Mobile Live Video Upstreaming

Philip Lundrigan, Mojgan Khaledi, Makito Kano,
Naveen Dasa Subramanyam, and Sneha Kasera
Department of Computer Science
University of Utah
{philipbl, mojgankh, makito, naveends, kasera}@cs.utah.edu

***Abstract*—We design and build a system we call *mobiLivUp*, that utilizes nearby smartphones to improve live wide-area video upstreaming. In mobiLivUp, to distribute the video to nearby devices, the video streaming device creates a small wireless network using Wi-Fi Direct. Other devices then connect to this network. Parts of the video stream are sent to these connected devices, which then upload their parts to a location in the wide-area network using their cellular connections. We develop algorithms and methods to effectively distribute video data to nearby nodes and for incentivizing cooperation from these nodes. We test our system through trace-driven simulation and implementation in various settings. Our experiments show that, in general, mobiLivUp increases the aggregate video throughput, depending on the number of nodes forwarding data and their data rates.**

I. Introduction

Smartphones have opened up new possibilities, from the way we organize our lives, to the way we communicate with each other. With smartphone cameras capable of capturing high definition video, we are able to communicate in more powerful ways. An example of this is live streaming video from a mobile device. Live streaming video allows people to share in an experience together, at the same time. Live streaming has become a popular activity for consumers with the advent of apps like Periscope [1] and Meerkat [2]. In such apps, a user shares a link to their live stream with friends through social media like Facebook or Twitter. Friends can connect and watch the video stream in real-time, communicating with the person live streaming through comments and likes. Comcast has recently announced that it will allow customers to live stream video from their XFINITY Share mobile app to another user's cable box [3]. Another example of a use for live streaming video is live action sports using small recording devices, like the GoPro Hero [4]. Some of these small devices are now equipped with transmitters that can broadcast video to cellular capable devices for uploading in real-time. Live streaming video opens up possibilities to new applications that have not been possible before.

While live video can be upstreamed at a high quality over high bandwidth Wi-Fi links, the same cannot be said about video transmission over cellular links for the following reasons. First, a cellular connection cannot always support the video data rate. Cellular networks have a difficult time keeping up with the demands of smartphone data usage [5] and video streaming perpetuates this problem. Furthermore, cellular networks (as with most networks) have been optimized for fast download speeds, but not upload. Second, spatio-temporal variations in cellular channel conditions can reduce the data rate in unpredictable ways. Even when standing still, the data rate of a cellular connection can vary dramatically [6] [7] over time. This makes it hard for any variable video rate algorithm to work effectively. Third, smartphone video quality is increasing faster than that of cellular connections. Each year, new phones are equipped with better cameras and this is not slowing down. Ultra HD or 4K cameras are already starting to be commonplace in new smartphones [8]. Higher quality video requires a higher cellular data rate.

In this paper, we address the increasingly important problem of upstreaming live video over cellular links in the absence of infrastructure Wi-Fi networks. Specifically, we design and build a system, that we call *mobiLivUp*, that utilizes nearby smartphones and their cellular bandwidth collectively to effectively increase the live upstream bandwidth. Due to the nature of cellular networks, each mobile device is guaranteed some slice of the cellular bandwidth. The use of multiple devices to transmit on the wireless medium in parallel, without interfering, can enhance the aggregate video transmission rate.

mobiLivUp takes advantage of a smartphone's multiple wireless interfaces. In mobiLivUp, to distribute the video to nearby devices, the video streaming device creates a small wireless network, using Wi-Fi Direct [9].[1] Other devices then connect to this Wi-Fi Direct network. Parts of the video stream are sent to these connected devices, which then upload their parts to a location in the wide-area network using their cellular connections. The following advantages are gained by using multiple cellular connections:

i. The total aggregate throughput increases as more cellular connections are used. Each cellular connection contributes its data rate which when combined together, allows the streaming device to stream video as if it had a higher data rate.

ii. The throughput is stabilized as more cellular connections are used. As stated earlier, cellular networks can have highly fluctuating data rates. This effect is minimized by combining multiple cellular connections together. For example, when more cellular connections are used and spatial diversity is higher, the less likely it is for each connection to fluctuate in the same way, causing the aggregate data rate to be more smooth than the individual fluctuating data rates.

[1]Does not require a Wi-Fi infrastructure network.

iii. Using multiple cellular connections allows for cellular provider diversity. When two or more different cellular providers are used, the difference in cell tower placement and network particularities will lead to a more stable video streaming experience. For example, in a given location, one cellular provider might have a dead spot, but another cellular provider might not [10].

Using multiple mobile devices to cooperatively download stored data (but not upload live video) has been studied in prior work. To the best of our knowledge, mobiLivUp is the first system that deals with uploading *live* video data by using multiple neighboring nodes in real-world scenarios.

Cooperative live video uploading presents challenges that have not been addressed in past work. The most critical of these challenges is to ensure that video streaming content traveling through multiple paths arrive at the destination in the correct sequence. Without intervention, the video data will arrive out of sequence, rendering the video unplayable. Receive buffers can be used to relieve this problem to a certain point, but if the receive buffer is too big, the video playback will not be live. Buffering alone can not solve this problem. mobiLivUp uses a novel algorithm to intelligently distribute video packets between cooperating neighboring devices, responding quickly to changes in cellular data rates. Our distribution algorithm also incorporates a mechanism to incentivize devices to forward data. With the incentive model, nodes that participate in mobiLivUp are paid for their usage and everyone benefits as a result.

We evaluate mobiLivUp in three ways. First, we implement mobiLivUp in the ns3 simulator to test and evaluate the method by which we distribute data to forwarding nodes. We also use the simulator to test our incentive model and demonstrate its functionality. Second, we create a wireless testbed which allows us to test the system under controlled conditions, but with more realistic conditions than simulation. Third, we deploy our system while traveling on a commuter train. Our experiments show that, in general, mobiLivUp increases the aggregate video throughput, depending on the number of nodes forwarding data and their data rates. In the case of the commuter train experiment, with two smartphones, mobiLivUp increases the video throughput by up to 88.6% in comparison to that obtained using only one smartphone.

II. Related Work

The idea of using multiple nearby devices in cooperation to share cellular connections has been used for cellular offloading [10] and video downloading. MicroCast [11] is an example of using Wi-Fi or Bluetooth to cooperatively *download* a video using multiple cellular connections. The use cases of our system and MicroCast seem similar, but there are significant differences. The biggest difference is that MicroCast does not deal with live video data. This allows for different parts of the video to be indefinitely buffered by individual phones. MicroCast is concerned about getting the whole video to all devices, which leads to their focus on a dissemination algorithm. Our system is only concerned about getting the video from the video source to a destination. We use other devices as the means to an end, whereas MicroCast uses the devices as clients. Lastly, MicroCast uses a modified wireless driver, but our system uses standard Wi-Fi Direct. This increases the deployability of our system since no changes need to be made to the operating system for it to work. MicroCast also assumes that all peers remain static during the video download, whereas with our system, we do not make that assumption. We assume a setting in which peer nodes can come and go.

MultiPath TCP (MPTCP) [12] allows for data to be transferred by different interfaces on a device. However, MPTCP is limited to the number of interfaces that are on the device, while our system can scale up to the number of devices in close proximity that are willing to cooperate. One could use emulation to add an interface for each wireless peer to allow for normal use of MPTCP. As far as we know, interface emulation for wireless clients has not been implemented on Android and is beyond the scope of this project. In a recent work, Lim et al. [13] use MPTCP in a simulation framework for stored video and file transfer applications, but not for transmitting live video. SCTP [14] is also limited to the number of interfaces on one device and is not necessarily optimized for live video transmission.

Both MPTCP and MicroCast use some form of queuing to schedule which connection should download a specific segment of data. With MPTCP, a shared receive buffer is used between subflows. With MicroCast, each participating device has a backlog of segments to be downloaded. A new segment is assigned to the device with the smallest backlog. In mobiLivUp, data queuing takes place on separate devices and not on the device where the data is coming from. As a result, explicit feedback by each device is required. In mobiLivUp, this feedback is indirectly obtained from the gatherer, and the splitter pushes data to the participating devices at specific rates, adjusting the rates as necessary.

Carrier aggregation [15] is a technology, part of LTE-Advanced, that allows for a device to receive multiple bands, increasing that device's data rate and therefore throughput. We use multiple cellular devices, each receiving their own band, to transmit the data. However, there are two important differences. First, with carrier aggregation, the amount of bands dedicated to one device is controlled by the network and not by the device itself. Our system gives control to the device for the number of bands it wants to use to upload video. Second, our system allows for both spatial diversity and cellular provider diversity. The devices used in our system can be at better radio locations and on different cellular providers, increasing the robustness of our system.

Like our system, Quality-Aware Traffic Offloading (QATO) [10] allows a mobile node to offload its data upload to another nearby mobile node with a better network connection. QATO also suggests using Wi-Fi Direct for transmission of data from the source to the neighboring node. However, QATO only uses one mobile node amongst its neighbors to offload uploading. While this limits the benefits obtained from QATO, QATO does not have to deal with

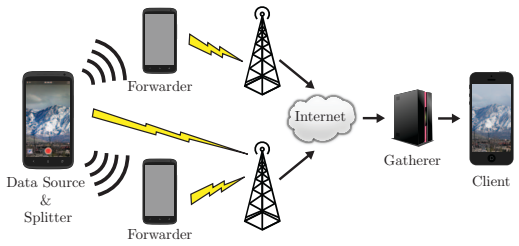


Fig. 1. General architecture of mobiLivUp. Data passes from the splitter to forwarders, then to the gatherer through their cellular connections.

splitting and gathering. Importantly, QATO only experiments with stored data, such as pictures, for uploading. Our system is built for live video transmission.

Link-alike [16] studies the idea of having a wireless device distribute upload data to multiple residential wired connections for a higher aggregate upload rate. However, Link-alike has wired connections. Also, mobiLivUp deals with live video data which is not tolerant to delays, whereas Link-alike deals with general uploaded data which can be buffered as needed.

III. Architecture Components

mobiLivUp consists of four major components: a splitter, a forwarder, a gatherer, and a client. Figure 1 shows how each component connects to the others. A data source (the video streaming application) passes data to the splitter. The splitter broadcasts its availability, allowing nearby peers who are willing to forward data to connect. To keep the splitting transparent to the client, a gatherer server is needed to act as a proxy. The gatherer also collects statistics about each forwarder and sends feedback to the splitter.

Each component is described below in more detail.

A. Splitter

The splitter takes data from the data source and splits it between forwarding nodes. The splitter is started when a user wants to stream video. When the splitter is first started, it connects to the gatherer. The connection between the splitter and gatherer stays open during the life of the splitter, acting as a control channel for the splitter and gatherer. The gatherer sends feedback to the splitter about each forwarder.

Using Wi-Fi Direct [9], forwarding nodes connect to the splitter. Wi-Fi Direct is a technology used primarily by smartphones to connect point to point, without having to be connected to the same wireless network. With mobiLivUp, the splitter acts as an access point allowing forwarding nodes to connect to it.

When a forwarding node connects to the splitter, information is sent to the forwarder, such as the internal IP address, a unique identifier for the forwarding node, and the IP address and port number of the gatherer, allowing the forwarding node to forward data packets from the splitter to the gatherer. A forwarding node in return sends the cost of using it to forward data. This cost metric is explained in the Forwarder section below. The splitter determines which forwarding nodes it will use based on the cost and the data rate that forwarding node can offer, as described in Section V-B. The splitter itself acts as a forwarding node, using its cellular connection to send data to the gatherer as well as using its Wi-Fi interface to distribute data to forwarding nodes. The throughput of each forwarding node is sent as feedback from the gatherer to the splitter. This throughput estimate is end-to-end, taking into account both the Wi-Fi Direct connection and the cellular connection. The feedback loop also incorporates loss, as loss affects the throughput. This means that no matter which wireless interface is the bottleneck, the splitter will be able to adjust accordingly.

The splitter determines how much data to send to each forwarder based on a distribution algorithm which is described in Section V-A.

B. Gatherer

The gatherer, a server running in the cloud, receives data from each of the forwarding nodes and the splitter, combines the data together, and sends it to the client. It makes sure the data packets are in the correct order before sending them to the client. This is a necessary step since the packets can become out of order due to the different paths and network conditions they are traveling. The gatherer collects throughput statistics for each of the forwarding nodes and sends it as feedback to the splitter.

C. Forwarder

A forwarding node looks for a splitter to connect to through Wi-Fi Direct. Once it has found a splitter to connect to and received information about where to forward data, it advertises its cost to the splitter. This cost represents the cost of forwarding the data which affects the user's data plan, bandwidth, and battery life. The cost can adapt with respect to all of these values and change as necessary. The idea is that a forwarder user will charge more if their battery is low or if they have a small amount of data left in their data plan. If the splitter selects the forwarder, it forwards incoming data from the splitter to the gatherer, through its cellular connection. The forwarder is using both of its wireless interfaces at the same time – Wi-Fi to receive data from the splitter and cellular to send data to the gatherer.

IV. Wi-Fi Direct

mobiLivUp uses Wi-Fi Direct to create a local wireless network for the distribution of video data from the splitter to forwarders. It is important to understand the capabilities of this wireless network as it has a significant role in the performance of the system. Wi-Fi Direct allows for an easy configuration of a wireless network by automating the access point setup, authentication, and association, but ultimately, the underlying 802.11 protocol is the same as a traditional Wi-Fi setup (clients connected to an access point).

To test the capability of the Wi-Fi Direct link and smartphones, we send UDP traffic between two Android smartphones at different rates and monitor loss. We find that the loss increases non-linearly as the data rate increases (see Figure 2).

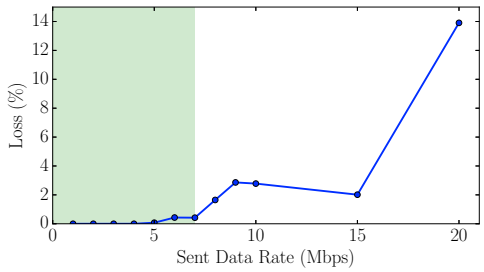


Fig. 2. Loss characteristic as the send data rate increases. The shaded region is the ideal throughput with minimal loss.

We run an Android version of Wireshark on the sender and determine that the loss is not happening at the sender. To rule out losses on the wireless link, we run similar experiments using a laptop sending to another laptop at the same data rates. In these experiments, we observe that there is no loss. We conclude that the losses are happening at the receiver buffer of the smartphone. Based on these findings, 7 Mbps is the highest data rate with reasonable amount of loss (0.42%). While the high receiver loss is an interesting finding, it is not a problem for our system given that cellular upload rates are typically below 7 Mbps [17]. Moreover, the end-to-end feedback loop from splitter to gatherer in our system can account for any bottleneck along the path and the distribution algorithm will adjust appropriately.

V. Distribution Algorithm and Incentive Model

Two important aspects of our system are how to distribute data to forwarding nodes and how to motivate forwarders to participate in our system. Using both of these components, mobiLivUp is able to dynamically select which forwarding nodes to distribute data to and utilize those selected forwarding nodes fully.

A. Distribution Algorithm

As described in the previous section, the splitter takes data from the video source and distributes the packets to itself and other forwarding nodes that were selected based on our incentive model. To suitably distribute video packets among different nodes based on their cellular data rates, we assign a weight[2] to each node. There are different approaches on how to update the weights according to the network conditions. The goal of updating the weights is to maximize the throughput of all of the forwarding nodes, thus maximizing the utility of each node. Because of the dynamic nature of cellular networks, it is not always possible to have the weights set to the optimal values. In this situation, there is a trade-off between being aggressive and reacting too quickly to temporary changes.

In the context of our distribution algorithm, we treat the splitter node as a forwarder node. We assume that the data source will be able to adapt the video to match the aggregate

[2]This weight is a different value than described on the Incentive Model section (V-B).

```
1:  while splitter is running do
2:      receive feedback from gatherer
3:      determine selected forwarders based on feedback
4:      A_total = calculate total actual based on feedback
5:      if A_total ≥ V_max then // Equalize state
6:          j = forwarder with highest actual throughput
7:          k = forwarder with lowest actual throughput
8:          E_j = A_j − Δ
9:          E_k = A_k + Δ
10:         continue
11:     for all i in selected forwarders do
12:         E_i = expected throughput for ith forwarder
13:         A_i = actual throughput for ith forwarder
14:         if A_i ≥ E_i then // Increasing Rate state
15:             E_i = A_i + Δ
16:         else if A_i > A'_i then // Constant Rate state
17:             // Do nothing
18:         else // Reducing Rate state
19:             E_i = A_i ∗ (1 − σ)
20:         A'_i = A_i
```

Fig. 3. Distribution algorithm

throughput of the forwarders, to a certain limit. In other words, if we are only able to send at 1 Mbps, in aggregate, we expect the data source to degrade the video to match it.

Our distribution algorithm is shown in Figure 3. The feedback that the gatherer sends to the splitter contains the actual throughput for each of the forwarders. The splitter also knows the expected data rate of each forwarding node based on the input data rate of the video source and the weight of each forwarding node. The splitter chooses which forwarders to use based on their actual throughput and our incentive model. Let E_i and A_i be the expected throughput and actual throughput for the ith selected forwarding node. mobiLivUp's distribution algorithm determines how to adjust the expected throughput, E_i, updating the weight for that forwarding node, causing the ith forwarder to receive E_i from the data source. The algorithm contains three states: increasing, reducing, and constant. Each state is explained below (see lines 12–20 of Figure 3):

Increase A node is in the *increasing state* when $A_i \geq E_i$. When a forwarding node is in the increase state, its expected throughput gets increased by a small constant value, Δ, such that $E_i = A_i + \Delta$.

Constant A forwarding node is in the *constant state* when $A_i < E_i$ and $A_i > A'_i$, where A'_i is the previous actual throughput of the ith forwarder. In this state, the expected value, E_i, stays the same.

Reduce *Reduce state* is when $A_i < E_i$ in which $E_i = A_i * (1 - \sigma)$, where σ is the decrease factor.

We discuss the exact values of Δ and σ in section VII-A. This algorithm is similar to additive increase, multiplicative decrease (AIMD) technique, but it has the important distinction of containing a third state, constant. This is an important discovery as we implement and test this algorithm. Under certain conditions, a forwarder's throughput can be increasing, but less than E_i ($A_i < E_i$). There are many reasons why this occurs. Some of the factors that affect this are how much the expected throughput value (E_i) is increased by in the increase

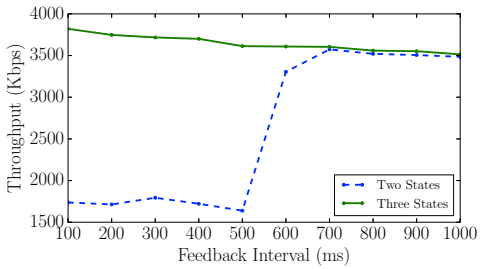


Fig. 4. Comparing using two states (increase and reduce) to three states (increase, reduce, and constant) for different feedback intervals.

state (Δ), how big the window for calculating throughput is at the gatherer, and how often feedback is sent to the splitter from the gatherer (feedback interval). With only two states, a forwarding node's throughput could be increasing such that $A'_i < A_i < E_i$, but still be considered in the reduce state and be incorrectly lessened. The extra state, constant, takes care of this scenario.

Figure 4 compares using a two state AIMD distribution algorithm to the three state distribution for different feedback intervals. We use a cellular trace with three forwarding nodes for both distribution algorithms to calculate the average throughput for a given feedback interval. See section VII for how we collect the cellular trace.

When the feedback interval is below 500 ms, the two state distribution algorithm performs poorly relative to the three state distribution algorithm. This is because the feedback is coming too quickly, not allowing enough time for a forwarding node's throughput to increase to the expected value, causing the distribution algorithm to classify some forwarding nodes in the reduce state. It is only when the feedback interval goes above 500 ms that you seem performance similar to the three state approach. The 500 ms value is specific to this particular cellular trace and can change based on the network conditions. Using the three state approach allows any feedback interval to work with peak performance. For our simulations and experiments we use a feedback interval value of 100 ms.

When the maximum video data rate is reached (a value set by the video streaming application), such that $A_{total} \geq V_{max}$, where A_{total} is actual throughput of each forwarder summed together and V_{max} is the max video rate, the distribution algorithm stops and enters a phase where it tries to equalize the data rates of each of the forwarding nodes (see lines 5–10 of Figure 3). This is done by lowering the expected throughput of the highest forwarder ($E_{high} = A_{high} - \Delta$) and raising the expected throughput of the lowest forwarder ($E_{low} = A_{low} + \Delta$). If the forwarder with the lowest expected throughput is able to support the extra throughput ($A_{total} \geq V_{max}$), then the process will be done again. If the forwarder is unable to handle the extra throughput ($A_{total} < V_{max}$), then the total throughput of each forwarder will not be greater than the maximum video data rate and the distribution algorithm will start again. By equalizing the throughput of each selected forwarder, the whole system is more stable. Rather than having a few forwarders at high throughput (possibly close to their limits), it is better to equally share the throughput across many forwarders so all are below their respective limits. If any fluctuations occur on a link, the other forwarders will be able to absorb the extra throughput. By proactively equalizing the rates, we are protecting against later adjustments (and loss) due to under-performing forwarders. mobiLivUp uses coarse granular feedback and hence is robust to temporary disruptions in cellular connectivity. For the same reason, it uses less cellular bandwidth for controlling the system.

B. Incentive Model

In mobiLivUp, we propose a pricing based method that provides incentives for the forwarding nodes to cooperate. We assume that all nodes are rational and selfish. A forwarding node's main goal is to maximize its profits but not to harm others. We use a simple auction to model cooperative bandwidth sharing. In our auction, the splitter holds the auction among n nearby forwarding nodes called *players*. Each player i has an individual private value c_i which is the cost of sending one unit of data to the gatherer using the cellular connection. c_i depends on various parameters including available cellular bandwidth, cellular data rate, and battery level on the phone. Our auction works as follows.

The splitter sends a request to the forwarding nodes and the forwarding nodes reply by sending their bids, c_i, to the splitter. The splitter determines the allocation rule and the payment mechanism based on the received bids and the received feedback from the gatherer about the actual data rates of the forwarders. Due to the changes in the actual data rates, the splitter holds auction every time it receives feedback from the gatherer. The forwarding nodes can also change their bids in each auction and it is possible that the number of forwarding nodes varies in different auctions because of the mobility of mobile phones. Our auction provides incentive for forwarding nodes to cooperate by implementing a *dominant equilibrium*. In this setting, each forwarding node's best strategy is to report its actual cost c_i, regardless of other player's strategies. Each player's utility is defined as its total received payment minus its cost of participation.

Problem Formulation And Solution: In our auction, the splitter must consider both the cost and the actual data rate in selecting forwarding nodes. Let $w_i \in [0, 1]$ be the weight that is assigned to the player i based on the gatherer feedback. We define a score s_i for forwarding node i, as $s_i(c_i, w_i) = \frac{w_i}{c_i}$. Note that in this equation, s_i depends on both the cost c_i and the actual data rate w_i. The splitter uses these scores to select a set of forwarding nodes. We obtain the utility of player i with score s_i from the following formula: $u_i(s_i) = p_i(a_{s_i}) - c_i$.

The number of players that are selected by the splitters depends on a budget limit B, the maximum amount that the splitter can pay per second. Depending on the type of the service, the splitter may choose different values for B[3].

[3]The budget limit, B can be increasing function of the total data rate, $\sum_{i=1}^{k} w_i$, where k is the number of selected players.

The splitter determines the allocation and payment based on the scores of forwarding nodes under the following conditions:
Optimal. The mechanism should maximize the total score, i.e., total actual data rates divided by total costs.
Incentive Compatibility. There is no selfish forwarding node that has an incentive to lie about the cost, c_i.
Individual Rationality. The utility of all forwarding nodes should be non-negative to provide incentive for them to participate in the game.

Our problem description is as follows:

$$\max_{a} \sum_{i=1}^{n} a_{s_i} s_i$$

$$s.t.$$

$$\forall i, c_i' \in C, p_i(a_{s_i}) - c_i \geq p_i(a_{s_i'}) - c_i \quad (1)$$

$$\forall i, p_i(a_{s_i}) - c_i \geq 0 \quad (2)$$

$$\sum_{i=1}^{n} p_i(a_{s_i}) \leq B \quad (3)$$

Here, $a_{s_i} \in \{0, 1\}$ represents the allocation to player i with score s_i, when the splitter assigns data to the player, $a_{s_i} = 1$, otherwise, $a_{s_i} = 0$. Also, $p_i(a_{s_i})$ represents the amount that the splitter pays to the player i under allocation rule a_{s_i}. Equation 1 provides incentive compatibility for cost[4]. Equation 2 is for individual rationality, and Equation 3 captures the budget limit of the splitter. The splitter orders the forwarding nodes based on their scores decreasing scores (s_i). Then, it selects the largest number of forwarding nodes $\{1, 2, \ldots, k\}$ such that $\forall i \in \{1, 2, \ldots, k\}$, $c_i < \frac{Bw_i}{\sum_{i=1}^{k} w_i}$.

The payment, $p_i(a_{s_i})$, is obtained from the following formula:

$$p_i(a_{s_i}) = min(\frac{w_i c_{k+1}}{w_{k+1}}, \frac{Bw_i}{\sum_{i=1}^{k} w_i}) \quad (4)$$

Here, $k + 1$ is the index of the player with the largest score after the selected k players. In equation 4, the payment increases linearly with the forwarding node's actual data rate, w_i. The proofs of incentive compatibility and individual rationality have been left out due to space constraints.

VI. Implementation

The splitter and forwarder components of our system are written as an Android application in Java on Samsung Galaxy S4 phones. For the data source component, we used Spydroid [18], an open source project that can stream a smartphone's camera video to a client either through HTTP or RTSP. The gatherer component is implemented on a server in Python. When the splitter is initially started, it creates a TCP connection with the gatherer. This connection acts as a way for control messages to be sent back and forth between the two components. On this connection, feedback is sent from the gatherer to the splitter. For a client to connect to a data source, it connects through the gatherer. The gatherer forwards all messages from a client to the splitter. The splitter then forwards messages to the data source.

When the RTSP connection has been established and data starts being sent to the splitter from the data source, the splitter distributes packets based on the available data rate of each of the forwarders. To keep track of the ordering of the packets, a sequence number is added to each packet. RTP contains a sequence number, but we did not want to be dependent on any particular application protocol so we do not use it. The splitter also adds the MAC address of the forwarding node it is using to send the packet. This allows the gatherer to keep statistics of how each forwarding node is performing. The gatherer calculates the throughput of each forwarding node and sends this back as feedback to the splitter. Based on the feedback, the splitter changes the weightings of the forwarding nodes to match their actual data rate. The gatherer sends feedback to the splitter every 100 ms. The weight of the forwarding node and the send time are also added to the packet header for debugging and data collection purposes.

The gatherer buffers received packets to make sure they are in order. Once packets are in the right sequential order, it sends the ordered packets to the client. Since we are using RTP, which uses UDP, there is no way of telling if a packet has been lost or if it is still in transition. This presents a problem to the gatherer to know if it should wait for a missing packet to arrive or skip it. To solve this, individual queues are kept for each forwarding node. When a packet is received, it is put into its corresponding queue. If sequence numbers of all the heads of the different queues is greater than the sequence number of the packet in question, then we know the packet must have either been lost or severely reordered by the network and can be counted as a loss. This allows the gatherer to skip that packet and continue to put the rest of the packets in order.

VII. Evaluation

We evaluate mobiLivUp using ns-3 simulations and an implementation on smartphones. There are three purposes for evaluating mobiLivUp in simulation: (i) to test our design with a large number of forwarding nodes (in comparison to only two nodes in our implementation) under various conditions, (ii) to determine the best parameters for the distribution algorithm, and (iii) to develop and understand the interaction between the video distribution algorithm and the incentive mechanisms. We collect and use cellular data traces and Wi-Fi Direct traces to give us realistic wireless characteristics such as data rates and loss.

We also implement the splitter and forwarder as Android applications in Java on two Samsung Galaxy S4 phones. We implement our gatherer on a server in Python. We use our implementation in the following two ways. First, we create an emulation platform that emulates cellular forwarding nodes using Wi-Fi, i.e., the forwarding nodes transmit data using Wi-Fi (and not a cellular network). Here, as in our simulation, we determine the transmit data rate of the forwarding nodes using our traces. However, unlike our simulation, we use real live video transmission from our Android phones to the

[4]Since the data rate of each forwarding node is obtained from the gatherer feedback, we do not need to provide incentive compatibility for the data rate.

TABLE I
SUMMARY OF SIMULATION RESULTS

# of Forwarders	Individual Phone (Kbps)	Aggregate Goodput (Kbps)	Jitter (ms)
3	2010.23	3615.27	3.07225
5	2197.61	5508.57	1.99712
10	2573.50	8711.93	1.32504

gatherer. The purpose of this emulation is to validate our implementation in a more realistic setting (compared to the simulation), while still having a controlled environment.

Finally, we use our implementation with Wi-Fi Direct and cellular links in a real-world live video transmission during a commuter train ride to evaluate the benefits of our approaches in real time under real conditions. We present the results of our simulation, implementation, and commuter train ride below.

A. Simulation-Based Evaluation

We implement mobiLivUp in the ns3 simulator. We first determine the best increase rate and decrease factor for our distribution algorithm.

Increase Rate (Δ), Decrease Factor (σ) : Using a cellular trace, we test our algorithm with a variety of forwarding nodes and increase rates. Figures 5a and 5b show the average throughput and loss of the cellular trace compared to the rate increase amount (in Kbps). The different lines represent different amounts of forwarding nodes used in the simulation. We observe that larger the increase amount higher the throughput and higher the loss. Intuitively, the higher the increase value, the more aggressive the probing gets, leading to increased throughput but more loss. Interestingly, the throughput levels out at around 400 Kbps, however the loss rate does not. Based on these findings, we choose an increase rate of 200 Kbps in our evaluation. It has an increased throughput compared to lower values, but the loss rate is still low.

Similar to the increase rate, we use our simulation to determine the best decrease factor. Figures 5c and 5d show the average throughput and loss compared to changes in the decrease factor. Our results show that a decrease factor between 0.2 and 0.25 allows us the best combination of high throughput and low loss.

We now test our distribution algorithm under various network conditions. Due to space constraints, we only show our results for one of the cellular traces (the same one we use for determining our increase rate and decrease factor). Table I shows the overall results using different amounts of forwarding nodes. Figure 6a shows the specific throughput of each forwarding node for the five forwarder case.

Compared to the individual phone with the highest throughput, mobiLivUp performs 79.8% better when using three forwarders, 150.7% better when using five forwarders, and 238.5% better when using ten forwarders. Figure 6a gives a clearer idea of how much each forwarder contributes to the overall throughput. The splitter node contributes the most, followed by forwarder one. All the other nodes contribute about the same amount of throughput to the aggregate throughput.

TABLE II
SUMMARY OF EXPERIMENT RESULTS

Scenario	Method	Median Goodput (Kbps)	Median Jitter (ms)
Varied Data Rate	One Phone	2242	0.067
	mobiLivUp	2200	1.854
Indoor Cell Trace	One Phone	553	11.670
	mobiLivUp	1067	716.7
Train	One Phone	1838	0.448
	mobiLivUp	3077	1.452

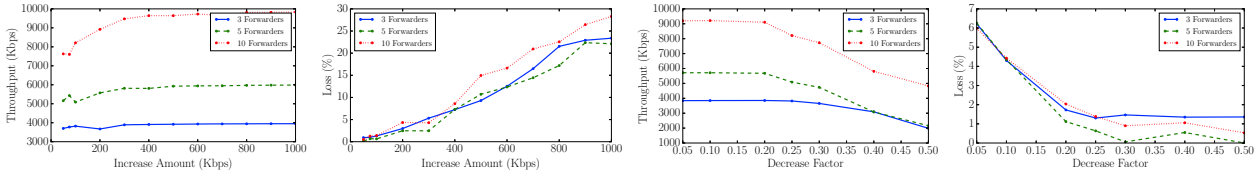
To understand how our distribution algorithm works with mobility, we use the SLAW mobility model [19] to determine when nodes connect and disconnect to the splitter over the course of a one hour simulation. Forwarder one stays connected to the splitter for the whole simulation. Forwarder two comes in at 15 minutes and leaves at 18 minutes. Forwarder three connects at the start of the simulation and disconnects after 51 minutes. Forwarder four comes in at 53 minutes and stays until the simulation ends. Finally, forwarder five comes in at 45 minutes and leaves at 52 minutes. Figure 6b shows how our algorithm adapts as forwarders connect and disconnect.

To demonstrate and test the function of our distribution algorithm when it has achieved the maximum throughput, we run a simulation using the following conditions. The splitter and all forwarding nodes that connect have 1400 Kbps of available throughput. The splitter starts out alone sending below the maximum video rate of 1500 Kbps. Forwarder one, two, three, and four connect at 15, 25, 35, and 45 seconds respectively. As each forwarder connects, the aggregate throughput is well above the maximum video data rate. As a result, the throughput of each node is equalized. This behavior can be seen in Figure 6c.

The results in Figure 7 show how the incentive model interacts in the simulator. In this simulation, only three forwarding nodes are used. Figure 7a shows the data rate limit of each of the forwarding nodes (thick line) in relation to the data rate the distribution algorithm has selected (thin line). Figure 7b shows the score of each forwarding node (see Section V-B). As the scores of the forwarding nodes change (based on how much data rate they can provide and their cost), the splitter selects different forwarders that can maximize the aggregate data rate and stay under its budget. In Figure 7b, this occurs at about 14 seconds and 21 seconds. This shows that the splitter is able to dynamically select the best set of forwarders that can provide the best value to it.

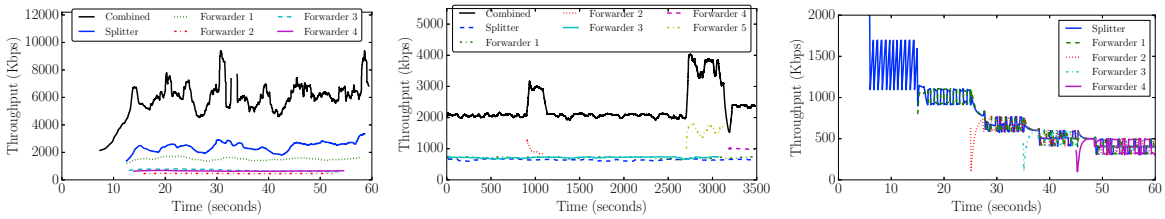
B. Wireless Testbed Evaluation

In this section, we look at the performance of mobiLivUp under three different scenarios. For the testbed, instead of using cellular connections to send data to the gatherer, Wi-



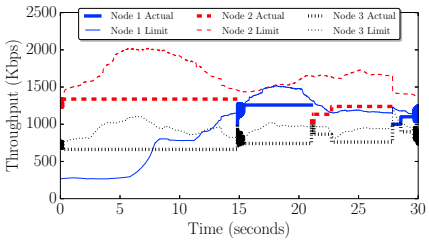
(a) (b) (c) (d)

Fig. 5. (a) and (b) show the throughput and loss as the increase rate changes. (c) and (d) show the throughput and loss as the decrease factor changes.

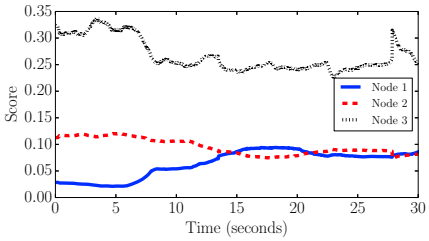


(a) (b) (c)

Fig. 6. Simulation results. (a) shows the throughput of individual forwarding nodes compared to the goodput of the system. (b) shows the distribution algorithm adjusting to forwarders connecting and disconnecting. (c) shows the throughput of all forwarders are equalized when new forwarders are added.



(a)



(b)

Fig. 7. (a) shows the data rate limit, based on the cellular trace, and the actual data rate, selected by the distribution algorithm. (b) shows the scores of each forwarding node based on the incentive model.

Fi is used.[5] Wi-Fi Direct is still used to connect the cellular devices together. The testbed allows for the data rate of individual forwarding nodes to be set for a specific amount of time. Using the testbed, we can emulate different data rate conditions, allowing us to see how the system reacts. The first scenario changes the data rate of each of the wireless nodes (Section VII-B1), testing how our system reacts to large changes in cellular conditions. In the second scenario, we playback a cellular trace (Section VII-B2) to test how the system responds to real conditions.

To measure the effectiveness of mobiLivUp, we use two metrics: goodput and jitter. Goodput is the rate at which the gatherer can send usable data to the client (in order video data). Goodput captures the effect of out of order packets because out of order packets need to be buffered before they can be sent to the client. Having only two devices available to us, we do not evaluate the incentive model using the wireless testbed. Table II summarizes the data for each scenario, showing the median for goodput and jitter. We present a more detailed discussion of the results below.

1) Varied Data Rate: In this scenario, the data rate of the splitter and forwarder start at 360 Kbps and 6 Mbps, respectively. After 60 seconds, the splitter's data rate increases to 6 Mbps. After 120 seconds, the forwarder's data rate decreases to 360 Kbps. Based on a study of throughput characteristics of cellular networks [6], we select 360 Kbps and 6 Mbps because they roughly represent the lower bound and average cellular bandwidth. The big difference in data rate between the splitter and gatherer help to illustrate any potential problems with out of order packets. The results are shown in Figure 8a.

The purpose of this scenario is to see how quickly the system can respond to changes in data rate. In this case, the changes occur at the 60 and 120 second marks. This scenario also tests how much the goodput is affected by these network changes. The median goodput is 2200 Kbps. The aggregate goodput is not much higher than the splitter or forwarder's

[5]Our data rates are low enough and do not saturate the Wi-Fi channel. Any data rate loss due to wireless devices sharing the channel will not affect the results of the testbed

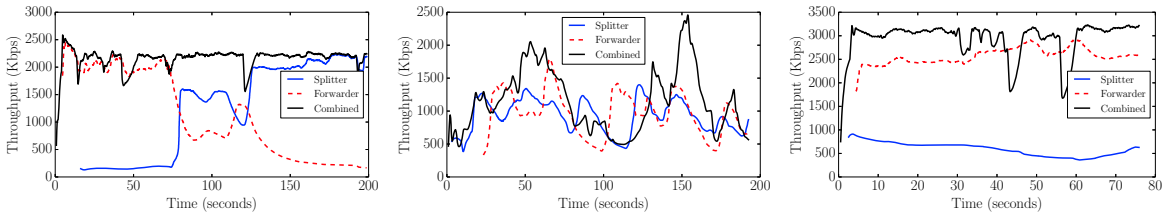


Fig. 8. (a) shows the results of varied data rate experiment. (b) shows results of the cellular trace experiment. (c) shows the results of the train experiment.

throughput. This is due to the low data rate of the splitter and forwarder, which is by design of the scenario. The results show that mobiLivUp is able to transition from one forwarder to another with minor dips in goodput.

2) Indoor Cellular Trace: The data rates used in this scenario are calculated from a cellular trace. We collected data to measure the maximum throughput of two cellular links at the same time. The maximum throughput was then entered into the wireless testbed to emulate the cellular environment.

In this particular trace, the data rates change dramatically and at a rapid pace. The trace was collected deep inside a building where cellular connectivity fluctuates. The rapid changes make it harder for the distribution algorithm to determine the data rate of each forwarding node and as a result, at some points, the aggregate goodput is less than an individual forwarding node's throughput.

The throughput of the splitter and forwarder and the goodput are shown in Figure 8b. mobiLivUp is able to maintain a higher goodput compared to a single phone 67.98% of the time with an average goodput of 1067 Kbps, compared to 553 Kbps when using just one phone. The jitter of mobiLivUp is higher than with one phone as expected due to it using multiple paths. This high jitter can be overcome by waiting to play the video for some amount of time. In this specific scenario, waiting about one second to play the video is acceptable.

C. Train Ride

We tested our system on a train that travels to different cities in the area. The train travels up to 79 mph and makes a stop roughly every eight miles. We traveled a total of 90 miles, collecting data along the way, using two cell phones, one acting as a splitter/forwarder and another acting as a forwarder. Figure 8c shows the throughput of each of the nodes as well as the goodput. The goodput exceeds the throughput of both the forwarding nodes, except for two instances. The displayed results only show a small portion of the data that we collected on the train, while Table II takes into account all of the data collected on the train. The results of this experiment show that mobiLivUp is capable of producing almost ideal aggregate throughput in real network conditions. mobiLivUp performs better than one phone 88.6% of the time with a 67% improvement in goodput.

In summary, in all of our experiments, mobiLivUp has a higher goodput when compared to the performance of a single phone, with the exception of the varied data rate experiment, in which the single phone and mobiLivUp perform comparably on average. Using mobiLivUp, we can have a much higher quality of video but need to introduce a playout delay of only a few seconds to deal with the higher jitter.

VIII. Conclusions

mobiLivUp is a novel system that provides greater aggregate upload of real-time video data through using cellular connections of nearby neighbors. We designed a novel distribution algorithm to distribute live video to nearby devices at their maximum capacity. Using an incentive model, we can select nearby devices to participate, in which both parties benefit.

References

[1] "Periscope," https://www.periscope.tv.
[2] "Meerkat," https://meerkatapp.co.
[3] Comcast, "X1: Using XFINITY Share," http://goo.gl/kNREgq.
[4] "GoPro," http://gopro.com.
[5] Cisco, "Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 2013-2018," Tech. Rep., February 2014. [Online]. Available: http://goo.gl/ScBQaI
[6] Y.-C. Chen and et al., "Characterizing 4g and 3g networks: Supporting mobility with multipath tcp," *UMass Amherst Technical Report: UM-CS-2012-022*.
[7] ——, "A measurement-based study of multipath tcp performance over wireless networks," ser. IMC '13.
[8] M. Smith, "Sony shows (and tells) us why 4K on a phone isn't crazy," http://goo.gl/lxYJZA.
[9] Wi-Fi Alliance Technical Committe P2P Task Group, *Wi-Fi Peer-to-Peer (P2P) Technical Specification, Version. 1.2*, 2010.
[10] W. Hu and G. Cao, "Quality-aware traffic offloading in wireless networks," ser. MobiHoc '14.
[11] L. Keller and et al., "Microcast: Cooperative video streaming on smartphones," ser. MobiSys '12.
[12] A. Ford and et al., "TCP Extensions for Multipath Operation with Multiple Addresses," Internet Requests for Comments, RFC 6824, January 2013. [Online]. Available: https://tools.ietf.org/html/rfc6824
[13] Y. Lim and et al., "Improving energy efficiency of MPTCP for mobile devices," *CoRR*.
[14] R. Stewart, "Stream control transmission protocol," Internet Requests for Comments, RFC 4960, September 2007. [Online]. Available: http://tools.ietf.org/html/rfc4960
[15] Z. Shen and et al., "Overview of 3gpp lte-advanced carrier aggregation for 4g wireless communications," *IEEE Communications*, 2012.
[16] S. Jakubczak and et al., "Link-alike: Using wireless to share network resources in a neighborhood," *AMC MC2R*, 2009.
[17] Sascha Segan, "Fastest Mobile Networks 2015," http://www.pcmag.com/article2/0,2817,2485837,00.asp.
[18] S. Guigui, "Spydroid-ipcamera," https://goo.gl/2iQiqX.
[19] K. Lee, S. Hong, S. J. Kim, I. Rhee, and S. Chong, "Slaw: A new mobility model for human walks," in *INFOCOM 2009, IEEE*, April 2009, pp. 855–863.