OPERETTA: An Optimal Energy Efficient Bandwidth Aggregation System

Karim Habak
Wireless Research Center
Egypt-Japan University of
Sc. and Tech. (E-JUST)
Email: karim.habak@ejust.edu.eg

Khaled A. Harras
Computer Science Department
School of Computer Science
Carnegie Mellon University Qatar
Email: kharras@cs.cmu.edu

Moustafa Youssef
Wireless Research Center
Egypt-Japan Univ. of Sc. and Tech. (E-JUST)
and Alexandria University, Alexandria, Egypt.
Email: moustafa.youssef@ejust.edu.eg

***Abstract*—The widespread deployment of varying networking technologies, coupled with the exponential increase in end-user data demand, have led to the proliferation of multi-homed or multi-interface enabled devices. To date, these interfaces are mainly utilized one at a time based on network availability, cost, and user-choice. While researchers have focused on simultaneously leveraging these interfaces by aggregating their bandwidths, these solutions however, have faced a steep deployment barrier. In this paper, we propose a novel optimal, energy-efficient, and deployable bandwidth aggregation system (OPERETTA) for multiple interface enabled devices. OPERETTA satisfies three goals: achieving a user defined throughput level with optimal energy consumption over multiple interfaces, deployability without changes to current legacy servers, and leveraging incremental deployment to achieve increased performance gains. We present the OPERETTA architecture and formulate the optimal scheduling problem as a mixed integer programming problem yielding an efficient solution. We evaluate OPERETTA via implementation on the the Windows OS, and further verify our results with simulations on NS2. Our evaluation shows the tradeoffs between the energy and throughput goals. Furthermore, with no modifications to current legacy servers, OPERETTA achieves throughput gains up to 150% compared to current operating systems with the same energy consumption. In addition, with as few as 25% of the servers becoming OPERETTA enabled, OPERETTA performance reaches the throughput upper bound, highlighting its incremental deployment and performance gains.**

I. Introduction

With the continuous advances in technology, decreasing cost of electronics, and increased user demand for mobile data, it is the norm nowadays to find devices with various network interfaces. These devices such as laptops, netbooks, tablets, and various smart phones, are equipped with a variety of networking interfaces to communicate with any technology available, such as Bluetooth, Wifi, 3G and WiMax. Such devices, however, are currently not able to utilize these existing interfaces in order to enhance the overall system performance. These multiple interfaces can be used simultaneously by partitioning and distributing data across them in order to achieve higher throughput, enhance the user experience or minimize the energy consumption.

There have been many approaches that address the multiple interface bandwidth aggregation problem over the years at different layers of the protocol stack [1]–[14] (Section II). These solutions, however, mainly focus on leveraging these interfaces to increase system throughput, while overlooking all other aspects that characterize an optimal, energy-efficient, and deployable bandwidth aggregation system.

An effective bandwidth aggregation system should be: (1) Easily deployable without requiring changes to legacy servers, applications, or the addition of new hardware (such as proxy servers). (2) Energy-efficient to conserve the limited battery resources of mobile devices while being flexible in meeting the user's needs of optimal throughput, if required. (3) Leveraging incremental system adoption and deployment to further enhance performance gains.

We therefore present OPERETTA as a novel Optimal Energy-Efficient Deployable Bandwidth Aggregation System. Our system is based on a middleware that lies right below the application layer, requiring no changes to either the OS kernel nor the applications. We also present OPERETTA's system architecture that allows it to work with current legacy servers and leverage any OPERETTA enabled servers to further enhance performance. One of the core functionalities of the OPERETTA middleware is to schedule different connections to different interfaces. We formulate the OPERETTA scheduling problem that allows the user to achieve a desired throughput with minimal energy consumed. We show that this is a mixed integer programming problem that has a special structure allowing it to be efficiently solved.

We evaluate OPERETTA via implementation on the Windows OS, as well as via simulation, and compare the results to the optimal achievable throughput and energy consumption. Our results show that, with no changes to the current legacy servers, OPERETTA can achieve 150% enhancement in throughput as compared to the current operating systems, with no increase in energy consumption. In addition, with only 25% of the nodes becoming OPERETTA enabled, the system performance reaches the throughput upper-bound. This highlights that OPERETTA achieves the goals of being optimal, energy-efficient, as well as easily and incrementally deployable.

The remainder of this paper is organized as follows. Section II discusses the related work and how OPERETTA compares to them. We then present the overall architecture of our system in Section III. Section IV formulates the OPERETTA scheduling problem. In Section V, we discuss our

Windows OS implementation and its deployability. We evaluate OPERETTA and compare it to the optimal and baseline schedulers in Section VI. Finally, Section VII concludes the paper and provides directions for future work.

II. Related Work

There have been many approaches that address the multiple interface bandwidth aggregation problem over the years. These techniques are implemented at different layers of the protocol stack. Application layer solutions typically require applications to be aware of the existence of multiple interfaces and be responsible for utilizing them [10]. Socket level solutions, on the other hand, modify the kernel socket handling functions to enable existing applications to use multiple interfaces [10], [11]. Although [10] does not modify existing applications, it requires changes to the legacy servers in order to support these new sockets. In addition, [11] requires feedback form the applications about their performance, and hence is not backwards compatible with previous versions of the applications.

Many bandwidth aggregation techniques, however, naturally lie in the transport layer [1]–[9]. These solutions replace TCP with mechanisms and protocols that handle multiple interfaces. Such techniques require changes to the legacy servers and hence have a huge deployment barrier. Finally, network layer approaches update the network layer to hide the variation in interfaces from the running TCP protocol [12]–[14]. Chebrolu et al. [12] requires having a proxy server that communicates with the client and is aware of the client's multiple interfaces. Others [13] selected to implement their system at both connection ends which makes their deployment rely on updating the legacy servers. On the other hand, MAR [14] requires having a special router as well as an optional proxy server. The fact that modern operating systems, such as Windows, MAC OS, and Linux, allow users to use only one of the available interfaces, even if multiple of them are connected to the Internet, attest that all the current proposals for bandwidth aggregation face a steep deployment barrier.

Our previous work in deployable bandwidth aggregation systems (DBAS) focuses on the actual implementation of the basic core system [15]. This work was then extended with G-DBAS [16] which added energy awareness on top of DBAS. These systems, however, either operate only in the connection-oriented mode, or do not provide optimal scheduling algorithms that exploit the full potential of the interfaces available. OPERETTA builds on our previous work by achieving a user defined throughput with optimal minimum energy-consumption while maintaining the core features of being deployable, as well as providing incremental performance gain as it becomes more adopted over the Internet.

III. System Architecture

In this section, we provide an overview of the OPERETTA architecture, depicted in Figure 1, followed by a brief description of its components. This architecture builds upon our core DBAS system [15], with most changes proposed in this paper lying in the new optimal OPERETTA scheduler.

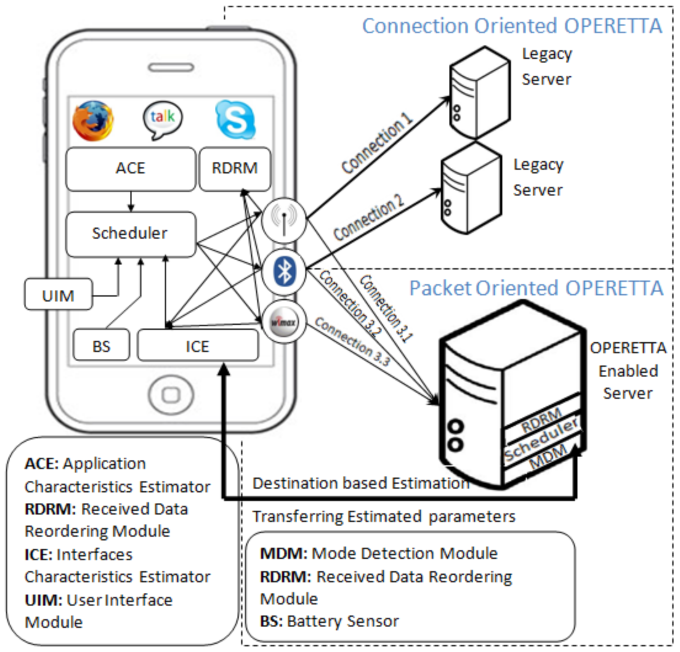

Connection Oriented OPERETTA
Legacy Server
Legacy Server
Connection 1
Connection 2
Packet Oriented OPERETTA
Connection 3.1
Connection 3.2
Connection 3.3
OPERETTA Enabled Server
RDRM
Scheduler
MDM
ACE
RDRM
Scheduler
UIM
BS
ICE
Destination based Estimation
Transferring Estimated parameters
ACE: Application Characteristics Estimator
RDRM: Received Data Reordering Module
ICE: Interfaces Characteristics Estimator
UIM: User Interface Module
MDM: Mode Detection Module
RDRM: Received Data Reordering Module
BS: Battery Sensor

Fig. 1. OPERETTA system architecture and scheduling parameters.

We consider a client host equipped with multiple network interfaces connected to the Internet. Each interface has its own parameters in terms of bandwidth, latency, loss ratio, and energy consumption. The device runs multiple applications with varying communication characteristics. In its default mode, OPERETTA schedules different connections to the interfaces such that a connection can be assigned to only one of the available interfaces. Once assigned to an interface, all the packets of this connection utilize the same interface. OPERETTA, therefore, achieves bandwidth aggregation without requiring any changes to the legacy servers (the server deals with a normal connection). If the other end of the connection is also OPERETTA enabled, the system leverages this fact to further enhance performance by switching to a packet-based mode, where each packet can be scheduled independently on a different interface (incremental deployment property). OPERETTA is composed of the following components as shown in Figure 1.

A. Application Characteristics Estimator

To be fully deployable, OPERETTA does not require any changes to existing applications. Knowing the application characteristics, however, enables us to fully utilize the available interfaces and make better scheduling decisions. Our approach is to automatically estimate the characteristics of the applications based on their behavior. These characteristics are stored in a database for keeping track of the applications behavior. This functionality is the responsibility of the application characteristics estimation module that utilizes both qualitative and quantities measures.

In qualitative measures, some features can be used to characterize the behavior of an application. For example, the

TABLE I
POWER CONSUMPTION TABLE [17]

Interface	Technology	Low-Power Idle	Active Tx
Netgear MA701	Wifi	264 mW	990 mW
Linksys WCF12	Wifi	256 mW	890 mW
BlueCore3	Bluetooth	25 mW	120 mW
GSM 850/900/1800/1900	GPRS	0.4 W	1.27 W

process name can be used to determine whether the process is realtime, e.g. Skype, or bandwidth intensive, e.g. an FTP client. Similarly, specific ports reserved by the application can also be used to characterize the applications behavior.

In quantitative measures, on the other hand, the module also estimates the average connection data demand in bytes of any given application. After a connection is terminated, the module updates the estimated values of connection data demand (C_{demand}) as:

$$C_{demand} = (1 - \alpha)C_{demand} + \alpha CC_{demand} \tag{1}$$

where CC_{demand} is the number of bytes transmitted by the terminated connection and α is a smoothing coefficient, taken equal to 0.125 to evaluate the equation efficiently. We note that the granularity of estimation can be rendered more fine grained at the expense of increased complexity and lower scalability.

B. Mode Detection Module

The purpose of this module is to detect whether the end point of the connection supports OPERETTA or not to enable the optional packet-oriented mode. This module is implemented as a service that runs on a specific port reserved for OPERETTA. When a client tries to establish a new connection, it first attempts to open a dummy connection to this port. If successful, this indicates that OPERETTA is supported at the end point and hence the packet-oriented mode can be used. If so, OPERETTA will open multiple connections to the destination seamlessly from the application over multiple interfaces and will distribute the data across those interfaces. Otherwise, OPERETTA reverts to the connection-oriented mode. To avoid the delay of determining the operation mode, the connection-oriented mode can be used simultaneously while probing the end point.

C. Interface Characteristics Estimator

This module is responsible for estimating the characteristics of each network interface. In particular, it estimates the available bandwidth at each interface. It periodically connects and communicates with various geographically dispersed servers to estimate the uplink and downlink available bandwidth for each interface. These estimates are then combined with the statistics collected during the normal data transfer operation. These estimates are sufficient as the bandwidth bottlenecks typically exist at the client's end not the server's, which is typically connected to a high bandwidth link and designed to scale with the number of clients.

To characterize energy consumption, Table I shows how the power consumption of a network interface depends on the NIC and technology used [17]. Hence, in order to make OPERETTA estimate the energy consumption for its attached network interfaces, we built an online service with a database containing the energy consumption rates for various network interfaces. Once OPERETTA runs, it queries the database for the energy consumption rates of each network interface. This can also be estimated dynamically during the device operation.

D. Battery Sensor

This module senses the available battery level and whether the device is plugged to a power source or running on battery.

E. User Interface Module

This module is responsible for obtaining the user's preferences and interface usage policies (utility). The user can configure this module to enforce some interface selection criteria. For example, the user may wish to assign specific applications (e.g. realtime applications) to specific interfaces (e.g. wired). Similarly, the user might prefer to reduce cost and give higher priority/weight to interfaces that have a free connection or low energy.

F. Scheduler

This is the core of the OPERETTA system. It uses the application, interface characteristics estimators, and the device status, to schedule the connections to different interfaces. We provide more details about the OPERETTA scheduler in Section IV

G. Received Data Reordering Module

This module is utilized in the packet-oriented mode. It is responsible for reordering the incoming data chunks from different interfaces in order to pass them on to the application layer at the receiving end. This is enabled by the sender adding an OPERETTA header, which only contains a chunkId that is used in reordering the chunks when they reach their destination. When an interface is down, unacknowledged chunks can then be re-sent over other interfaces, showing the effect of connection migration.

IV. OPERETTA SCHEDULER

In this section, we formulate the optimal OPERETTA scheduling problem. We start by describing our system model followed by the optimal scheduling problem. We then discuss the problem solution and special cases.

A. System Model

Table II summarizes the system parameters and notation. We assume a mobile device with m interfaces each with rate r_j and energy consumption rate of a_j, where a_j equals the difference in power consumption between the active and idle states of interface j. The device is running a set of applications that share these interfaces. Our scheduling unit is a connection, regardless of its application source[1]. We refer to a standard network connection as a ***stream*** to avoid confusion with the

[1]The application, however, may be used to characterize the connection in terms of QoS by the Application Characteristics Estimation Module (Section III-A).

TABLE II
LIST OF SYMBOLS USED

Symbol	Description
$\mathcal{T}$	The overall system throughput
$\mathcal{L}$	The current system load
$\mathcal{L}_i$	The current system load for stream i
$\mathcal{S}_i$	Determines whether stream i is connection based (1) or packet-based (0)
r_j	The effective bandwidth of interface j
a_j	Difference in power between active and idle states of interface j
$\mathcal{E}$	The energy consumed in order to transfer the system load
$\mathcal{E}_j$	The energy consumed for interface j to transfer its load
Δ_j	The time needed for interface j to finish its load
x_{ij}	For connection-oriented streams, equals 1 if stream i is assigned to interface j. Equals 0 otherwise.
w_j	The ratio of packets assigned to interface j
n	Number of active streams including the new request
m	Number of interfaces

scheduling unit. OPERETTA's goal is to assign streams to interfaces in order to minimize the required energy ($\mathcal{E}$) to achieve a desired throughput ($\mathcal{T}$). Scheduling decisions are taken when a new stream (number of streams active in the system is n, including the new stream) is requested from an application. The Mode Detection Module then determines (Section III-B) whether the operation mode is connection-based ($S_n = 1$), or packet-based ($S_n = 0$) if the other end is OPERETTA-enabled. If it is connection-based, the scheduler goal is to determine to which interface it should be assigned (sets $x_{nj} = 1$ for only one interface j). In either case, the percentage of packets to be assigned to each interface, i.e. interfaces relative packet load (w_j), should be re-calculated based on the current system load ($\mathcal{L}$).

Utility Function: The target throughput for the scheduler is set based on a user utility function that balances the system throughput and the energy consumption. At one extreme, the scheduler's goal can be to minimize energy ($\mathcal{E}$) required to consume the system load. This can be achieved by assigning all traffic (streams) to the interface ($i_{p_{\min}}$) with the minimum energy consumption per bit, achieving a throughput of $r_{i_{p_{\min}}}$. At the other extreme, the scheduler's goal may be to maximize system throughput ($\mathcal{T}$), in which case, all interfaces should be carefully utilized to maximize throughput, regardless of the consumed energy. In this case, the maximum achievable throughput is $\sum_i r_i$. In between, a parameter $\alpha, 0 \le \alpha \le 1$, is used to tune the system performance between these two extremes achieving a throughput of $r_{i_{p_{\min}}} + \alpha \sum_{i \neq i_{p_{\min}}} r_i$. This α is chosen based on a user utility function that reflects user preferences.

B. Optimal Scheduling

In this section, we describe our objective function and system constraints. The decision variables are: (1) If $S_j = 1$, which interface to assign the new stream n to (variable x_{nj}) and (2) the new values for w_j, $\forall_j : 1 \le j \le m$.

1) Objective Function: The overall objective of the scheduler is to minimize the overall system energy consumption ($\mathcal{E}$) to consume the system load:

$$\text{Minimized } \mathcal{E} = \sum_j \mathcal{E}_j$$

where, $\mathcal{E}_j$ is the energy consumption of interface j. For interface j, this energy can be divided into two parts: (1) energy needed to finish the load of the connection-oriented streams and (2) energy needed to finish the load of the packet-oriented streams. The former is equal to $a_j t_j$, where t_j is the time required for interface j to finish its connection-oriented load, $t_j = \sum_{i=1}^{n} \mathcal{L}_i \mathcal{S}_i x_{ij} / r_j$. Similarly, the later is equal to $\sum_{i=1}^{n} a_j \mathcal{L}_i (1 - \mathcal{S}_i) w_j / r_j$. Since connection-oriented streams cannot be re-assigned to other interfaces, the objective function becomes:

$$\text{Minimize } \mathcal{E} = \sum_{j=1}^{m} \frac{a_j}{r_j} \left(w_j \sum_{i=1}^{n} (\mathcal{L}_i (1 - \mathcal{S}_i)) + \mathcal{L}_n \mathcal{S}_n x_{nj} \right) \tag{2}$$

2) Constraints: The following constraints must be satisfied:

Target Throughput: As defined in Section IV-A, the user utility function defines a minimum throughput that should be achieved ($\mathcal{T}_{\text{Target}} = r_{i_{p_{\min}}} + \alpha \sum_{i \neq i_{p_{\min}}} r_i$). Therefore,

$$\mathcal{T} = \frac{\mathcal{L}}{\max_j \Delta_j} \ge \mathcal{T}_{\text{Target}} \tag{3}$$

where Δ_j is the time needed by interface j to finish all its load (connection and packet based).

$$\rightarrow \forall_j, \Delta_j = \frac{\sum_{i=1}^{n} (\mathcal{L}_i (1 - \mathcal{S}_i) w_j) + \sum_{i=1}^{n} (\mathcal{L}_i \mathcal{S}_i x_{ij})}{r_j} \le \frac{\mathcal{L}}{\mathcal{T}_{\text{Target}}}$$

Rearranging:

$$\rightarrow \forall_j, w_j \sum_{i=1}^{n} (\mathcal{L}_i (1 - \mathcal{S}_i)) + \mathcal{L}_n \mathcal{S}_n x_{nj} \le \frac{\mathcal{L} r_j}{\mathcal{T}_{\text{Target}}} - \sum_{i=1}^{n-1} (\mathcal{L}_i \mathcal{S}_i x_{ij}) \tag{4}$$

Note that the RHS is constant.

Integral Association: If the new stream is connection-oriented, it should be assigned to only one interface:

$$\sum_{j=1}^{m} x_{nj} + (1 - S_n) = 1 \tag{5}$$

Note that when $S_n = 0$, $x_{nj} = 0, \forall j$, which is the case when the new stream is determined to be packet-based by the Mode Detection Module.

Packet Load Distribution: For packet-oriented streams, their total load should be distributed over all interfaces:

$$\sum_{j=1}^{m} w_j = 1 \tag{6}$$

Variable Ranges: The trivial constraints for the range of the decision variables

$$w_j \ge 0, 1 \le j \le m \tag{7}$$

$$x_{nj} \in \{0, 1\}, 1 \le j \le m \tag{8}$$

C. Scheduling Algorithm

In summary, OPERETTA aims to minimize the energy consumed based on Equation 2, while satisfying the set of constraints mentioned above.

Algorithm 1 OPERETTA scheduling algorithm

```
Input: Type of new stream ($S_n$), type of current streams ($S_i$), their assignment to interfaces ($x_{ij}$), remaining load for each stream ($\mathcal{L}_i$), interfaces characteristics ($r_j, a_j$), and desired user utility ($\alpha$).
Output: Assignment of the new stream to an interface ($x_{nj}$, $x_{nj} = 0$ if packet-based) and new weights for distributing packet-based streams over each interface ($w_j$).
Initialization: Order the interfaces in an ascending order according to their energy consumption per unit data ($\frac{a_i}{r_i}$).
if The server is not OPERETTA enabled then
    for $i = 1, ...., m$ do
        $x_{ni} = 1; \forall_{k \neq i} x_{nk} = 0$
        $\mathcal{T}_{max} = \frac{\mathcal{L}}{\max_j \left( \frac{1}{r_j} \sum_i^n \mathcal{L}_i \mathcal{S}_i x_{ij} \right)}$
        $\mathcal{T}_{Target} = \min \left( r_1 + \alpha \sum_{i=1}^m r_i, \mathcal{T}_{max} \right)$
        for $j = 1, ...., m$ do
            calculate $\omega_j$ using Equation 9
        end for
        calculate $\mathcal{E}$ using Equation 2
        calculate $\mathcal{T}$ using Equation 3
    end for
    $x_{ni} = 1$, if $x_{ni} = 1$ achieves $\mathcal{T} \geq r_1 + \alpha \sum_{i=1}^m r_i$
     with $\min \mathcal{E}$ or $\max \mathcal{T}$ if $\max \mathcal{T} < r_1 + \alpha \sum_{i=1}^m r_i$
else
    $\forall_j : x_{nj} = 0$
end if
$\mathcal{T}_{max} = \frac{\mathcal{L}}{\max_j \left( \frac{1}{r_j} \sum_i^n \mathcal{L}_i \mathcal{S}_i x_{ij} \right)}$
$\mathcal{T}_{Target} = \min \left( r_1 + \alpha \sum_{i=1}^m r_i, \mathcal{T}_{max} \right)$
for $j = 1, ...., m$ do
    calculate $\omega_j$ using Equation 9
end for
```

In general, this problem is a mixed 0-1 Integer Programming problem. However, it has a special structure that allows for an efficient solution. In particular, we have two cases: if the new stream that triggered the scheduling decisions is packet-based ($S_n = 0$) and if it is connection-based ($S_n = 1$). Algorithm 1 summarizes the OPERETTA scheduling algorithm.

1) Solution for the packet-based case: In this case, $x_{nj} = 0 \, \forall j$. The problem becomes a standard linear programming problem. Actually, it can be mapped to the continuous Knapsack problem which can be solved in linear time. In particular, in order to minimize $\mathcal{E}$, we have to utilize the interfaces with the smallest $\frac{a}{r}$ in order. Therefore, we sort all the interfaces in an ascending order according to their $\frac{a_i}{r_i}$. Then from the throughput constraint (Equation 4) we find that:

$$\forall_j w_j \leq \frac{\mathcal{L} r_j - \mathcal{T}_{\text{Target}} \sum_i \left(\mathcal{L}_i \mathcal{S}_i x_{ij} \right)}{\mathcal{T}_{\text{Target}} \sum_i \left(\mathcal{L}_i \left(1 - \mathcal{S}_i \right) \right)}$$

We then iterate on the interfaces setting w_j using the following formula:

$$w_j = \min \left(1 - \sum_{k<j} w_k, \frac{\mathcal{L} r_j - \mathcal{T}_{\text{Target}} \sum_i \left(\mathcal{L}_i \mathcal{S}_i x_{ij} \right)}{\mathcal{T}_{\text{Target}} \sum_i \left(\mathcal{L}_i \left(1 - \mathcal{S}_i \right) \right)} \right) \quad (9)$$

2) Solution for the connection-based case: In this case, we need to determine the binary variables $\forall_j x_{nj}$ such that only one of them equals 1 and others are 0. A quadratic time algorithm sets each one of them to 1 and then solves for $\forall_j w_j$ using the linear time algorithm in the previous section. The values that have the minimum $\mathcal{E}$ are selected.

There are some interesting special cases that can be obtained from our general framework presented in this section. First, if the goal is to maximize throughput, i.e. $\alpha = 1$, then $\tau_{\text{Target}} = \sum_{i=1}^m r_i$. In addition, if all streams are packet-based, i.e. all servers are OPERETTA enabled, the scheduler reduces to a weighted round robin scheduler based on Equation 4. For the case when the user is interested in energy consumption, i.e. $\alpha < 1$, the scheduler is reduced to a weighted round robin scheduler for the subset of interfaces that have the least a_i/r_i ratio, based on the required saving in energy consumption. This means that the interfaces with high energy per bit consumption will be inactive, until we reach the extreme case of utilizing only the interface with the minimum energy per bit when $\alpha = 0$. We compare the performance of OPERETTA and weighted round robin schedulers in Section VI.

V. Implementation

To verify the deployability of OPERETTA and evaluate our system, we extend our previous DBAS implementation [15] on the Microsoft Windows Operating System; currently, we have clients for Windows 7, Vista, and XP. We choose the Windows platform to demonstrate the deployability of OPERETTA even on closed source operating systems. In this section, we briefly discuss OPERETTA implementation, specifically highlighting its middleware and monitoring application. Both software modules are installed using a standard Windows executable.

We implement the OPERETTA middleware as a Layered Service Provider (LSP) [18], which is installed as part of the TCP/IP protocol stack in the Windows OS. This middleware uses the same concept adopted in implementing firewalls and network proxies in order to control traffic flows. The procedure starts when the application, aiming to connect to the Internet, uses the Winsock 2 API. Windows will dynamically link it to the Winsock 2 library which contains the implementation of all the Winsock 2 functions. This library sends its requests to the service provider which later forwards it to the base protocol, such as TCP/IP. Our OPERETTA LSP intercepts the Winsock 2 API requests and either schedules the connection to its selected interface or distributes data across the different network interfaces based on the mode of operation. In addition, it performs the functionality of the mode detection, application characteristic estimation, and interface characteristic estimation described in Section III.

The monitoring application we implement represents the user interface module that captures the user's preferences and interface usage policies and further monitors OPERETTA behavior. This module is utilized to enable the user to enter the desired utility function as well as for testing purposes, where it allows the user to select one of the scheduling techniques which OPERETTA offers. It also provides the ability to perform manual assignment of certain applications to certain interfaces. Finally, this module allows users to monitor OPERETTA internal data structures by interfacing with the OPERETTA middleware.

VI. Performance Evaluation

In this section we evaluate the performance of OPERETTA via implementation. All results have been validated using NS2 simulations. We start by describing our experimental setup followed by presenting and analyzing our results.

A. Experimental Setup

Table III lists the main parameters employed in our evaluation. Without loss of generality, our testbed consists of four nodes: an OPERETTA enabled server, a legacy server, a client, and an intermediate traffic shaper node. Both servers are connection destinations. The intermediate node is a device running the NIST-NET [19] network emulator to emulate the varying network characteristics of each interface. The client is the connection generator enabled with multiple network interfaces. On the client, we run different applications that vary in terms of the number of connections they open per second (β) and the average connection data demand (λ). The client is connected to the intermediate node through three interfaces: IF$_1$, IF$_2$ and IF$_3$ representing Wifi, Bluetooth and GSM network interfaces. Each server is connected to the intermediate node using a high bandwidth link, L_1 and L_2. We note that the combined bandwidth of IF$_1$, IF$_2$ and IF$_3$ is less than each server bandwidth in order to test the impact of varying the interface characteristics and scheduling strategies. We define $\gamma \in [0, 100]$ as the percentage of connections that have the OPERETTA enabled server as its destination. When $\gamma = 0$ all the connections have the legacy server as their destination. However when $\gamma = 100$ all the connections have the OPERETTA enabled server as their destination.

We evaluate OPERETTA using two classes of applications: small load, and large load. Small load applications represent typical web browsing applications with an average connection demand of $\lambda_{small} = 22.38KB$ [20]. Large load applications, on the other hand, represent P2P and FTP applications with an average connection demand of $\lambda_{large} = 0.285MB$ [20]. The connection establishment rate follows a poisson process with mean (β) connections per second. β_{small} and β_{large} are changed to achieve different application mixes. Each experiment represents the average of 15 runs.

Overall, we use throughput and energy consumption per unit data as our metrics while varying several parameters including

TABLE III
Experiments parameters

Parameter	Value range	Nominal Value
L_1 (Server #1) Bandwidth(Mbps)	6	6
L_2 (Server #2) Bandwidth(Mbps)	6	6
IF$_1$ Bandwidth(Mbps)	0.25 - 2	1
IF$_1$ power consumption (mWatt)	634	634
IF$_2$ Bandwidth(Mbps)	0.7	0.7
IF$_2$ power consumption (mWatt)	95	95
IF$_3$ Bandwidth(Mbps)	2	2
IF$_3$ power consumption (mWatt)	900	900
β_{small} (Connections/sec)	13	13
β_{large} (Connections/sec)	0 - 5	1

user utility (α), stream type mix (γ), interface characteristics, and estimation error. We compare OPERETTA to four baseline schedulers:

1) Throughput Optimal Scheduler: This imaginary scheduler represents the maximum achievable throughput.

2) Energy Optimal Scheduler: This imaginary scheduler represents the minimum energy consumption.

3) Round Robin (RR): Assigns streams or packets, based on the destination server type (legacy or OPERETTA enabled), to network interfaces in a rotating basis.

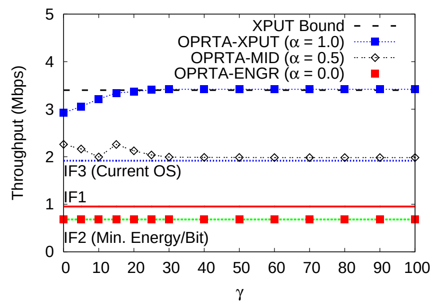
4) Weighted Round Robin (WRR): Similar to RR scheduler but weights each interface by its estimated bandwidth.

B. Results

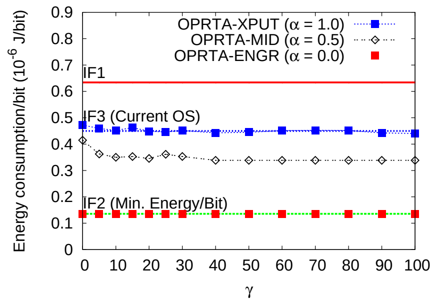
In this section, we study the effect of the stream-type mix on the performance of OPERETTA, its adoption to interface dynamics, its performance on OPERETTA for different utility functions, its robustness to estimation error, effect of connection heterogeneity, and compare OPERETTA's performance to the baseline scheduling algorithms.

1) Impact of Stream-Type Mix (γ): As discussed in Section III, OPERETTA performs best when all streams are packet-based as the scheduler can leverage the fine grained granularity to enhance the performance. Figure 3 shows the effect of changing γ on the performance of OPERETTA for three user utility values (α): 1.0 (max. throughput), 0.0 (min. energy), and 0.5 (mix).

The figure reveals a number of interesting observations: (1) When γ is low, most of the streams are connection-oriented. This makes the scheduling decision coarse grained (once the stream is assigned to an interface, all its packets have to go through this interface until it terminates), reducing the optimality of scheduling. (2) For the throughput maximization mode ($\alpha = 1$), even with no servers that are OPERETTA-enabled ($\gamma = 0$), OPERETTA can enhance the throughput by 150% as compared to the current OSs. OPERETTA reaches the throughput upper bound when we have only 25% of the streams packet-based. In other words, to obtain the best possible throughput, OPERETTA requires only 25% of the servers to be OPERETTA enabled. This gain comes with energy consumption that is equal to the same energy consumed by the current OSs (as shown in Figure 3(b). This result highlights the significant gains OPERETTA achieves, in terms of throughput and energy, with the ability to support incremental deployment. (3) For the user utility value $\alpha = 0.5$, since

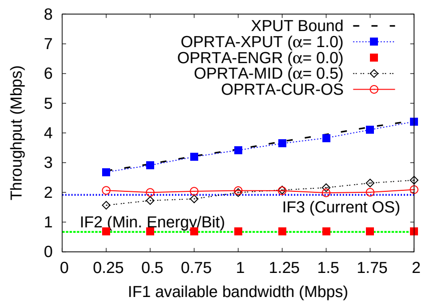


(a) Throughput

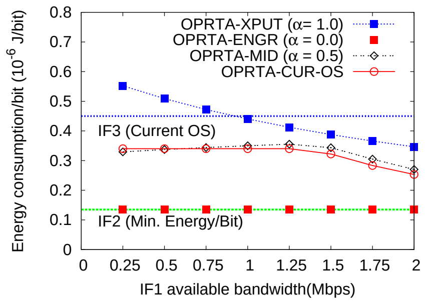


(b) Energy consumption per unit data

Fig. 3. Impact of changing the stream-type mix ($\gamma = 0$ for all steams connection-based, i.e zero network support).

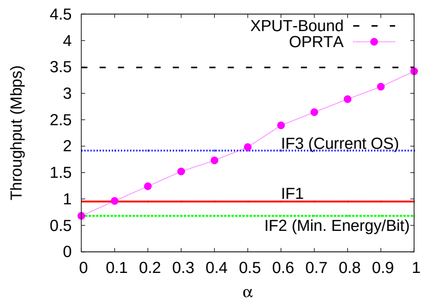


(a) Throughput

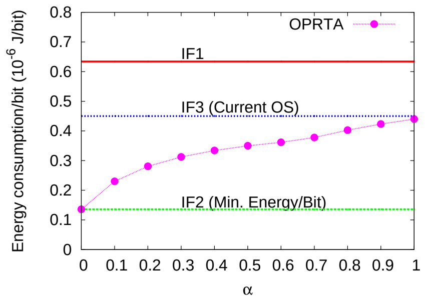


(b) Energy consumption per unit data

Fig. 4. Impact of interface dynamics (bandwidth) on performance.



(a) Throughput



(b) Energy consumption per unit data

Fig. 5. Impact of changing the utility value (α) on performance.

OPERETTA's goal is to minimize the energy consumption under a given throughput constraint, it favors solutions that leads to minimum energy. This is why it sacrifices throughput for the gain in energy as γ increases. (4) Finally, to minimize energy, OPERETTA running in the minimum energy mode, $\alpha = 0$, utilizes the interface with the minimum energy per bit, regardless of its throughput. Note that what matters is the energy consumed per bit and not the energy consumed per unit time (power). The former takes into account the fact that a faster interface that consumes slightly more power will consume less overall energy to transfer the same amount of data compared to a slow interface that consumes less power. For the remainder of the paper, we fix γ at 25%.

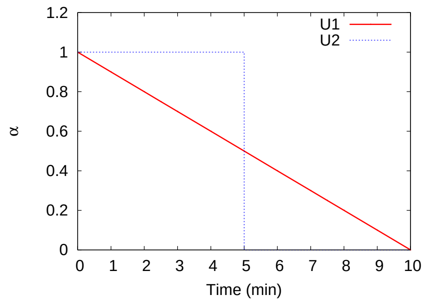
2) Impact of Interface Dynamics: We now study the effect of dynamically changing the interface bandwidth. For this, we fix the bandwidth of IF_2 and IF_3 as shown in Table III and change the bandwidth of IF_1 from 0.25 to 2 Mbps.

Figure 4 shows that different variants of OPERETTA, based on the user utility, behave differently under changing interface characteristics. The OPRTA-ENGR scheduler, whose goal is to minimize energy, always uses the interface with the minimum energy consumption per bit, regardless of the bandwidth. On the other hand, the OPERETTA variants that optimize for throughput (OPRTA-XPUT and OPRTA-MID) can leverage the increase of the IF1 bandwidth to enhance the overall throughput. We introduce here another variant (OPRTA-CUR-OS) whose goal is to achieve the same throughput of the current OS's (i.e., that of IF3) with the minimum energy consumption. This is highlighted in Figure 4(b). The break in the energy consumption for the OPRTA-MID and OPRTA-CUR-OS variants at IF1 bandwidth = 1.5 Mbps can be explained by noting that at this point IF1 energy consumption per bit becomes attractive. Therefore the two variants start using it for transmitting data.

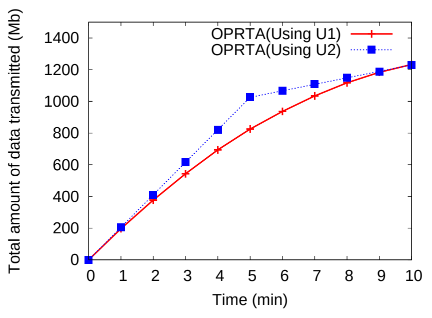
We note that the usage of each interface changes as the bandwidth of IF1 changes and becomes more attractive in terms of the energy consumed per bit for the OPRTA-CUR-OS variant. Even though the overall throughput may be constant, the used interfaces change to minimize the energy consumption as the system dynamics change. When IF1 bandwidth reaches 1.5 Mbps, its a/r ratio becomes better than IF3, and therefore, it completely replaces IF3.

3) User Utility: In this section, we show the effect of changing the parameter α on performance. Figure 5 shows that OPERETTA adapts to different user utility functions. When the user is more concerned with energy ($\alpha = 0$) OPERETTA can provide the minimum energy consumption (by scheduling all traffic on the interface with minimum energy consumption). On the other hand, when the user's main concern is throughput ($\alpha = 1$), OPERETTA leverages all interfaces to maximize the throughput while increasing the energy consumption.

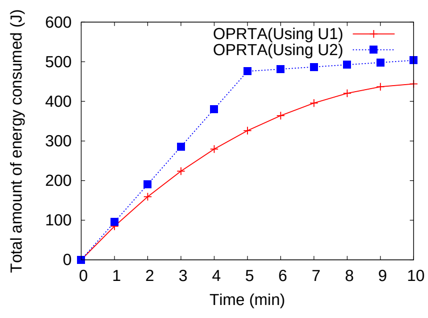
To provide further insight about the system dynamics, we show the instantaneous values of the data transmitted and energy consumed for two different user utility functions (Figure 6). (1) $U1$ represents the case where utility decreases linearly with time. This can be the case when utility is proportional to the remaining battery level. (2) $U2$ represents the case where the user's value of the energy consumption suddenly switches. This can be the case when the user sets a threshold on the minimum battery level for high performance after which he



(a) Different utility functions outputs

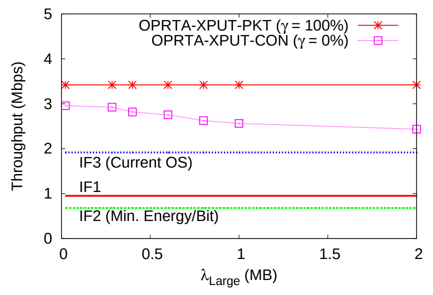


(b) Total data transmitted

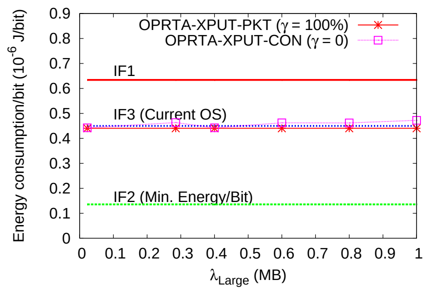


(c) Total energy consumption

Fig. 6. Effect of using different utility functions.

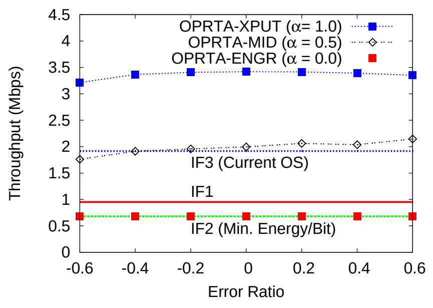


(a) Throughput

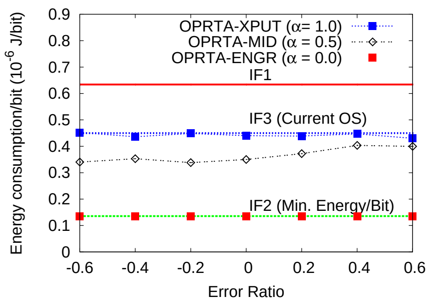


(b) Energy consumption per unit data

Fig. 7. Impact of changing the connection heterogeneously (λ_{Large}).

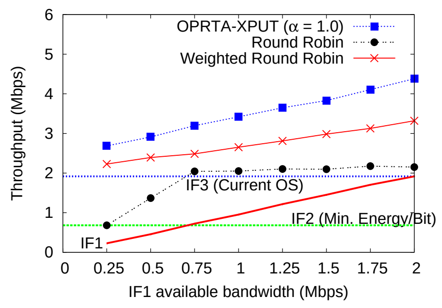


(a) Throughput

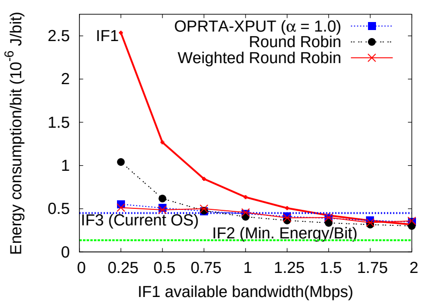


(b) Energy consumption per unit data

Fig. 8. Effect of application characteristics estimation error on the performance of OPERETTA.



(a) Throughput



(b) Energy consumption per unit data

Fig. 9. Comparison with baseline schedulers.

switches to an energy-saving mode.

The figure shows that the utility functions captures user intent; the linear utility function has both linear throughput and energy consumption. The step utility function changes both the throughput and energy consumption at the change point (t= 5 min). Overall, we observe that the smoother the utility function the better it performs.

4) Impact of the Connection Heterogeneity (λ_{Large}): Figure 7 demonstrates the effect of skewing the distribution of the connection length, i.e. fixing λ_{Small} and increasing λ_{Large}. We observe that as the connection lengths are more heterogenous, the connection-oriented scheduler suffers from the coarse grained scheduling granularity and deviates from the optimal performance. The packet-oriented variant ($\gamma = 100$), is not affected by the heterogeneity as its scheduling granularity is packet-based. This result further highlights that OPERETTA's performance can be incrementally enhanced as more servers become OPERETTA enabled.

5) Robustness to Estimation Error: In this section, we evaluate the robustness of OPERETTA to error in the estimation of application characteristics. We use a severe error model where the estimator consistently causes a fixed error ratio. Figure 8 shows that different variants are not sensitive to error in application characteristics estimation. Figure 8 also shows that the estimation error mainly impacts energy consumption. This result comes with an increase in throughput, which exceeds the user defined threshold. However, the impact of loss in optimality on throughput is negligible. This is because we have two different behaviors.

For OPRTA-XPUT, both overestimating the mean stream demand as well as underestimating it decreases the overall system throughput because all the interfaces are balanced and utilized to their maximum. Incurring estimation errors affects this balance, which would lead to the decrease in the overall systems throughput. However, we observe that underestimating the mean stream demand is more critical because it makes the scheduler push more streams on the low bandwidth interface. On the other hand, for OPRTA-MID we observe that underestimating the mean connection demand renders the scheduler unable to achieve the required throughput, while overestimating it makes the scheduler push more data onto the high bandwidth interfaces achieving 11% more throughput than needed with consuming energy at 114% of the optimal energy consumption. Similar results were obtained when we tested other less severe error models.

6) Comparison with Baseline Schedulers: In this section, we compare the performance of the OPRTA-XPUT ($\alpha = 1$) with the round robin and weighted round robin schedulers. We fix the bandwidth of IF_2 and IF_3 as shown in Table III and change the bandwidth of IF_1 from 0.25 to 2 Mbps.

Figure 9 shows that both the OPRTA-XPUT and weighted round robin schedulers exploit the heterogeneity of the interfaces to achieve high throughput. However, as the overall available bandwidth increases, OPRTA-XPUT's advantage increases as it takes into account the application characteristics. On the other hand, the round robin scheduler assigns the same amount of data to each interface. Therefore, the low bandwidth interface becomes a performance bottleneck. Both baseline schedulers are far from the optimal throughput.

In terms of the energy consumption per unit data, a similar trend is observed; the round robin scheduler performs the worst as it does not take the interface characteristics into account. The OPRTA-XPUT variant consumes almost the same energy as the weighted round robin with significant gain in throughput.

VII. Conclusion and Future Work

We proposed OPERETTA, an optimal and deployable bandwidth aggregation system for wireless devices. We presented the OPERETTA architecture and formulated the optimal scheduling problem that aims at minimizing the energy consumed to achieve a user-defined system throughput utility. We showed that the problem special structure allows it to be solved in an efficient time bounded by $o(m^2)$ in the worst case, where m is the number of interfaces. We also presented the details of our OPERETTA implementation showing its ease of deployment as a Windows executable.

In addition, we evaluated OPERETTA using both implementation and simulation. OPERETTA can be tuned to achieve different user utility goals. In the throughput maximization mode, it can provide more than 150% enhancement in throughput compared to the current operating systems, with no changes to the current legacy servers. This result comes with no increase in energy consumption. Moreover, the performance gain increases with the increase in percentage of OPERETTA-enabled servers, reaching the maximum achievable throughput with changes to as few as 25% of servers. We also showed that in order to achieve the same throughput as current operating systems, OPERETTA requires significantly less energy (saving up to 44%). We also compared OPERETTA to other schedulers and highlighted the reasons for the significant performance advantage that OPERETTA can achieve. This highlights its significant performance gains, incremental deployment, and ease of use.

For our future work, we plan to extend our system by optimizing for multiple objectives including cost and trust in addition to the throughput and energy consumption, addressing multi-user collaboration, and implementing our system on both Android and Linux operating systems.

References

[1] L. Magalhaes and R. Kravets. Transport level mechanisms for bandwidth aggregation on mobile hosts. *Urbana*, 51:61801.

[2] M. Zhang, J. Lai, A. Krishnamurthy, L. Peterson, and R. Wang. A transport layer approach for improving end-to-end performance and robustness using redundant paths. In *USENIX Annual Technical Conference*, 2004.

[3] H.Y. Hsieh and R. Sivakumar. A transport layer approach for achieving aggregate bandwidths on multi-homed mobile hosts. *Wireless Networks*, 11(1):99–114, 2005.

[4] Y. Dong, D. Wang, N. Pissinou, and J. Wang. Multi-path load balancing in transport layer. In *Next Generation Internet Networks, 3rd EuroNGI Conference on*, pages 135–142. IEEE, 2007.

[5] A. Argyriou and V. Madisetti. Bandwidth aggregation with SCTP. In *GLOBECOM'04*, volume 7, pages 3716–3721, 2004.

[6] K.H. Kim, Y. Zhu, R. Sivakumar, and H.Y. Hsieh. A receiver-centric transport protocol for mobile hosts with heterogeneous wireless interfaces. *Wireless Networks*, 11(4):363–382, 2005.

[7] D. Sarkar. A concurrent multipath TCP and its markov model. In *Communications, 2006. ICC'06. IEEE International Conference on*, volume 2, pages 615–620. IEEE, 2006.

[8] J.R. Iyengar, P.D. Amer, and R. Stewart. Concurrent multipath transfer using SCTP multihoming over independent end-to-end paths. *Networking, IEEE/ACM Transactions on*, 14(5):951–964, 2006.

[9] L. Magalhaes and R. Kravets. MMTP: multimedia multiplexing transport protocol. *ACM SIGCOMM Computer Communication Review*, 31(2 supplement):220–243, 2001.

[10] H. Sakakibara, M. Saito, and H. Tokuda. Design and implementation of a socket-level bandwidth aggregation mechanism for wireless networks. In *Proceedings of the 2nd annual international workshop on Wireless internet*, page 11. ACM, 2006.

[11] B.D. Higgins, A. Reda, T. Alperovich, J. Flinn, TJ Giuli, B. Noble, and D. Watson. Intentional networking: opportunistic exploitation of mobile network diversity. In *ACM Mobicom'10*, pages 73–84.

[12] K. Chebrolu, B. Raman, and R.R. Rao. A network layer approach to enable TCP over multiple interfaces. *Wireless Networks*, 11(5):637–650, 2005.

[13] D.S. Phatak and T. Goff. A novel mechanism for data streaming across multiple IP links for improving throughput and reliability in mobile environments. In *INFOCOM 2002. Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 2, pages 773–781. IEEE, 2002.

[14] P. Rodriguez, R. Chakravorty, J. Chesterfield, I. Pratt, and S. Banerjee. Mar: A commuter router infrastructure for the mobile internet. In *ACM MobiSys'04*, page 230.

[15] K. Habak, M. Youssef, and K.A. Harras. DBAS: A Deployable Bandwidth Aggregation System. *International Conference on New Technologies, Mobility and Security (NTMS'12)*, 2012.

[16] K. Habak, M. Youssef, and K.A. Harras. G-DBAS: A Green and Deployable Bandwidth Aggregation System. In *IEEE Wireless Communications and Networking Conference (WCNC'12)*. IEEE, 2012.

[17] T. Pering, Y. Agarwal, R. Gupta, and R. Want. Coolspots: reducing the power consumption of wireless mobile devices with multiple radio interfaces. In *Proceedings of the 4th international conference on Mobile systems, applications and services*, pages 220–232. ACM, 2006.

[18] W. Hua, J. Ohlund, and B. Butterklee. Unraveling the mysteries of writing a winsock 2 layered service provider. *Microsoft Systems Journal*, pages 96–113, 1999.

[19] M. Carson and D. Santay. Nist net-a linux-based network emulation tool. *Computer Communication Review*, 33(3):111–126, 2003.

[20] J. Erman, A. Mahanti, M. Arlitt, and C. Williamson. Identifying and discriminating between web and peer-to-peer traffic in the network core. In *ACM WWW'07*.