Distributed Link Bonding for Hybrid Wireless Networks

Hooman Javaheri, Guevara Noubir, *Senior Member, IEEE*, Yin Wang

Abstract—In this paper, we explore a distributed network-layer cooperation strategy called Distributed Link-Bonding (DLB). Our focus is on heterogeneous networks with devices equipped with two types of radio frequency (RF) links: short-range high-rate interface (e.g., IEEE802.11), and a long-range low-rate interface (e.g., cellular). The principle behind this strategy is to exploit the benefits from both traffic diversity and channel diversity through traffic-multiplexing, where multiple nodes pool their long-range links together to improve their effective throughput and the network efficiency. We consider several types of traffic and show that the proposed technique can achieve significant improvements of delay, perceived throughput, and network utilization through analysis, simulations and prototypes. We discuss several network architecture and finally propose *Neptune*, an architecture and implementation of the DLB concept that combines transparency to the applications, simple scheduling, and robustness and efficiency for TCP.

Index Terms—Diversity, Cooperation, Hybrid Wireless Networks.

1 Introduction

Mobile phone *data services* are transforming the mobile world with applications such as web browsing, video/music streaming, cloud computing, and distributed online games. Although, a set of third generation technologies have been developed and deployed for several years, the quality of service is still unsatisfactory. Users experience slow mobile Internet access, dead-signal areas and localized poor system performance. This is because the wireless link quality can change significantly over space and time, due to shadowing, multipath fading and interference. Moreover, the user traffic pattern might also be quite different over time. As a result, some long-range wireless links are highly congested, while others are underused. Therefore, the overall network utilization is low.

To solve these problems, instead of deploying more base stations, which is the conventional approach adopted by operators to improve the service coverage, we propose to exploit user cooperation and benefit from both channel diversity and traffic diversity. We call this strategy Distributed Link-Bonding (DLB). In DLB, the long-range links of the cooperating mobile nodes are bundled together using the local high-speed wireless network and the traffic from all the nodes in cooperation are being multiplexed through this bundled link. As a result, the traffic load can be balanced among all long-range links. Therefore, if one mobile node is in a fading or shadowing condition, it can still reroute the traffic through other nearby nodes. Furthermore, due to multiplexing, the average delay, the perceived throughput and network utilization can also be significantly improved. More potential is achievable if the base station is aware that the mobile nodes are in a DLB mode, as the base station can always pick the nodes with the best signal condition to transmit, and the data will be eventually forwarded to the nodes in weak signal condition. For the broadcast or multicast traffic, the DLB system would work better, because the data only need to transmit once through the long-range wireless links and the data will be re-broadcast in the local high-speed network. For example, in the case of live TV broadcast, one node might receive very low quality videos due to its limited bandwidth, but with DLB a group of nodes pool their links together to receive the high quality videos. DLB can be also useful in military and disaster rescue scenarios, where constrained mobile devices can pool their resources to build a more powerful long-range distributed link.

The proposed DLB technique has the advantage of being practical in terms of implementation and deployment. It can be implemented as a software middleware on the existing hardware and be transparent to the applications. Therefore, the existing applications would have an improved performance without requiring any awareness or modification.

For the proposed cooperation to become popular, there is a need to develop mechanisms that reward cooperative nodes, detect and punish selfish behavior, and protect privacy [1], [2]. Such mechanisms are very important for the success of a distributed diversity system, but they are out of scope of this paper, and will be the subject of our future research. Our main focus is on understanding and demonstrating the potential of DLB.

In Section 2, we first present the system model and our cooperative approach. Then, we analyse the performance of DLB in serial multiplexing mode. We present the simulation results for both serial multiplexing mode and parallel multiplexing mode with exponential traffic and

• *H. Javaheri, G. Noubir and Y. Wang are with the College of Computer and Information Science, Northeastern University, Boston MA, 02115. E-mail: {hooman, noubir, yin}@ccs.neu.edu*

heavy-tail traffic. In Section 3, we discuss several possible architectures for DLB, and describe two proxy-based DLB solution – tunnelling-based DLB and Neptune DLB. In the end, we present our prototype experimentation results.

1.1 Contributions

We explore the potential of cooperation under Distributed Link-Bonding, which exploits both channel diversity and traffic diversity, to improve the performance of the hybrid wireless networks. We show that the proposed strategy is a simple and effective method to improve the signal coverage, delays, perceived throughput and network utilization through analysis, simulations and experiments. We also develop prototypes for the proposed DLB system. Our prototypes demonstrate that the proposed architecture and mechanisms can be implemented on current mobile phones, transparent to applications, and easy to use with simple user controls. Furthermore, we show that although the used links can have significant differences in delays, our solution does not suffer from the typical out of order packet problem that usually degrades TCP performance.

1.2 Related work

In this section, we provide a quick overview of some of the research that has been done and that overlaps with the concepts, problems, and solutions that we address in this paper.

The fundamental diversity and multiplexing principle in communication systems have been studied in [3], [4]. Link Aggregation or *Multi-Link Trunking* (MLT) on the link layer for wired Ethernet is proposed in [5]. It allows multiple physical links to be grouped into one logical Ethernet link to provide fault-tolerance and high-speed communication between routers. Multi-homed transport protocols such as the IETF *Stream Control Transmission Protocol* (SCTP) [6] provides the link redundancy and multi-sessions, but it does not account for the wireless environment with multiple types of air-interfaces. Furthermore, it is not commonly deployed in the current Internet and requires the application to be aware of the network protocol to benefit from fault-tolerance and parallel sessions. Indirect TCP (I-TCP) [7], used to improve the performance of TCP for mobile wireless communications, splits a single TCP connection into two independent sequential TCP links to prevent the performance fluctuation caused by wireless communications from propagating to the fixed network. One of the advantages of this method is that it requires no change to the TCP protocol on the hosts. One aspect of our Neptune prototype for the DLB implementation is inspired by this idea (See Section 3.3) and used to embed our distributed diversity strategy. Wireless Mesh Networks have also been proposed as a cooperative approach to wireless communications, and studied in [8], [9], [10]. It allows neighbors to connect their home networks together to share their Internet access via gateways that are distributed in their neighborhood. Within wireless mesh networks, a multi-radio unification protocol for IEEE 802.11 wireless networks is proposed in [11]. More recently, several aspects of improving cellular networks through ad hoc networks relaying were investigated [12], [13], [14]. Most earlier work focusses on the capacity improvement (unicast and multicast) using *relaying* (usually requiring modifications to the cellular network architecture) and not on the potential gains of *bonding* the radios of multiple moderate-performance cellular links.

In our previous work [15], [16], we have introduced several distributed cooperation techniques based on signal combining and implemented on physical layers. In this paper, we proposed a new type of cooperation based on traffic multiplexing. To the best of our knowledge, this is the first work that proposes an architecture for Distributed Link-Bonding over two radio interfaces, provides an analysis and simulations to determine the expected delays and throughput, and a software middleware implementation for current mobile phones that is transparent to existing mobile applications.

2 Distributed Traffic Multiplexing

In this section, we propose DLB and explore its potential. The principle behind this strategy is to exploit the benefit from both traffic diversity and channel diversity through traffic-multiplexing, where multiple nodes pool their long-range links together to improve their effective throughput and the network efficiency. We first motivate the benefits of traffic multiplexing and link-bonding techniques; then introduce the model used in the analysis; and present the analytical and simulation results. In Section 3, we present several possible methods to implement DLB and our prototype implementation on some current smart phones.

2.1 System Model and Approach

The considered DLB system consists of a group of m nearby mobile nodes or mobile stations (MS), and base stations or base transceiver stations (BTS). The base stations are controlled by the base station controller (BSC), which dictates the carrier frequencies, communication power and rate, etc. The base stations are also connected to the backbone which leads to the telephone network and the Internet. Communication between mobile stations and base stations is through long-range low data-rate links (e.g., GPRS, EDGE, HSDPA, 1xEvDO). Mobile stations can also communicate with each other through short-range high data-rate links (e.g., WiFi). In this paper, we focus on the case where all actively cooperating mobile stations are within one-hop of each other through the high data-rate link. For example, in Figure 1 a base station BTS_1 is communicating with a mobile station MS_1 and another mobile station MS_2 in the vicinity through long-range low data-rate links; the links from

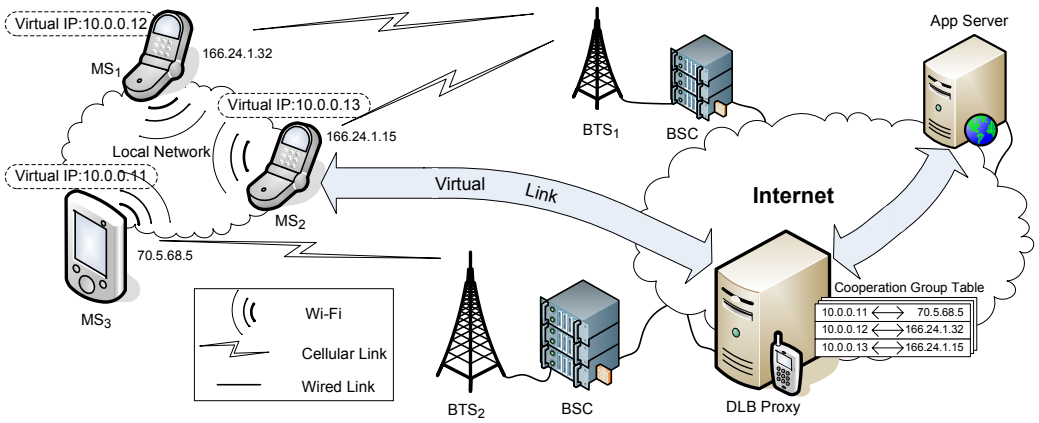

Fig. 1. Configuration of the proxy based DLB platform.

MS_1 to MS_2, from MS_2 to MS_3, and from MS_3 to MS_1 are short-range high data-rate links.

The application on the mobile nodes generates requests, and sends them to the remote application servers through their long-range cellular links. The responses such as web pages from the application servers are returned back to the mobile nodes. In the proposed DLB system, we assume the long-range cellular links are limited by low speed at a certain data-rate. For our analysis, we consider the following system characteristics and parameters:

User Traffic Distribution: For our analysis and simulations, we assume that the nodes generate requests according to specific distributions. We consider that a node is blocked until its request is fulfilled. We consider a *Blocked-Poisson* distribution for the user traffic (note that this is different from the traditional Poisson distribution). Here a user generates requests according to a Poisson process with rate λ but blocks until the current request is fulfilled. This process is motivated by the fact that mobile phones' users usually make one request at a time (e.g., browse a web page and wait until it is completely loaded before making a new request).

Service Time: This corresponds to the size of a web page, or the length of a session. We consider two types of distributions: *exponential* and *heavy-tail*. The exponential distribution is commonly used to model the traffic in networks and for queuing analysis, but the investigation of the Internet traffic has suggested that the request durations on the Internet are often according to heavy-tail distributions [17], [18], [19]. For heavy-tailed traffic, in terms of the size or duration of the requests there are not only many short requests, but also a few very long requests, which are not exponentially bounded [20], [21]. In contrast, the exponential traffic with the same mean would have far more medium size requests.

Cooperation Modes: We consider two cases: *non-cooperative* mode and *cooperative* mode. In the *non-cooperative* mode all the traffic generated by each node/user is transmitted over the nodes' own long-range links. In the *cooperative* mode, the traffic generated by the cooperating nodes is multiplexed over all long-range links. The combined capacity can be viewed as the sum of the capacities of all the nodes' long-range links. In this mode, we consider two ways of multiplexing the user traffic:

- *Serial-multiplexing* consists of queuing each user/node's request until all previous requests are serviced. This means that the request is processed at the combined rates of all long-range interfaces.
- *Parallel-multiplexing* processes all requests simultaneously. The speed of processing a request depends on the number of active requests in the system.

While parallel-multiplexing is more realistic from an implementation and deployment perspective, it is harder to evaluate analytically. We evaluate the serial-multiplexing analytically and compare it to the parallel-multiplexing simulations. We also experimentally evaluate the parallel-multiplexing.

Evaluation Metrics: We study two characteristics of these systems:

- *Average Service Delay:* This is the average amount of time that a packet stays in the system. The service delay is the waiting period plus the transmission period. The waiting period starts when a packet is dispatched from the application layer for transmission. If the queue is not empty or if the channel is occupied, the packet is put on hold until it reaches the head of the queue and the channel is cleared. The transmission period is the actual amount of time it takes to finish transferring the packet. This period depends on the packet length and the transmission rate.
- *Perceived Throughput:* This is the expected ratio of

request length and the processing time plus the waiting period. In other words, it is the throughput that users actually experienced. It can be defined as

$$PT = E(\frac{L}{\frac{L}{\mu'} + D}),$$

where L denotes the request length, D is the waiting period before a request's processing starts, and μ' is the aggregate service rate.

2.2 Stationary Regime for Blocked-Poisson Traffic, and Exponential Service Time

In this part, we analyse the system in non-cooperative mode and cooperative DLB serial-multiplexing mode using *Queuing theory* and a *Markov-chain* model. Serial-multiplexing is a simplified model, so we can derive its performance analytically. The performance of parallel-multiplexing will be evaluated through simulations (see Section 2.3) and prototype experiments.

Non-Cooperative Mode: Each node is working independently. Assume the incoming traffic is a Poisson process with arrival rate λ, and the transmission rate of the long-range link is μ. And the system is running in the blocking mode, in which no new requests would be generated if the current request is still not complete. It can be modelled as a two-state Markov-chain, Figure 2.

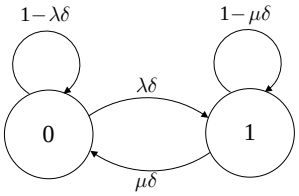

Fig. 2. Markov chain for each node in non-cooperative mode

A straightforward analysis gives the following results. *Global Balance Equations*:

$$P_1 = \rho \cdot P_0, \quad \rho = \lambda/\mu$$

The average number of requests in the system is:

$$N = \lambda/(\mu + \lambda)$$

It can be easily shown that the perceived throughput is equal to μ.

DLB Serial Multiplexing Mode: For ease of analysis, we assume that all the cooperative nodes have long-range interfaces with the same characteristics (i.e., same average transmission rate μ). The incoming traffic from each node is a Poisson process with the same arrival rate λ. And each of the node is also running in the blocking mode. This system can be modelled as a Markov chain with $m + 1$ states (see Figure 3). State i corresponds to the state when the system has i pending requests. In the serial-multiplexing mode, only one request is processed while other requests are queued. Since the system has m cooperating nodes, the processing speed is m times faster in comparison to the non-cooperative mode.

Let N_k denote the number of requests in the system at time $k\delta$ (including the one being processed), P_i denotes the stationary probability of being in state i, and $P_{i,j}$ denotes the transition probability from state i to state j.

$$\begin{aligned} P_{i,j} &= P\{N_{k+1} = j | N_k = i\} \\ P_{0,0} &= P\{0 \text{ requests arrive from } m \text{ nodes}\} \\ &= e^{-m\lambda\delta} \\ &= 1 - m\lambda\delta + o(\delta) \end{aligned}$$

$$\begin{aligned} P_{i,i\geq 1} &= P\{0 \text{ arrivals from } m-i \text{ nodes, } 0 \text{ departs}\} \\ &= e^{-(m-i)\lambda\delta} \cdot e^{-m\mu\delta} \\ &= 1 - (m-i)\lambda\delta - m\mu\delta + o(\delta) \end{aligned}$$

$$\begin{aligned} P_{0,1} &= P\{1 \text{ requests arrive from } m \text{ nodes}\} \\ &= m\lambda\delta \cdot e^{-m\lambda\delta} \\ &= m\lambda\delta + o(\delta) \end{aligned}$$

$$\begin{aligned} P_{i\geq 1,i+1} &= P\{1 \text{ arrives from } m-i \text{ nodes, } 0 \text{ departs}\} \\ &= (m-i)\lambda\delta \cdot e^{-(m-i)\lambda\delta} \cdot e^{-m\mu\delta} \\ &= (m-i)\lambda\delta + o(\delta) \end{aligned}$$

$$\begin{aligned} P_{1,0} &= P\{0 \text{ requests arrive from } m-1 \text{ nodes,} \\ &\quad 1 \text{ departs}\} \\ &= e^{-(m-1)\lambda\delta} \cdot (1 - e^{-m\mu\delta}) \\ &= m\mu\delta + o(\delta) \end{aligned}$$

$$\begin{aligned} P_{i>1,i-1} &= P\{0 \text{ arrives from } m-i \text{ nodes, } 1 \text{ departs}\} \\ &= e^{-(m-i)\lambda\delta} \cdot m\mu\delta \cdot e^{-m\mu\delta} \\ &= m\mu\delta + o(\delta) \end{aligned}$$

The *Global Balance Equation* is:

$$P_i \cdot (m-i)\lambda \quad = \quad P_{i+1} \cdot m\mu$$

Let $\rho = \lambda/\mu$. Then, from the balance equations:

$$\begin{aligned} P_{i+1} &= \frac{m-i}{m}\rho \cdot P_i \\ P_i &= \frac{(m-1)!\,\rho^i}{(m-i-1)!\,m^i} \cdot P_0 \end{aligned}$$

Since $\sum_{i=0}^{m} P_i = 1$, we have

$$P_0 = [\sum_{i=0}^{m} \frac{(m-1)!\,\rho^i}{(m-i-1)!\,m^i}]^{-1},$$

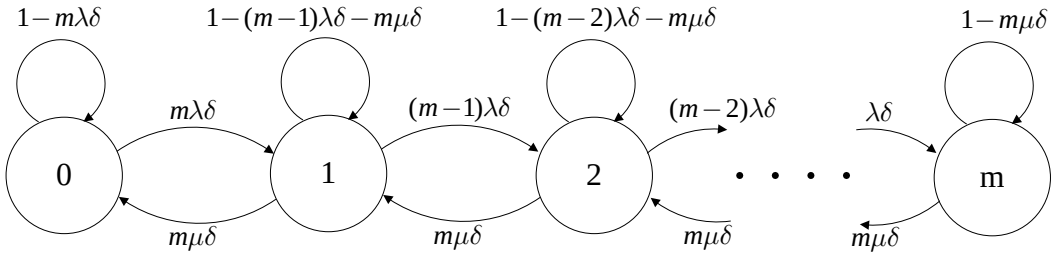

Fig. 3. Markov chain for m cooperating nodes in serial-multiplexing mode.

and

$$P_i = [(m-i-1)!\,(\frac{m}{\rho})^i \cdot \sum_{j=0}^{m} \frac{\rho^j}{(m-j-1)!\,m^j}]^{-1}$$

The average number of requests in the system is:

$$N = \sum_{i=0}^{m} i \cdot P_i$$

To compute the average delay, let us consider a newly arrived request. We first compute the current system state probability: $P(N_t = i|N_t \neq m)$, $0 \leq i < m$. This corresponds to the probability that i requests are already in the system. The number of queued requests cannot be m or larger, because users are blocked after each request. Therefore, if there are already m queued requests no new request would arise.

$$\begin{aligned} P(N_t = i|N_t \neq m) &= \frac{P(N_t = i, N_t \neq m)}{P(N_t \neq m)} \\ &= \frac{P_i}{1-P_m} \end{aligned}$$

For each $P(N_t = i|N_t \neq m)$ the delay is $X_1 + \cdots + X_i + X$, where X_i denotes the time to process (transmit) existing request i and X is the time to process the new request. The average time for processing each of these requests is $(i+1)/(m\mu)$. This holds true even for the request being processed because of the memoryless characteristic of the exponential distribution. Note that we cannot compute the average delay directly using Little's Theorem, because the effective arrival rate of the system is unknown but less than λ, which is due to the blocking nature of our request generation process. Combining all the cases, we obtain the following average delay for finishing a transmission request:

$$\begin{aligned} T &= \sum_{i=0}^{m-1} (X_1 + \cdots + X_i + X) \cdot P(N_t = i|N_t \neq m) \\ T &= \sum_{i=0}^{m-1} (\frac{i+1}{m\mu}) \cdot \frac{P_i}{1-P_m} \end{aligned}$$

Using the same approach the perceived throughput is as the following.

$$PT = \int_0^{\infty} \frac{1}{\mu} e^{-\frac{L}{\mu}} \sum_{i=0}^{m-1} \frac{L}{(\frac{L+i}{m\mu})} \cdot \frac{P_i}{1-P_m}\, dL$$

Our simulation of the Blocked-Poisson traffic with exponential service time and serial-multiplexing exactly matches the above analytical results (See Figures 4 and 5). These results will be discussed the next section.

2.3 Traffic Pattern and Simulation

Our simulation testbed is built up using *Matlab* and *Simulink*. We simulate two types of traffic – Blocked-Poisson and heavy-tail in both serial-multiplexing and parallel-multiplexing. In this section, we present the simulation results in terms of service delay and perceived throughput.

2.3.1 Blocked-Poisson Traffic (Serial vs. Parallel-Multiplexing)

Depending on the types of applications, parallel-multiplexing might be more practical and closer to the reality in comparison with serial-multiplexing. However, the analysis of parallel-multiplexing is more complex, and it is difficult to obtain a closed-form formula for it, because in such DLB system the service rate for completing a request depends on the number of requests within the system.

To evaluate the performance, we simulate both multiplexing schemes. We normalize the average service time per request to 1 unit of time (i.e., $\mu = 1$). We vary the arrival rate for each node between $\frac{\mu}{10^3}$ and $10^3\mu$. Note that we can use the arrival rates higher than the service time because our arrival process is *Blocked*-Poisson (the length of the queue never exceeds m). For each plotted point, we run the simulation for 500,000 units of time.

As reported in Figures 4 and 5, at low and moderate load both schemes show a significant reduction in delay and increase of perceived throughput. We also observe in some situation the serial-multiplexing slightly outperforms the parallel-multiplexing. This is because the service delay consists of the waiting time and the transmission time. Although in the serial-multiplexing case the request would wait if another request is under transmission, this would not slow down the request under transmission. Overall, serial-multiplexing performs

better. For example, assume there are 2 requests with the same size arrive in the system, and system service rate is 1 request per unit of time. So, in serial-multiplexing, one request takes 1 unit of time, and the other request takes 2 units of time (1 unit of time for waiting and 1 unit of time for transmitting). The average is 1.5 units of time. In contrast, in parallel-multiplexing, both requests takes 2 units of time. However, we note that serial-multiplexing exhibits a higher variance of service delay (jitter).

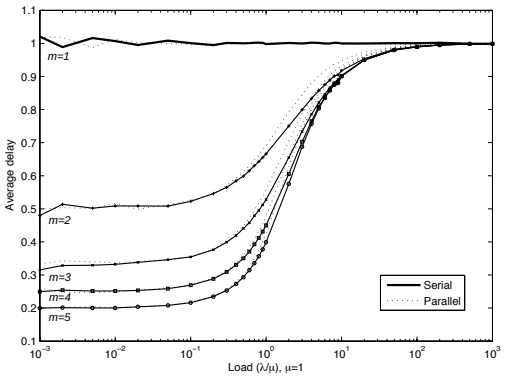

Fig. 4. Average delay per packet for serial and parallel-multiplexing with exponentially distributed service time.

2.3.2 Heavy-Tail Traffic

The distribution of the size of web pages is commonly modelled as more of a heavy-tail than an exponential distribution. Because the analysis of the cooperative mode is much harder for this type of traffic, we also use simulations to quantify the performance improvement resulting from DLB. In our simulations, we use Pareto distribution, which is a popular distribution for modelling more realistic traffic which exhibits a heavy-tail [20], [21]. Let l be the random variable that represents the duration of the request. According to the Pareto distribution, the probability density function for l is

$$p(l) = \frac{ab^a}{l^{a+1}}, \quad l \geq b,$$

where a and b are the parameters, which define the shape and scale of the distribution respectively.

Figure 6 indicates that under heavy-tail traffic, significant increase of the perceived throughput can be achieved when the load is low or moderate. We also observe that the serial-multiplexing outperforms the parallel-multiplexing, and in some situations the difference is substantial. For example, the parallel-multiplexing results in up to 45% lower perceived throughput than the serial-multiplexing compared to a 15% in exponential traffic, see Figure 7 and Figure 8.

We also compared the performance of DLB under heavy-tail and exponential traffic. Figures 9 and 10 show the performance of serial-multiplexing DLB under both traffic for the same average request length. In some

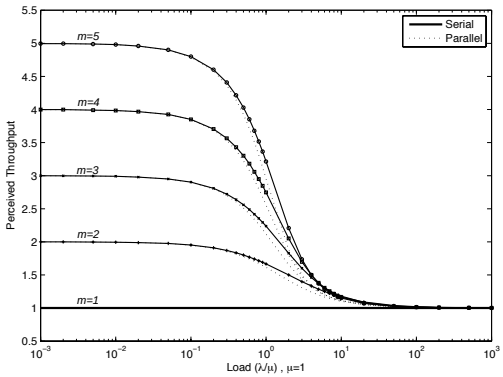

Fig. 5. Perceived Throughput as a function of load for exponentially distributed service time and $m = 1, \cdots, 5$.

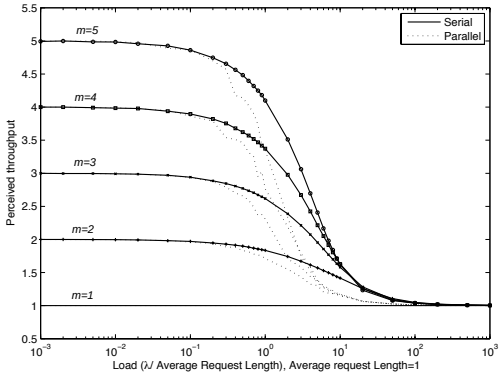

Fig. 6. Perceived throughput as a function of load for heavy-tail traffic and $m = 1, \cdots, 5$.

situation such as moderate load, heavy-tail traffic results in a perceived throughput up to 80% better than exponential traffic. This is because for the heavy-tailed traffic there are many short requests, but also a few very long requests. In contrast, the exponential traffic with the same mean would have a far more medium size requests. And transmitting long requests is more efficient than short requests, given the same total amount of data. For example, assume there are 2 requests each with 1 unit of size arriving at the system, and the system transmit rate is 1 unit of size per unit of time. So one request takes 1 unit of time, and the other request takes 2 units of time (1 unit of time for waiting and 1 unit of time for transmitting). So they together take 3 units of time. In contrast, assume one long request with 2 units of size, , which is the same amount of data as the previous case, it would take only 2 units of time to complete.

2.4 Scheduling and Rate Allocation

Each of the long-range link has different speed and delay. Rate allocation is important to the DLB system, because it decides how the packets from the traffic source are scheduled and routed through the cooperative network.

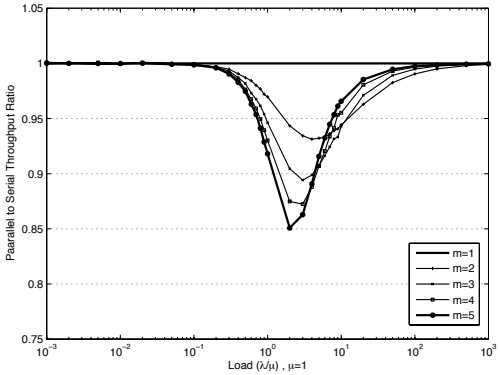

Fig. 7. Ratio of perceived throughput of parallel to serial-multiplexing for exponentially distributed service time.

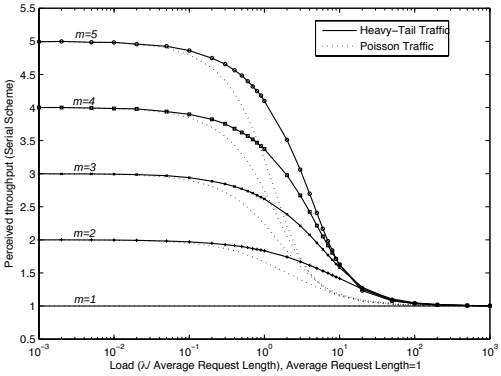

Fig. 9. Comparison of perceived throughput under exponential and heavy-tail traffic for serial-multiplexing. Both traffics have the same average request length.

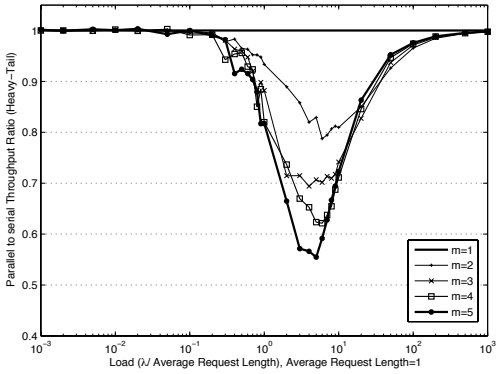

Fig. 8. Ratio of perceived throughput of parallel to serial-multiplexing for heavy-tail traffic.

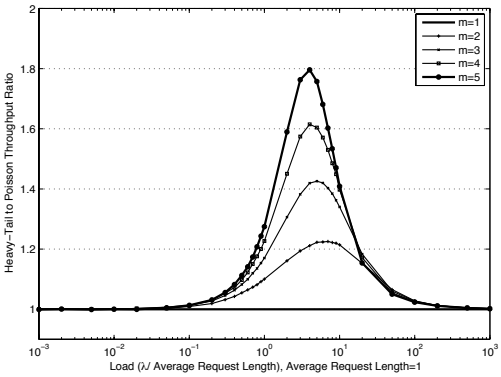

Fig. 10. Ratio of perceived throughput for heavy-tail traffic to exponential using serial-multiplexing. For the same average request length, heavy-tail traffic results in substantially better throughput.

Failing to have such a rate allocation algorithm, which accounts for the network condition, would cause the local network to congest on some links while underuse other links. Thus, it degrades the overall system performance.

Fortunately, this problem can be easily addressed in our cooperative environment. This is because the proposed DLB system has a simple topology that there is only one hop between cooperative nodes through the local high-speed network. Besides, the two types of air-interface used in the DLB systems are orthogonal, so the local wireless communications would not interfere with the long-range wireless communications.

The strategy we adopt in the DLB system is a water-filling algorithm (also called local-queue balancing [22] or back-pressure [23]). For each cooperating node the long-range communication queue status (i.e., length and delay) is monitored and reported to the master node. When the traffic from the application layer arrives at the DLB packet dispatcher of the master node, it measures the queue status for each assisting nodes and dispatch the packet to the assisting node with the best queue condition such as minimum queue length or shortest queue delay. For ease of implementation, our prototypes adopt the queue length as the metric. This strategy can improved to reflect the original local-queue balancing and back-pressure techniques utilising different queues for different destinations.

This rate allocation algorithm has the benefit that the channel condition can be calculated quickly and accurately as the queue length or delay is an effective measure of the channel condition. Besides, the algorithm is relatively simple, it would not cause much overhead on the CPU or network bandwidth. However, dispatching packet with such method would not prevent the packets from reaching the destination out of order, which would cause severe degradation to the performance of existing transport layer protocols such as TCP. In Section 3.3, we address the packet re-ordering issue in detail, propose solutions, and implement them within our DLB prototype.

3 Distributed Link Bonding Prototyping

In this section, we first discuss the potential architectures and approaches for implementing DLB. We present a tunnelling-based DLB prototype and the Neptune prototype. Both prototypes are implemented as system plug-ins that can run in the current network stack. We discuss the problems we have encountered during prototyping and propose solutions. Finally, we present our experimentation results and the achieved performance of DLB on smart phones.

3.1 Cooperative Solutions

In the previous section, we show that the proposed DLB system can significantly improve the network efficiency and boost the perceived throughput for current cellular systems. Several approaches are possible to implement DLB, such as *DLB-Aware Applications*, *DLB-Aware Network Layer*, *Multi-homed Transport Protocols*, and *Proxy Based Protocols*. All these approaches can be implemented in software on existing hardware. However, some of them necessitate modifications of existing applications or existing Internet protocols. These are undesirable constraints. In our implementation, we built two prototypes, a - *Tunneling Based Proxy* and the - *Neptune* prototype. Both prototypes not only run on existing hardware, but also impose no changes on the current Internet. Besides, our prototypes are implemented as system plug-ins, and can be turned on and off by the user with a single click. They are transparent to the application layer, so most of the existing applications can benefit from DLB without any modifications.

We first discuss the possible schemes for implementing DLB:

- **DLB-Aware Applications:** DLB can be implemented at the application layer. Cooperation at the application layer requires modifications of the application or a complete rewrite of the application. There are many peer-to-peer cooperative applications. For example, Skype uses cooperation between nodes to improve the call quality, raise the call completion rate and traverse firewalls; PPLive uses cooperation between nodes to deliver fast online video streaming; BitTorrent uses cooperation to accelerate file download.
  In the DLB scenario, for upload traffic, whenever an application client initiates a data connection, the application client splits the data and sends it to a list of cooperating nodes. The data is forwarded to the destination application server, and finally reassembled by the application server. The download traffic flows in a similar way. However, this method is not generic and cannot be shared by all the applications. A lot of work that has to be done for each application, which makes it impractical and not scalable from a software engineering perspective.
- **DLB-Aware Network Layer:** Some modifications can be made to the network layer stack to support DLB. One way is to enable the loose source routing in the Internet Protocol. In the IP protocol, there are two header options - *Strict Source and Record Route* (SSRR) and *Loose Source and Record Route* (LSRR) to allow the packet sender to partially or completely specify the route that the packet takes through the network. Without specifying these options, each router on the Internet determines the routing path solely based on the packet's destination and its own routing table. We can leverage the loose source routing option to realize the DLB cooperation.
  The DLB cooperation can be implemented in the following way: When the traffic arrives at the network layer on a mobile node, for each packet a cooperating node's IP address is added to the loose source routing record field in the IP header options. This forces the packet to travel over one of the cooperating nodes. In such a way, the traffic can be balanced over all the cooperating nodes. In the same way, when the traffic is travelling back from the application server to the mobile node, the application server also needs to add a cooperating node to the source routing record to make sure the packets go through these cooperating nodes.
  Although this method looks promising, it still raises many issues. First of all, to implement this strategy, it not only requires modifications of the IP stack on the client side, but also requires the modifications of the IP stack on the application servers too. It is undesirable and unrealistic to make those changes on all the application servers throughout the Internet. Furthermore, the source routing options are usually blocked for the concerns of security attacks such as Internet address spoofing and Denial-of-Service.
- **Multi-homed Transport Protocols:** A typical multi-homed transport protocol is the IETF *Stream Control Transmission Protocol* (SCTP) [6]. Similar to UDP, SCTP is a message based transport protocol in contrast with TCP, which is a stream based protocol. SCTP provides reliable data transfer and partial ordering of the data delivery. Among many of the features that SCTP supports, multi-streaming is the one we are interested in. It allows multiple independent streams to be transmitted in parallel. For example, a web page may contain html files, javascript files, and images. SCTP allows transmitting those files simultaneously.
  The DLB cooperation can be implemented using SCTP. When the traffic arrives at the transport layer (for both uploading and downloading), the DLB system starts multiple SCTP streams in a way that each one passes through a cooperating node. SCTP ensures the reliable transmission of each of the streams. On the receiver side, those streams are eventually delivered to the application layer.
  One of the limitations of such a solution is that it re-

quires all the Internet servers to run SCTP. But SCTP is not universally deployed on the current Internet. Another drawback is that to leverage parallel multi-streaming, the application needs to be aware of it. It is not completely transparent to the application. The application has to make sure each of the streams is independent and can be transferred independently. As a result, this also requires the modifications of applications.

- **Proxy Based Protocols:** In order to avoid changing the server side network stack and applications, proxies can be introduced into the system to bridge between the existing systems and the DLB cooperative environment. A diagram of the proxy based DLB platform can be seen in Figure 1.
  In this approach, the system redirects the application's uplink traffic to an *Agent* running on the mobile client. The *Agent* dispatches the data to go through each of the cooperating nodes and the destination is set to be a *Remote Proxy* (or DLB Proxy) on the Internet. The *Remote Proxy* then acts as a remote *Network Address Translation* (NAT) and forwards the packets to the final destination, which is an application server. For the downlink, all traffic from the application server is sent to the DLB proxy, which then sends the traffic to a set of cooperating nodes, and eventually the *Agent* on each cooperating nodes relays it to the original recipient.
  Since all the traffic goes through this DLB proxy, a potential limitation of this method is that the proxy might become a bottleneck. Nevertheless, the advantages of this solution are promising and our prototypes adopt this proxy based approach. As the DLB cooperation gains momentum, more proxies can be deployed at the edge of the Internet and even at the server side.

3.2 Tunnelling-Based Architecture and Prototype

Our first prototyping attempt is a tunnelling-based architecture. It uses the packet encapsulation technique commonly used in *Virtual Private Networks* (VPN) to create a virtual tunnel between the local cooperating nodes and the *Remote Proxy*. The system implementation is based on the *Click Modular Router* [24], [25] and runs on Linux. For the ease of implementation (in this first prototype), we use laptops with cellular PC cards as the mobile stations. The detailed software and hardware configuration can be found in Section 3.4. In the second prototype – Neptune (See Section 3.3), we are able to run the DLB system on the Google Android-based mobile phones[26]. The DLB system consists of two parts: an *Agent* running in each mobile node and a *Remote Proxy* running on a server accessible from the Internet.

The *Agent* is a software plug-in to the OS of the mobile node. It contains several components:

- *Status Update:* A service to broadcast and collect the identity and the status of cooperating nodes. This allows each node to discover other cooperative nodes. It also allows other nodes to learn the status information such as the pending queue length, the link quality and delay. These will be used in the packet scheduler to decide which node should be used for forwarding the traffic.
- *Data Collection:* A module to capture all the data sent from mobile applications. This is implemented as a virtual network interface bound with a virtual IP address. After changing the routing table of the mobile node, all of the outgoing packets are sent through this interface, and therefore captured and processed by our system. This requires no modifications to the existing applications. Our implementation uses an IP over UDP encapsulation technique, which means that each IP packet is wrapped inside a UDP packet.
- *Packet Scheduler:* A module to decide which cooperating node (including itself) it should forward to, based on the rate allocation algorithm (Section 2.4). It then encapsulates the packet in a UDP packet and forwards it to the selected node.
- *Packet Forwarding:* A service on the node to receive the incoming UDP packets and forward them to the destination. Upon receiving a packet, the UDP header is removed to extract the original IP packet. If the destination of this IP packet is another cooperating node or the *Remote Proxy*, it encapsulates this packet in UDP again and sends it out. If the destination is itself, it delivers the packet to the application.

The *Remote Proxy* is a lightweight service which can be run on any Internet server. It acts like a NAT router providing packet forwarding and network address translation services.

When a UDP packet arrives at the proxy from a cooperating nodes, the proxy strips off the UDP header and gets the original IP packet. Then the proxy executes a *network address translation* (NAT) to modify the source address to its own address. Finally the packet is transmitted through the wired network and delivered to the destination. The reverse process from the destination back to the mobile node is similar. Upon receiving an IP packet from the destination, the proxy performs a NAT and changes the destination address to the original initiator and feeds it to the packet scheduler. The scheduler then decides which cooperating node it should forward to based on the rate allocation algorithm, encapsulates it in a UDP packet, and forwards it to the selected node. For a better performance, we recommend that it would be placed as close to the mobile nodes as possible. This would reduce the number of hops required to relay packets.

Switching the DLB service on and off is easy. It can be done by a start/stop of the *Agent* process and an update to the routing table. One of the reasons to choose IP over UDP is that the firewalls are placed between mobile nodes and the Internet by the cellular operators for security purposes. So we cannot use the techniques

like IP over IP or NAT directly because the modified IP packets will be dropped by the operator's firewall. Our experimentation result shows that the UDP packet encapsulation is fast with negligible overhead.

We have conducted several experiments with the tunnelling-based DLB system. The results are presented in Section 3.4. We found that it works well for the UDP traffic, but the TCP traffic performance is unstable. One serious problem with this implementation is that our tunnelling-based DLB prototype is a network layer system, so it is unaware of the packet ordering. This can significantly degrade the performance of TCP. Packets re-ordering can happen quite often when the traffic is split over multiple different operators. This was one the main motivations to use an alternative solution in our Neptune approach and prototype, which will be described in the next section.

3.3 Neptune Architecture and Prototype

As described previously, the performance of TCP can be significantly degraded due to the arrival of out-of-order packets [27]. In the DLB system, packet reordering can happen frequently as the long-range links of the cooperative nodes have different rates and delays (our experiments sometimes show several hundreds of milliseconds difference between two cellular links when using two different operators). In TCP, the out-of-order packets result in duplicate acknowledgements (DUPACKs) from the receiver. The sender, which receives these DUPACKs, cannot differentiate if they are due to an out-of-order delivery or that a packet was lost and DUPACKs are the result of subsequently received packets. Current TCP designs and implementations were optimized for wired networks where out-of-order arrivals are rare and packet loss due to transmission errors is small. So, TCP misinterprets the out-of-order packet delivery as a packet loss due to congestion. These assumptions are no longer true in our DLB wireless and heterogeneous cooperation environment.

A simple and quick fix is to intentionally drop the DUPACKs, but this would just be a band-aid solution. It does not solve the fundamental problem, and can cause side effects when dropping useful DUPACKs. Some other solutions [28], [29] use timestamps or additional TCP header bits to detect spurious retransmission and therefore eliminate the retransmission ambiguity. In [30], the authors propose RR-TCP to adaptively vary dupthresh to avoid false fast retransmits proactively. Other solutions such as TCP-DOOR [31], designed for MANET environment with occasional out-of-order packets, detects out-of-order packets by using additional sequence numbers. TCP-PR, proposed in [32], neglects DUPACKs completely, and relies solely on timers to detect packet loss. However, those solutions only consider the case of one single congestion path, which does not correspond to our DLB scenario where each link results in an independent transmission path.

According to the type of packet reordering that happens in our DLB system, where the packets are a mix of multiple independent transmission paths, we consider a solution that addresses the congestion control for each cooperative link separately. Our prototype Neptune is based on this idea.

As a proxy-based approach, Neptune shares a similar structure as our previously described tunnelling-based prototype. It also consists of two components: an *Agent* running in each mobile node and a *Remote Proxy* which is accessible from the Internet. Unlike the tunnelling-based prototype which deals with the packets at the network level, Neptune is working at the transport level, to resolve the packet re-ordering problem. The Neptune system is described below.

Inspired by Indirect-TCP [7], Neptune splits one TCP link into several connected TCP links. A local NAT is added to complete the data collection task. When the application initiates a TCP connection to a remote application server, NAT is applied to redirect the connection to a dummy service stub in the *Agent*. The application is unaware of this change, and feels as if it is connected to the remote application server. This is quite different from the tunnelling-based approach.

For the uplink, once the *Agent* captures the traffic stream of the application, the cooperation starts. It first transforms the traffic data into its own message format with a sequence number for each message. Notice that those messages are different from the TCP packets received from the application. The message is currently set to a fixed size of 4KB, which defines the minimum data unit to be transferred within Neptune. The scheduler then dispatches those messages to a group of cooperating nodes according to the rate allocation algorithm. The *Agents* in other cooperating nodes forward the messages to the *Remote Proxy* through independent TCP links. As a result, each of these links handles its own congestion independently. Eventually, the received messages from all the cooperating nodes are reassembled at the *Remote Proxy* according to their sequence numbers. The *Remote Proxy* then establishes another TCP connection to the destination application-server and forwards the data to it.

For the downlink, the data sent back from the application server can be obtained from the same TCP link which is connected to the application server. The *Remote Proxy* then transforms the traffic data into messages with sequence numbers. The scheduler dispatches them back to the cooperating nodes through the same TCP links that are previously connected from them. Each of the cooperating nodes forwards the messages to the source node. The *Agent* at the source node reassembles and delivers them to the application through the service stub.

This approach uses separate TCP links, which allows independent congestion control, for the links between each cooperating node and the *Remote Proxy*. From our experimental results (Section 3.4), we show that this approach can effectively avoid the performance fluctuation

found in the tunnelling-based prototype. As we can see, Neptune splits the original TCP connection from the application to the application server into three separate TCP connections: from the application to the *Agent* stub, from the *Agent* to the *Remote Proxy*, and from the *Remote Proxy* to the application server.

We implemented the *Agent* of Neptune on the Google Android platform. It runs as a background service, but it can be controlled through an Android Activity with UI. A screen shot of Neptune *Agent* controller is shown in Figure 11. Most of the job is done automatically in the background, and the user can simply switch it on and off through a single button. Note that in order to run Neptune, the user needs to have root privileges on the Android phone to gain permission for certain API calls.

Fig. 11. Neptune Agent controller on Google Android platform.

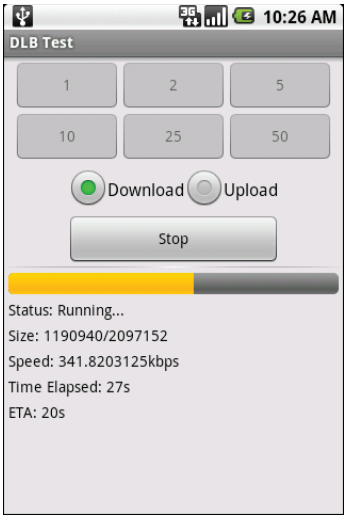

Fig. 12. A testing tool for DLB prototype.

3.4 Experimentation

First, we conducted several experiments with the *tunnelling-based* prototype. The experimental testbed consists of a computer which serves as the *Remote Proxy*, and the *Agents* running on multiple laptops with PC cards. We use two types of cellular PC cards: an HSDPA Sierra Wireless Aircard 860 from AT&T and a 1xEVDO Rev. A Pantech PX-500 from Spint (Table 1). The laptops are running the Fedora Linux with kernel 2.6.21. The Click Modular Router is version 1.6.0. For this prototype, the reason why we choose to use laptops, rather than a direct implementation on the phones, is that substantial kernel modifications are necessary in order to make the tunnelling-based prototype work, and the Click Modular Router is only available on the desktop Linux not on the Linux based mobile phones yet.

The measurements we carried aim at determining the overhead cost and providing a proof of concept for DLB. We developed and run a UDP speed test program that sends packets at a constant rate higher than what is sustainable by each of the cellular PC cards but still acceptable by the Internet. We use a 500KB/s data rate with a UDP packet size of 1KB. Table 2 summarizes our measurements. As we can see for the protocol with no ordering requirement such as UDP, our measurements indicate that *tunnelling-based* DLB prototype can be a very effective solution to boost the throughput with very little overhead. However, through our experiments, we also observe that for the TCP protocol, which has strict constraints on the packets ordering, our *tunnelling-based* DLB prototype fails to give consistent results and the performance fluctuated substantially (sometimes reaching a throughput even lower than what a single path can achieve). From our investigation, out-of-order packets are the source of the performance hit as discussed in Section 3.3.

We also conducted extensive experiments with our Neptune prototype. Neptune runs on existing smart phones and aims at solving the problems faced in the *tunnelling-based* DLB prototype. Neptune is implemented for the Google Android platform. In our experiments, we use two types of Android phones, the 1xEVDO Rev. A Motorola Droid from Verizon and the HSPA HTC G1 from T-Mobile (Table 1). We also include an Apple iPhone 3G from AT&T to cover more cases. Because Neptune is not designed for iOS running on the iPhone 3G, we also use a laptop tethered with the iPhone 3G, so we can use the PC implementation of Neptune. This is a similar method that we used in the experiments of the tunnelling-based prototype.

To measure the performance of Neptune, we also created a testing application (Figure 12) to test downloading/uploading with different sizes using HTTP (Most of the mobile traffic is HTTP e.g., web browsing, Internet radio, YouTube video). In the debug mode, the Neptune *Agent* can record the transient speed (at the second level), so that we can monitor the details of the running

Manufacture	Model	Technology	Operator	Software OS
Motorola	Droid	1xEvdo Rev.A	Verizon	Android OS 2.1 Linux Kernel 2.6.29
HTC	G1	HSPA	T-Mobile	Android OS 1.6 Linux Kernel 2.6.29.6
Apple	iPhone 3G	HSDPA	AT&T	iPhone OS 3.1.2
Sierra Wireless	Aircard 860	HSDPA	AT&T	Fedora 7 Linux Kernel 2.6.21
Pantech	PX-500	1xEvdo Rev.A	Sprint	Fedora 7 Linux Kernel 2.6.21

TABLE 1
DLB Testing Environment Setup

	HSDPA	1xEvDO	DLB Mode
9 am	920.8	1214.4	2037.6
12 pm	890.4	1249.6	2046.4
4 pm	935.2	1370.4	2148.0
8 pm	934.4	1336.8	2218.4

TABLE 2
UDP download speed tests for tunnelling-based DLB. (kbps)

performance for each cooperative node.

As expected, the specific results vary depending on the location and the time. Most of our experiments are conducted around the Northeastern University campus in Boston. Figure 13 and 14 show the performance of downloading and uploading of the Neptune DLB system with two Verizon Droid phones (same operator). As expected, the Neptune DLB system can almost double the throughput in both downloading and uploading. We also observe that the variance of the single link upload speed is smaller than the single link download speed. This can be explained by the fact that there is less upload traffic in comparison to the download traffic. With less competition, the upload speed graph looks smoother than the download speed graph. We note that even when bonding two phones of the same operator, we measure performance improvement. This is due to the fact that a single phone radio is more limited then the radio of the base station and does not reach the base station capacity.

Figure 15 and 16 show the performance of downloading and uploading of the Neptune DLB system with one Verizon Droid and one T-Mobile G1. From our measurements, the average speed of the Verizon network is around 900 kbps for downloading, and 600 kbps for uploading; T-Mobile network is much slower with an average speed of 500 kbps for downloading and 250 kbps for uploading. Therefore, in the cooperative DLB system, the slow network users can benefit even more.

We also experimented the Neptune DLB system with one Verizon Droid and one AT&T iPhone 3G. Similar results can be found in Figure 17 and 18. However, the upload throughput of iPhone 3G displays a periodical pattern. This is because the Neptune *Agent* is not actually running on the iPhone 3G. Instead, as we mentioned it is running on a laptop tethered with iPhone 3G. The OS in the laptop is the desktop Linux. It has a much larger transmission buffer, which causes our transient speed calculation to be inaccurate. Nevertheless, the average speed over a long period is still accurate enough.

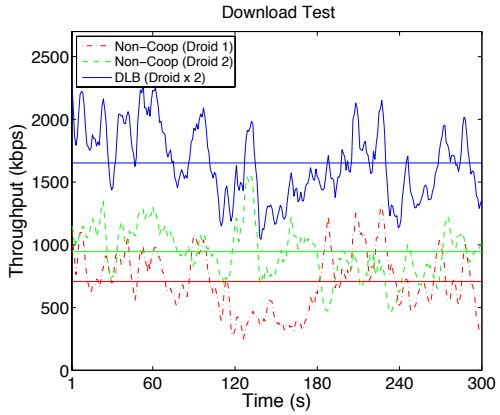

Fig. 13. Downloading throughput of Neptune DLB with two Verizon Droid phones in cooperation.

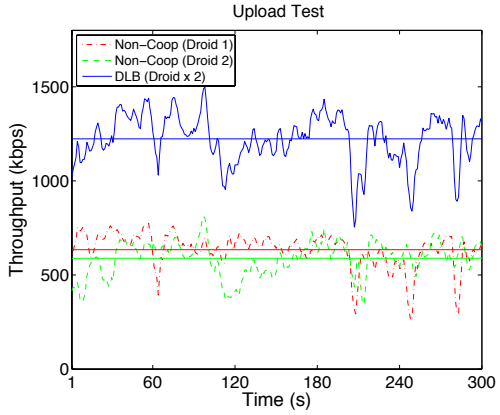

Fig. 14. Uploading throughput of Neptune DLB with two Verizon Droid phones in cooperation.

4 Conclusion

In this paper, we introduce DLB – a distributed cooperation technique for hybrid wireless networks. Our analysis and simulations reveal that it is a simple and effective technique to improve the signal coverage, delay, perceived throughput and network utilization. To

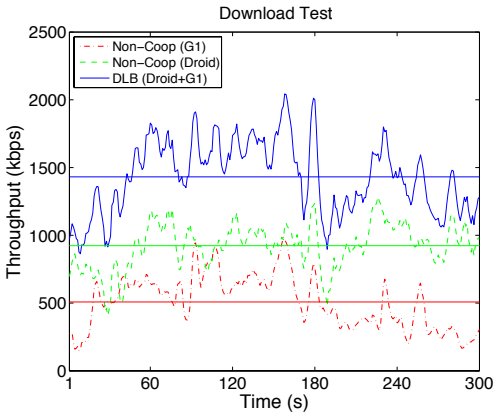

Fig. 15. Downloading throughput of Neptune DLB with one Verizon Droid and one Tmobile G1 in cooperation.

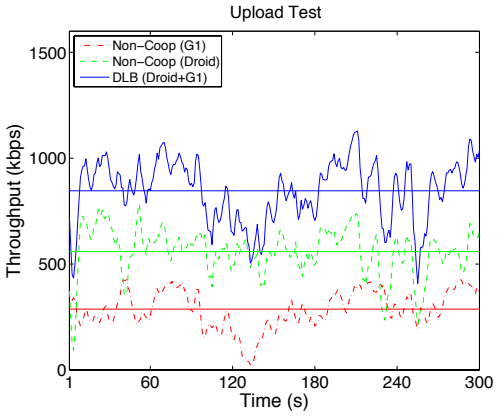

Fig. 16. Uploading throughput of Neptune DLB with one Verizon Droid and one Tmobile G1 in cooperation.

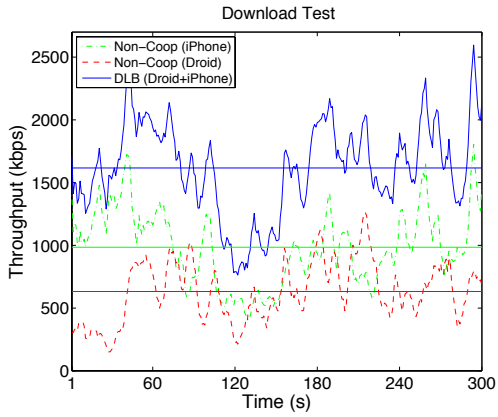

Fig. 17. Downloading throughput of Neptune DLB with one Verizon Droid and one AT&T iPhone 3G in cooperation.

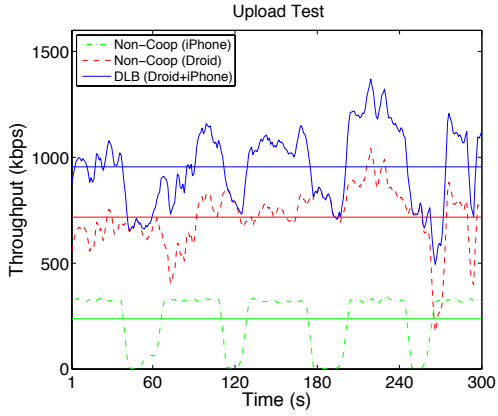

Fig. 18. Uploading throughput of Neptune DLB with one Verizon Droid and one AT&T iPhone 3G in cooperation.

demonstrate that the proposed technique is practically feasible, we propose an architecture, and develop prototypes on for current mobile phones (Android) and the linux operating system. Our experiments show that existing applications can benefit from a significant performance boost without modifications.

REFERENCES

[1] L. Buttyan and J.-P. Hubaux, "Stimulating cooperation in self-organizing mobile ad hoc networks," *Mob. Netw. Appl.*, vol. 8, no. 5, pp. 579–592, 2003.

[2] M. Feldman and J. Chuang, "Overcoming free-riding behavior in peer-to-peer systems," *SIGecom Exch.*, vol. 5, no. 4, pp. 41–50, 2005.

[3] T. S. Rappaport, *Wireless Communications: Principles and Practice 2 edition*. Prentice Hall PTR.

[4] D. P. Bertsekas and R. Gallager, *Data Networks 2 edition*. Prentice Hall PTR.

[5] *IEEE P802.3ad Link Aggregation Task Force*. http://www.ieee802.org/3/ad/index.html.

[6] R. Stewart, Q. Xie, K. Morneault, C. Sharp, H. Schwarzbauer, T. Taylor, I. Rytina, M. Kalla, L. Zhang, and V. Paxson, "Stream control transmission protocol," *IETF RFC 2960*, 2000.

[7] A. Bakre and B. Badrinath, "I-TCP: indirect TCP for mobile hosts," *Distributed Computing Systems, International Conference on*, vol. 0, p. 0136, 1995.

[8] *IEEE802.11s Working Group*. http://grouper.ieee.org/groups/802/11/.

[9] *Motorola's mesh networks*. http://www.meshnetworks.com.

[10] D. S. J. D. Couto, D. Aguayo, J. Bicket, and R. Morris, "A high-throughput path metric for multi-hop wireless routing," *MobiCom '03: Proceedings of the 9th annual international conference on Mobile computing and networking*, 2003.

[11] A. Adya, P. Bahl, J. Padhye, A. Wolman, and L. Zhou, "A multi-radio unification protocol for ieee 802.11 wireless networks," *BROADNETS '04: Proceedings of the First International Conference on Broadband Networks*, 2004.

[12] H. Luo, R. Ramjee, P. Sinha, L. E. Li, and S. Lu, "Ucan: a unified cellular and ad-hoc network architecture," in *MobiCom '03: Proceedings of the 9th annual international conference on Mobile computing and networking*. New York, NY, USA: ACM, 2003, pp. 353–367.

[13] R. Bhatia, L. E. Li, H. Luo, and R. Ramjee, "Icam: Integrated cellular and ad hoc multicast," *IEEE Transactions on Mobile Computing*, vol. 5, no. 8, pp. 1004–1015, 2006.

[14] L. K. Law, S. V. Krishnamurthy, and M. Faloutsos, "Capacity of hybrid cellular-ad hoc data networks," in *Proceedings of IEEE Infocom*, 2010.

[15] H. Javaheri, G. Noubir, and Y. Wang, "Cross-layer distributed diversity for heterogeneous wireless," *Wired/Wireless Internet Communications (WWIC)*, pp. 259–270, 2007.

[16] ——, "Distributed cooperation and diversity for hybrid wireless networks," *Wired/Wireless Internet Communications (WWIC)*, pp. 27–39, 2010.

[17] V. Paxson and S. Floyd, "Wide area traffic: the failure of poisson modeling," *IEEE/ACM Transactions on Networking (TON)*, vol. 3, no. 3, pp. 226–244, 1995.

[18] M. E. Crovella and A. Bestavros, "Self-similarity in world wide web traffic: evidence and possible causes," *SIGMETRICS '96:*

Proceedings of the 1996 ACM SIGMETRICS international conference on Measurement and modeling of computer systems, pp. 160–169, 1996.
[19] F. Hernández-Campos, J. S. Marron, G. Samorodnitsky, and F. D. Smith, "Variable heavy tails in internet traffic," *Performance Evaluation*, vol. 58, no. 2+3, pp. 261–261, 2004.
[20] T. D. Neame, M. Zukerman, and R. Addie, "A practical approach for multimedia traffic modeling," in *Broadband Communications*, 1999, pp. 73–82.
[21] M. Zukerman, T. Neame, and R. Addie, "Internet traffic modeling and future technology implications," 2003.
[22] B. Awerbuch and T. F. Leighton, "A simple local-control approximation algorithm for multicommodity flow," *Foundations of Computer Science, Annual IEEE Symposium on*, vol. 0, pp. 459–468, 1993.
[23] L. Tassiulas, "Adaptive back-pressure congestion control based on local information," *IEEE Transactions on Automatic Control*, vol. 40, no. 2, pp. 236–250, Feburary 1995.
[24] *The Click Modular Router*. http://www.read.cs.ucla.edu/click/.
[25] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek, "The click modular router." *ACM Transactions on Computer Systems*, vol. 18, no. 3, 2000.
[26] *Google Android Platform*. http://code.google.com/android/.
[27] E. Blanton and M. Allman, "On making TCP more robust to packet reordering," *SIGCOMM Comput. Commun. Rev.*, vol. 32, no. 1, 2002.
[28] R. Ludwig and R. H. Katz, "The Eifel algorithm: making TCP robust against spurious retransmissions," *SIGCOMM Comput. Commun. Rev.*, vol. 30, no. 1, 2000.
[29] S. Floyd, J. Mahdavi, M. Mathis, and M. Podolsky, "An extension to the selective acknowledgement (SACK) option for TCP," *RFC 2883*, 2000.
[30] M. Zhang, B. Karp, S. Floyd, and L. Peterson, "RR-TCP: A reordering-robust TCP with DSACK," *Network Protocols, IEEE International Conference on*, 2003.
[31] F. Wang and Y. Zhang, "Improving TCP performance over mobile ad-hoc networks with out-of-order detection and response," *MobiHoc*, 2002.
[32] S. Bohacek, J. Hespanha, L. Junsoo, C.Lim, and K. Obraczka, "A new TCP for persistent packet reordering," *Networking, IEEE/ACM Transactions on*, vol. 14, 2006.

Hooman Javaheri Hooman Javaheri is currently a Ph.D student in the College of Computer and Information Science at Northeastern University. He received a B.S. degree in Electrical Engineering from Sharif University of Technology, Iran, and an M.S. degree in Computer Science from Northeastern University. His research interests include wireless communication networks, RF interaction with biological systems, and biologically enabled computation and communication.

Guevara Noubir Guevara Noubir's research covers both theoretical and practical aspects of secure and robust wireless communication systems. He holds a PhD in computer science from the Swiss Federal Institute of Technology in Lausanne (EPFL, 1996). He joined Northeastern University in 2001 and is now an associate professor of computer science. He was a senior research scientist at CSEM SA (Switzerland) between 1997 and 2000 where he led several research projects and contributed to the definition of the third generation Universal Mobile Telecommunication System (UMTS). He is a recipient of the NSF CAREER Award. Dr. Noubir held visiting positions at Eurecom, MIT, and UNL.

Yin Wang Yin Wang is currently a Ph.D student in the College of Computer and Information Science at Northeastern University. Previously, he received a B.S. degree in Computer Science from Nanjing University, China and an M.S. degree in Computer Science from Northeastern University. His research interests include Mobile Ad hoc Networks, Cellular Networks, Mobile Devices and Embedded Systems.