of files of varying lengths downloaded using each interface alone, and downloaded via both interfaces jointly using the algorithms described in Section 4, with a fixed chunk size of 128 KB. During each download session, the instantaneous bit rate values of each interface are continuously recorded. We then ran the same experiments on the emulator using the recorded bit rate values from the testbed runs. We summarize the results in Table 1.

It can be seen that the download times obtained using the emulator are very close to the download times recorded using the testbed on the device for all cases, with error percentage less than 0.5%. We can also note that the aggregated bit rate measured when downloading over both interfaces is nearly equal to the sum of the bit rates of the individual interfaces; this clearly demonstrates the effectiveness of the proposed link aggregation algorithms and their implementation under realistic operational conditions.

Table 1: Download time on the device versus the emulator.

File size (MB)	Interface(s)	Average rate (KB/s)	Device time (s)	Emulator time (s)	Error (%)
50	**WiFi**	351.9	149	148	0.006
	Cellular	1191.6	44	40	0.09
	WiFi + Cellular	1417.0	37	35	0.05
100	**WiFi**	351.9	298	296	0.006
	Cellular	1018.0	103	103	0
	WiFi + Cellular	1476.9	71	70	0.014
200	**WiFi**	353.7	593	579	0.023
	Cellular	1048.6	200	195	0.025
	WiFi + Cellular	1344.3	156	154	0.012

6. Experimental Results and Analysis

We ran extensive experiments and recorded various performance metrics, that include network-related metrics such as download bit rates (WiFi alone, cellular alone, and combined WiFi/3G) in addition to user-centric metrics such as download time, download effective rate, startup delay, and buffer underflow time, as applicable. For results with two wireless interfaces, we used our testbed implementation on an Android smartphone; for results with multiple wireless interfaces, we used our emulator implementation with measured bit rate traces.

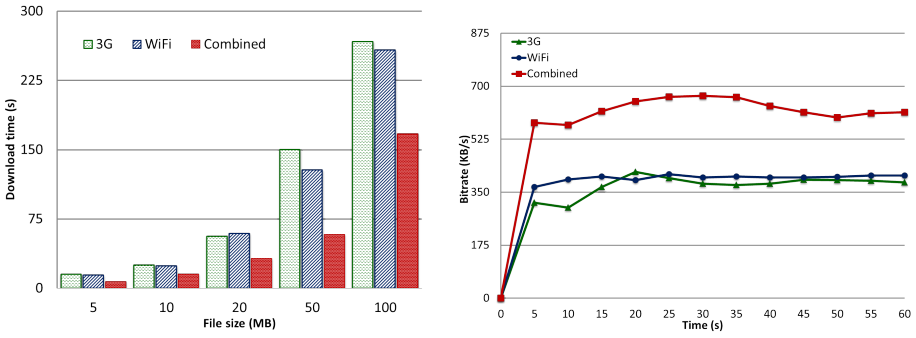
There are a number of software solutions that download data over multi-threaded applications in order to boost the download speed. Our proposed mechanisms can certainly benefit from those solutions and lead to yet better performance. Multi-threaded solutions, however, are always constrained by the available bandwidth of the current wireless interface that is being used by the application and not benefiting from co-existing wireless technologies supported by the same device. We consider DAP [34] and ADM [35] as two example software solutions for mulit-threaded file downloading and compare their performance as opposed to benefiting from multiple interfaces. In Table 2, we show the average download time of a 512 MB file over three consecutive downloads when using each of the example applications and our approach (denoted as M-HetNet). We downloaded a 512 MB file using 15 Mbps WiFi link using one, two, four, and eight threads in ADM and DAP. We compare the resulting download time to using one, two, four, and eight interfaces if available. Despite the fact that additional interfaces have considerably lower bit rates, adding more interfaces lead to better results in terms of reducing the total download time. The multi-threaded software solutions achieved slight improvements when the number of threads increased from two to four. No further improvements are recorded upon increasing the number of threads to eight. In the case of multiple interfaces, significant improvements are attained with the addition of new interfaces even much slower interfaces. With eight interfaces, the download time dropped to around 100 seconds while both of DAP and ADM had a download time of around 300 seconds each.

Table 2: Download time in seconds of 512 MB file using ADM, DAP and M-HetNet.

threads/interfaces	1	2	4	8
ADM	411	305	301	306
DAP	309	314	304	300
M-HetNet	312	208	135	102
Bit rate of added interface(s) (Mbps)	$I_1 = 15$	$I_2 = 7.5$	$I_3 = 7.5$, $I_4 = 5$	$I_5 = 5$, $I_6 = 3$ $I_7 = 3$, $I_8 = 1.5$

6.1. Results for File Downloading with Two Interfaces

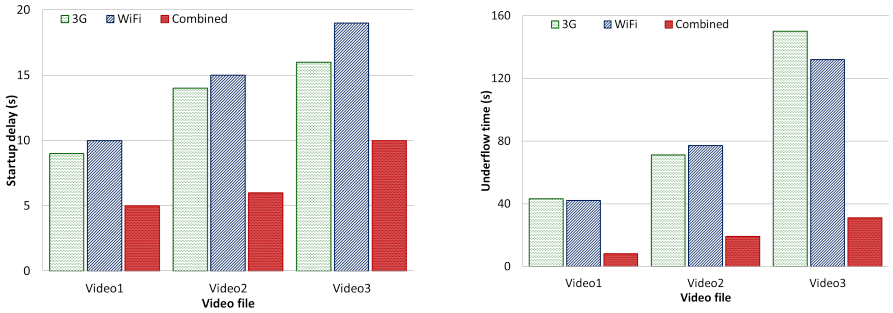
Fig. 8(a) presents measurement results for the download time in seconds for downloading a file from a remote web server as a function of the file size; the experiment was conducted by downloading each file over cellular 3G alone, over WiFi alone, and with WiFi/3G link aggregation (denoted as combined in the figure). Fig. 8(b) presents the instantaneous bit rate variation over a selected 60 s download duration for the WiFi interface alone, 3G interface alone, and aggregated WiFi/3G. These results demonstrate the effectiveness of the proposed link aggregation mechanism with significant performance gains, e.g., for a file size of 50 MB, the download time is 150 s over 3G alone, 125 s over WiFi alone, and reduced to 60 s with WiFi/cellular link aggregation.



(a) (b)

Figure 8: (a) Total file download time measurements with 3G cellular, WiFi, and combined WiFi/cellular link aggregation as a function of the file size. Chunk size is set to 1024 KB. (b) Measured instantaneous bit rates in KB/s during download duration of 60 s each with WiFi interface alone, 3G interface alone, and WiFi/3G link aggregation.

We would like to emphasize the following observations on the performance of the proposed algorithm: i. it is fully device centric as gains are achieved with only a mobile application running at the device level, and without any link quality estimation, server modification, or network level support; ii. it is self adaptive with the individual bit rate variations over time; iii. the WiFi/cellular



(a) (b)

Figure 9: (a) Measured video startup delay while streaming the three video files over WiFi alone, 3G alone, and combined WiFi/3G (link aggregation) (b) Measured video underflow delay while streaming the three video files over WiFi alone, 3G alone, and combined WiFi/3G (link aggregation).

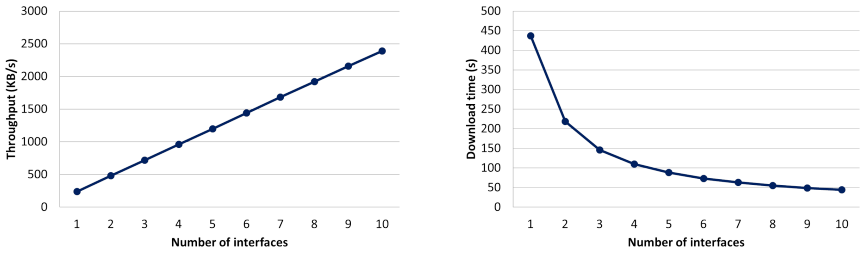
combined bit rate is not simply equal to the sum of the bit rates on the individual interfaces due to variations from lower layer protocols that are transparent to our implementation; iv. the obtained per-device gains map to direct network level gains due to the achieved traffic offloading and load balancing between the cellular and WiFi networks.

6.2. Results for Video Streaming with Two Interfaces

In this section, we present experimental performance results for video streaming with WiFi/cellular link aggregation. We experiment with three video files encoded at different bit rates with different sizes (Video 1: 40 MB, Video 2: 54 MB, Video 3: 67 MB). We collect measurements on the video stratup delay and buffer underflow. Startup delay represents the buffering time needed before playback starts while buffer underflow represents the time when the video player reads from an empty buffer causing the video to stall until more data arrives. Fig. 9(a) shows the video startup delay untill 5 s of the video content have been downloaded; after which, the video play back will be initiated. As expected, the video that is encoded at a higher rate requires more time to fill 5 s into the playback buffer in the device. Fig. 9(b) shows the total underflow time for each of the three video files. These results demonstrate the gains of the proposed WiFi/cellular link aggregation algorithm in enhancing the quality of experience for video streaming applications; the gains are significant in terms of both reduction of startup delay and buffer underflow.

6.3. Results for File Downloading with Multiple Interfaces

To analyze the performance of the proposed multi-interface link aggregation mechanism for file downloading described in Section 4.1.2, we downloaded a 100 MB file using up to 10 interfaces and recorded the aggregated bit rate and total download time. First, we configured the interfaces to use the same



(a) Throughput (b) Download time

Figure 10: Throughput and download time for a 100MB file using up to 10 interfaces with similar bit rates.

instantaneous bit rate logs measured from download sessions over the same WiFi connection. The average bit rate of the measured connection was around 240 KB/s. As shown in Fig. 10a, the aggregated throughput is growing in a linear fashion with the number of interfaces. Moreover, the download time, as shown in Fig. 10b, is decreasing at a constant rate as we increase the number of interfaces.

In order to evaluate the applicability of our algorithm to wireless interfaces with varying bit rates, we measured and logged the instantaneous bit rate of 10 wireless connections with different network quality conditions. The average bit rate of each measured connection is summarized in Table 3. Moreover, the bit rate traces for interfaces I_1 and I_2 are shown, as examples, in Fig. 11, where I_1 is varying more arbitrarily over time compared to I_2.

Table 3: Average bit rates of 10 interfaces used in the emulator.

Interface	I_1	I_2	I_3	I_4	I_5	I_6	I_7	I_8	I_9	I_{10}
Bit rate average (KB/s)	1018	328	943	1153	750	240	255	118	78	135

We ran a series of experiments, each time adding the next interface in Table 3 to the download session, while recording the total effective bit rate and download time. That is, when the experiment requires K interfaces, we use the rate traces for interfaces $I_1, I_2, ..., I_K$ from Table 3. As a benchmark, we compare our algorithm against the case where the file is divided into equal chunks over all the interfaces.

Fig. 12b shows that the download time using the equal-chunks algorithm is very unpredictable; the trend of the download time depends on whether we are adding a faster or slower interface to the available pool of interfaces, since fast interfaces would not be utilized after fetching their assigned chunks and would stay idle while waiting for the slower interfaces to finish. Our algorithm eliminates this problem by allowing the efficient interfaces that become idle to assist the lagging ones. Optimally, we expect that adding a new interface to the download session will lower the download time by a percentage that is

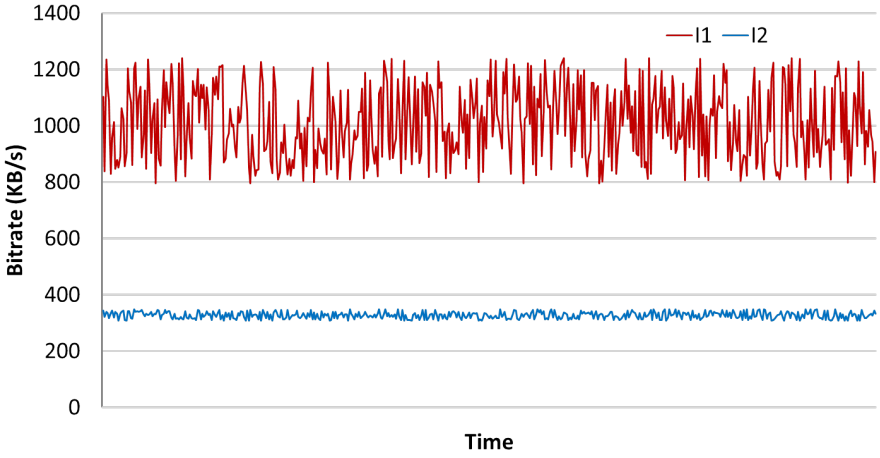


Figure 11: Bit rate traces of interfaces I_1 and I_2 over a given time window.

proportional to its bit rate. More specifically, given a file of length N bytes and given K interfaces, the total download time can be estimated as follows:

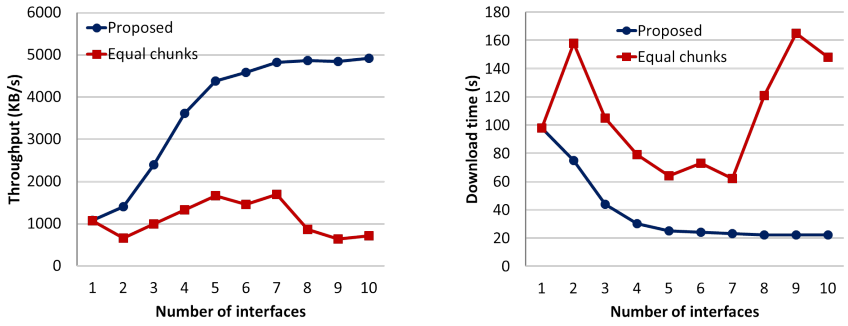
$$T(I_1, I_2, ..., I_K) = \frac{N}{\sum_{k=1}^{K} R_k} \tag{6}$$

where R_k is the average bit rate in KB/s of interface I_k. Fig. 12b shows that our multi-interface link aggregation algorithm for file downloading behaves close-to-optimal with throughput and download time improving proportional to each added interface quality. The download time constantly decreased as we added more interfaces to the session, till the computational and processing overhead exceeded the bit rate boost provided by the extra interfaces; this overhead is associated with activities such as managing and monitoring the interfaces, calculating chunk sizes, and executing the implemented link aggregation algorithm.

6.4. Results for Video Streaming with Multiple Interfaces

The multi-interface link aggregation algorithm for video streaming requires link quality estimation. Whenever an interface is idle, the scheduler uses logged performance metrics to calculate a bit rate estimate for the next time slot. Using the estimated bit rate, the scheduler then calculates the size of the chunk that must be assigned to the interface based on a fixed time slot duration Δt. We calculate the bit rate estimate by dividing the size of the total data streamed over the total time elapsed since the beginning of the streaming session over the given interface; this value is updated at the end of every chunk download.

For our video streaming experiments, we chose two video files trans-coded from the same video source but at two different resolutions: 2K (2560x1440) and 4K (3840x2160). The parameters of the two video files are shown in Table 4. The video startup delay is defined as the time needed till a given duration (initial



(a) Throughput (b) Download time

Figure 12: Effective bitrate and download time for a 100MB file using up to 10 interfaces with different bitrates

Table 4: Properties of the video files used for testing.

File	**Size (MB)**	**Format**	**Duration (s)**	**Video**			
				Codec	**Resolution**	**Bit rate (kbps)**	**Frame rate (fps)**
2k.mp4 (2K)	100	MP4	96	H264	2560x1440	8722	29
4k.mp4 (4K)	225	MP4	96	H264	3840x2160	19474	29

Table 5: Average bitrates of the 5 interfaces

Interface	I_1	I_2	I_3	I_4	I_5
Bitrate (KB/s)	535	231	118	85	64

buffering) of the video content has been downloaded to the device, after which, the video playing will be initiated. The underflow time corresponds to the total time the video player is reading from an empty buffer, which is an indicator of video stalling events and, thus, an indicator of end-user perceived quality of experience [36].

To better capture the performance when slower interfaces are available, we run experiments using five interfaces with average bit rates given in Table 5, in addition to experiments with the higher quality 10 interfaces from Table 3. In addition, we vary a wide range of parameters to capture their impact on performance; this provided useful insights on the potential gains of link aggregation in heterogeneous wireless networks.

Fig. 13 and Fig. 14 present the startup delay in msec versus the number of jointly utilized wireless interfaces for different design parameters. Results show that startup delay can be high for high quality video with relatively long initial buffering requirement (see Fig. 13(a)), whereas it decreases significantly for video content with lower resolution (see Fig. 13(b)). In addition, the bit rate quality characteristics of the available wireless interfaces impact performance notably as shown in the comparisons between three sets of interface parameters in Fig. 14. As interface quality improves, startup delay significantly reduces

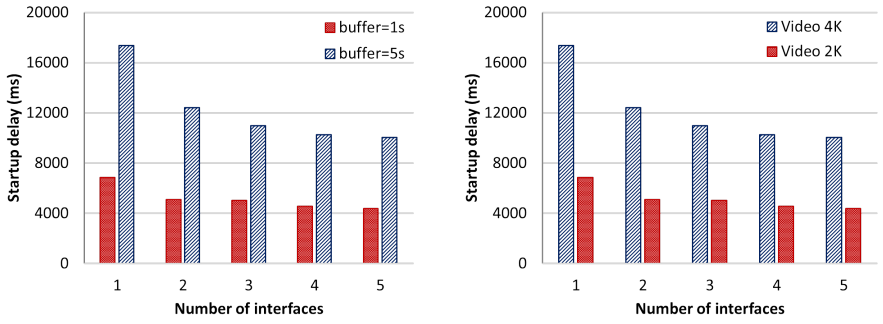


Figure 13: (a) Startup delay versus number of interfaces with different initial buffering durations (Video: 4K; $\Delta t = 2$ sec; Interfaces: from Table 5). (b) Startup delay versus number of interfaces with different video qualities (Initial buffering: 5 sec; $\Delta t = 2$ sec; Interfaces: from Table 5).

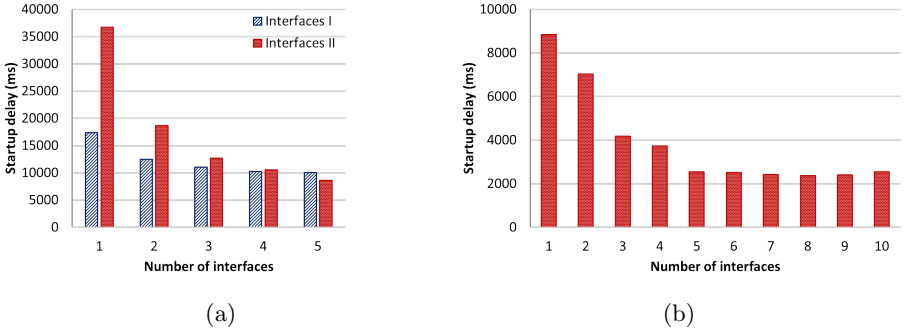


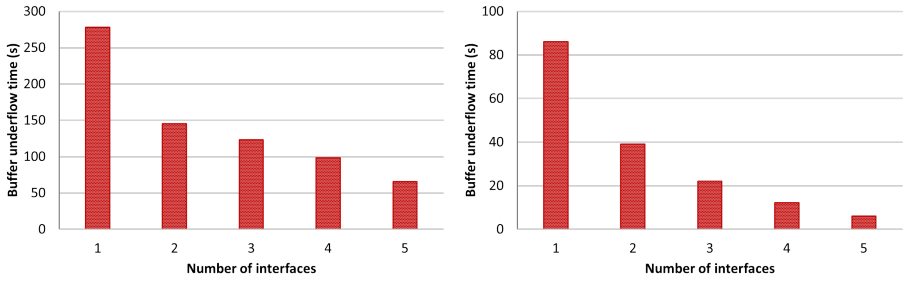
Figure 14: (a) Startup delay versus number of interfaces with different interface bit rates (Video: 4K; Initial buffering: 5 sec; $\Delta t = 2$ sec; Interfaces 1: from Table 5; Interfaces 2: All same trace with average 251 KB/s). (b) Startup delay versus number of interfaces (Video: 4K; Initial buffering: 5 sec; $\Delta t = 2$ sec; Interfaces: from Table 3).

leading to improved quality of experience.

Both figures also demonstrate the gains of utilizing multiple interfaces on performance; the more interfaces, the better, however, there is a limit after which steady state performance is achieved. This indicates that in practice, utilizing a few interfaces can lead to most of the possible gain. In case there are more interfaces available, then one can implement interface selection intelligence to choose the best subset while balancing tradeoff between performance gains and complexity or cost implications.

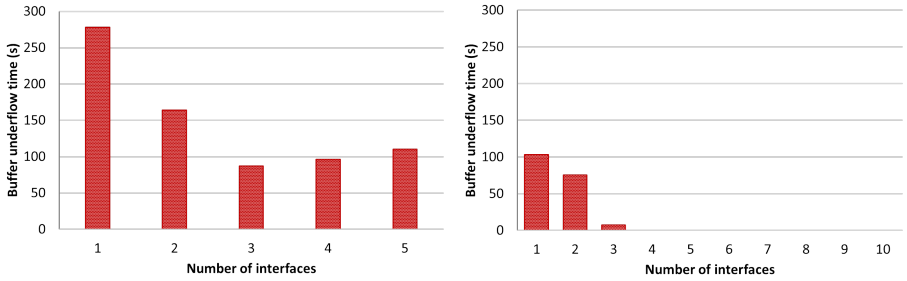
Fig. 15 and Fig. 16 present the obtained results for the video buffer underflow parameters in seconds as a function of the number of jointly utilized wireless interfaces with the proposed link aggregation algorithm. Similar performance trends as for the startup delay results can be observed. For example, for the 4K video, the total buffer underflow is reduced from around 280 sec to around

100 sec when using four interfaces instead of one (see Fig. 15(a)), and it goes down to around 5 sec for the 2K video (see Fig. 15(b)); similar performance improvement is also achieved for the 4K video when three high rate interfaces are used as shown in Fig. 16(b). Finally, comparing Fig. 15(a) to Fig. 16(a), one can quantify the impact of the slot duration (Δt) on performance, where it is shown that for this case $\Delta t = 2$ sec achieves better results than $\Delta t = 5$ sec.



(a) (b)

Figure 15: (a) Total buffer underflow time versus number of interfaces (Video: 4K; Initial buffering: 5 sec; $\Delta t = 2$ sec; Interfaces: from Table 5). (b) Total buffer underflow time versus number of interfaces (Video: 2K; Initial buffering: 5 sec; $\Delta t = 2$ sec; Interfaces: from Table 5).



(a) (b)

Figure 16: (a) Total buffer underflow time versus number of interfaces (Video: 4K; Initial buffering: 5 sec; $\Delta t = 5$ sec; Interfaces: from Table 5). (b) Total buffer underflow time versus number of interfaces (Video: 4K; Initial buffering: 5 sec; $\Delta t = 2$ sec; Interfaces: from Table 3).

The total underflow time is an accumulation of all the pause durations that took place during the complete video streaming session. In order to analyze the pause durations in more detail, we present in Fig. 17 the probability that the pause duration is bigger than the value given on the x-axis in seconds; this corresponds to the complementary cumulative distribution function of the pause duration. In this figure, we also compare the performance of our proposed algorithm to the following streaming strategies: i. one interface and, thus, no link aggregation; ii. link aggregation with fixed video chunk size equal to 256 KB; iii. link aggregation with fixed video chunk size equal to 1024 KB. These results

demonstrate the superiority of the device-centric dynamic algorithm proposed in this work for video streaming applications; e.g., the probability of a pause duration above 3 sec is 0 for our algorithm, compared to 77% for using one interface and 29% for fixed chunks with size 1024 KB.

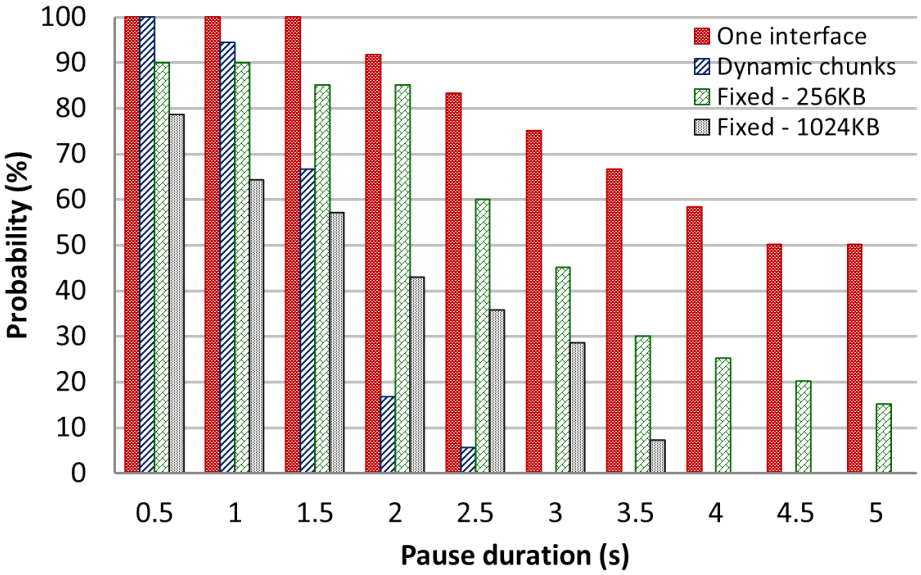


Figure 17: Probability of pause duration more than x seconds using the proposed algorithm (dynamic chunks) versus no link aggregation (one interface) and link aggregation with fixed chunk sizes (Video: 2K; Initial buffering: 5 sec; $\Delta t = 2$ sec; Interfaces: from Table 5).

7. Conclusions

In this work, we presented the design and implementation of optimized device centric link aggregation mechanisms for mobile devices with multiple wireless interfaces. The proposed mechanisms are customized to two different classes of mobile applications, elastic file downloading and real-time video streaming. Moreover, they apply to scenarios with two or more wireless interfaces at the mobile device, reflecting emerging use cases with high end smartphones/tablets or clusters of devices cooperating to download a given content. The novelty of the proposed mechanisms is the seamless operation without the need for centralized link quality estimation, networking protocol stack changes, proxy server implementation, or support from the cellular base stations or WiFi access points. We presented detailed performance evaluation using a developed Android mobile application for use cases with two interfaces (WiFi/cellular link aggregation) and using an accurate emulator for use cases with more than two interfaces. The results for both file downloading and video streaming demonstrate the effectiveness of the proposed mechanisms in terms of bit rate enhancement, delay

reduction, and quality of service improvement. Moreover, they demonstrate the feasibility of realizing these gains in state-of-the-art smartphones under realistic operational device constraints and network conditions.

Acknowledgements

This work was made possible by NPRP grant 7-1529-2-555 from the Qatar National Research Fund (a member of The Qatar Foundation). The statements made herein are solely the responsibility of the authors.

References

1. Cisco White Paper . Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 2014 - 2019. [Online - white_paper_c11-520862.pdf], Available: http://www.cisco.com/; 2015.

2. Lee S, Hyeon S, Kim J, Roh H, Lee W. The useful impact of carrier aggregation: A measurement study in south korea for commercial lte-advanced networks. *IEEE Vehicular Technology Magazine* 2017;:55–62.

3. Shen Z, Papasakellariou A, Montojo J, Gerstenberger D, Xu F. Overview of 3gpp lte-advanced carrier aggregation for 4g wireless communications. *IEEE Communications Magazine* 2012;:122–30.

4. Rosa C, Pedersen K, Wang H, Michaelsen P, Barbera S, Malkamaki E, Henttonen T. Dual connectivity for lte small cell evolution: functionality and performance aspects. *IEEE Communications Magazine* 2016;54(6):137–43.

5. Zhang Q, Wang Q, Feng Z, Yang T. Design and performance analysis of a fairness-based license-assisted access andresource scheduling scheme. *IEEE Journal on Selected Areas in Communications* 2016;34(11):2968–80.

6. Ramaboli A, Falowo O, Chan A. Bandwidth aggregation in heterogeneous wireless networks: A survey of current approaches and issues. *Elsevier Journal of Network and Computer Applications* 2012;35(6):1674–90.

7. Habak K, Youssef M, Harras K. An optimal deployable bandwidth aggregation system. *Elsevier Computer Networks* 2013;57(15):3067–80.

8. Jo M, Batista RL, Maciel TF, de Almeida ALF, Klymash M. A survey of converging solutions for heterogeneous mobile networks. *IEEE Wireless Communications* 2014;21(6):54–62.

9. 3GPP TR 37.834 V12.0.0 (2013-12) . Study on Wireless Local Area Network (WLAN) - 3GPP radio interworking (Release 12). [Online], Available: http://www.3gpp.org/DynaReport/37834.htm; 2013.

10. 3GPP TS 23.234 V12.0.0 (2014-09) . 3GPP system to Wireless Local Area Network (WLAN) interworking; System description (Release 12). [Online], Available: http://www.3gpp.org/DynaReport/23234.htm; 2014.

11. Tehrani MN, Uysal M, Yanikomeroglu H. Device-to-device communication in 5G cellular networks: Challenges, solutions, and future directions. *IEEE Communications Magazine* 2014;52(5):86–92.

12. Andreev et al. S. A unifying perspective on proximity-based cellular-assisted mobile social networking. *IEEE Communications Magazine (accepted for publication)* 2016;.

13. Jahed K, Fawaz M, Sharafeddine S. Practical device-centric WiFi/cellular link aggregation mechanism for mobile devices. In: *11th International Conference on Innovations in Information Technology (IIT)*. 2015:.

14. Bennis M, Simsek M, Saad W, Valentin S, Debbah M, Czylwik A. When cellular meets WiFi in wireless small cell networks. *IEEE Communications Magazine* 2013;.

15. Ling J, Kanugovi S, Vasudevan S, Pramod AK. Enhanced capacity and coverage by Wi-Fi LTE integration. *IEEE Communications Magazine* 2015;53(3):165–71.

16. El Helou M, Ibrahim M, Lahoud S, Khawam K, Mezher D, Cousin B. A network-assisted approach for RAT selection in heterogeneous cellular networks. *IEEE Journal on Selected Areas in Communications* 2015;33(6):1055–67.

17. Nguyen-Vuong Q, Agoulmine N, Ghamri-Doudane Y. A user-centric and context-aware solution to interface management and access network selection in heterogeneous wireless environments. *Elsevier Computer Networks* 2008;52(18):3358–72.

18. Chamodrakas I, Martakos D. A utility-based fuzzy TOPSIS method for energy efficient network selection in heterogeneous wireless networks. *Elsevier Applied Soft Computing* 2011;11(4):3734–43.

19. El Helou M, Lahoud S, Ibrahim M, Khawam K, Cousin B, Mezher D. A hybrid approach for radio access technology selection in heterogeneous wireless networks. *Springer Wireless Personal Communications* 2016;86(2):789–834.

20. Kim J. Feedback-based traffic splitting for wireless terminals with multi-radio devices. *IEEE Transactions on Consumer Electronics* 2010;56(2):476–82.

21. Abbas N, Dawy Z, Hajj H, Sharafeddine S. Energy-throughput tradeoffs in cellular/WiFi heterogeneous networks with traffic splitting. In: *IEEE Wireless Communications and Networking Conference (IEEE WCNC'14)*. Istanbul, Turkey; 2014:.

22. Krishna BM, Madhuri S, Sathya V, Tamma BR. A dynamic link aggregation scheme for heterogeneous wireless networks. In: *IEEE CONECCT 2014*. 2014:.

23. Abbas N, Hajj H, Dawy Z, Jahed K, Sharafeddine S. An optimized approach to video traffic splitting in heterogeneous wireless networks with energy and qoe considerations. *Journal of Network and Computer Applications* 2017;83:72–88.

24. Cordero JA. Multi-path TCP performance evaluation in dual-homed (wired/wireless) devices. *Elsevier Journal of Network and Computer Applications* 2016, in press;.

25. Wang J, Wen J, Zhang J, Xiong Z, Han Y. TCP-FIT: An improved TCP algorithm for heterogeneous networks. *Elsevier Journal of Network and Computer Applications* 2016, in press;.

26. Chebrolu K, Rao R. Bandwidth aggregation for real-time applications in heterogeneous wireless networks. *IEEE Transactions on Mobile Computing* 2006;5(4):388–403.

27. Wu J, Cheng B, Wang M, Chen J. Energy-efficient bandwidth aggregation for delay-constrained video over heterogeneous wireless networks. *IEEE Journal on Selected Areas in Communications* 2017;35(1):30–49.

28. Takiguchi T, Hidaka A, Masui H, Sugizaki Y, Mizuno O, Asatani K. A new application-level link aggregation and its implementation on Android terminals. *Wiley Wireless Communications and Mobile Computing* 2012;12(18):1664–71.

29. Habak K, Harras K, Youssef M. Oscar: A deployable adaptive mobile bandwidth sharing and aggregation system. In: *11th International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services (MOBIQUITOUS'14)*. London, UK; 2014:.

30. Dandachi G, Elayoubi S, Chahed T, Chendeb N. Network-centric versus user-centric multihoming strategies in lte/wifi networks. *IEEE Transactions on Vehicular Technology* 2017;66(5):4188–99.

31. Hassan MA, Xiao M, Wei Q, Chen S. Help your mobile applications with fog computing. In: *IEEE SECON Workshops 2015*. 2015:.

32. Rebecchi F, de Amorim MD, Conan V, Passarella A, Bruno R, Conti M. Data offloading techniques in cellular networks: A survey. *IEEE Communications Surveys and Tutorials* 2015;17(2):580–603.

33. IETF RFC 7233 . Hypertext Transfer Protocol (HTTP/1.1): Range Requests. Tech. Rep.; June 2014.

34. RubyCell . Download Accelerator Plus. [Online], Available: http://play.google.com; 2017.

35. DimonVideo . Advanced Download Manager. [Online], Available: http://play.google.com; 2017.

36. Mok RKP, Chan EWW, Chang RKC. Measuring the quality of experience of HTTP video streaming. In: *12th IFIP/IEEE International Symposium on Integrated Network Management and Workshops*. 2011:.