

PeerBooster: Enhancing Throughput in Wi-Fi Networks Through Network Virtualization

Oran Barak Roy Friedman
Computer Science Department
Technion - Israel Institute of Technology
{roy@cs, shadow@t2}.technion.ac.il

Gabriel Kliot¹
Microsoft Research
Redmond, WA, USA
Gabriel.Kliot@microsoft.com

ABSTRACT

This paper reports on a novel mechanism for boosting the throughput of local Wi-Fi networks. The mechanism maintains application transparency by virtualizing the wireless card, while using commodity hardware and an off-the-shelf unmodified driver. Our work is motivated by observed significant differences in performance between Wi-Fi infrastructure vs. Wi-Fi ad-hoc modes. The improvement is obtained through two complementing schemes. The first allows to dynamically reconfigure an existing ad-hoc network into an infrastructure wireless network, while the second allows simultaneous coexistence of infrastructure and ad-hoc wireless networks on the same logical interface over the same wireless channel. The combination of the two schemes increases the total network throughput by as much as 40% in various scenarios and communication topologies compared to the original ad-hoc or infrastructure networks.

We have implemented both schemes in Linux with the MadWifi driver for Atheros chipsets. The source code is publicly available.

1. INTRODUCTION

The proliferation of smart-phones and laptops has created a need for information sharing capabilities between nearby nodes. For example, children and students in schools and universities may wish to exchange large numbers of photos and even video clips between their devices. Likewise, academic and business colleagues may wish to exchange working documents and presentations at meetings. For example, we have witnessed this through the strong demand for the WiPeer [1] software, developed by our group, as well as similar products. Given the availability of wireless radios on mobile devices, and the high-bandwidth of these technologies, the most convenient way for sharing information is through direct wireless communication between the devices, also known as (local/single-hop) ad-hoc networking.

The 802.11 standard [18], widely known as Wi-Fi or Wireless LAN, includes two modes of operation (Figure 1). The first is called *Basic Service Set* (BSS), also known as infrastructure or access-point/station (AP/STA) mode. The second is called *Independent Basic Service Set* (IBSS) and is also known as Ad-Hoc mode. The difference between these two modes stems from their intended use: In the BSS mode,

one node acts as an access point and all other nodes are stations. The access point typically serves as the gateway to the Internet for all stations. All the communication in this mode must go (at the MAC level) through the access point including communication between two adjacent stations.¹ In contrast, in Ad-Hoc mode, nodes communicate directly, and all nodes have equal status w.r.t. the MAC protocol. Notice that many Wi-Fi network interface cards (NICs) can be configured to work as an access point, thereby allowing a mobile node (laptop, smartphone, PDA, etc.) to act as an 802.11 access point. This is regardless of whether at the IP layer this node acts as a real gateway to the Internet, or all the communication is local and performed only between nodes that are within transmission range of the access point².

Performance gaps: We have conducted extensive performance measurements for the obtainable throughput for both modes of operation using two leading Wi-Fi chipsets, namely Intel Wireless chip 2200/2915 [2] and Atheros AR5001X+ chipset. Our findings, reported in Section 3 below, indicate significant performance gaps between the two modes, depending on the network topology and communication scenario (similar phenomenon was reported in [14], yet without an explanation). On the one hand, in a two nodes network, BSS mode can achieve a 40% higher throughput compared to Ad-Hoc mode. While we initially thought that this gap can be attributed to the use of different MAC access procedures between the two modes (PCF vs. DCF, described in Section 2.1), we have quickly discovered that this is not the case. The real reasons stem from different vendor implementations and different optimization schemes employed in the two modes. We thoroughly investigated the reasons for this gap and they are described in Section 3.4.

On the other hand, when the network is comprised of more than two nodes, Ad-Hoc mode may become more efficient. This is because in a three nodes network operating in BSS mode, all communication between two stations must pass through an access point (at the MAC level). As a result, the throughput of this link in BSS mode is practically halved. In the same scenario Ad-Hoc mode maintains direct MAC links between all nodes and thus achieves higher throughput.

Our solution: These findings have motivated us to look for a way to combine the benefits of the two approaches, while

¹The work was partially done when the author was with the Computer Science Department in the Technion.

¹We discuss the new 802.11e DLS (Direct Link Setup) extension in Sec. 7.

²In this paper, we do not consider multiple hop ad hoc networks.

only relying on commercial off-the-shelf hardware and driver and without modifying any driver code. All our schemes are implemented in the user level mode, and utilize standard Linux configuration tools. This approach can encourage users to leverage our schemes in operational systems while taking minimal risks.

Specifically, our first scheme is based on a distributed leader election protocol, executed when the network is in Ad-Hoc mode. Following the run of this protocol, all nodes switch their Wi-Fi NICs to BSS mode such that the chosen leader becomes the AP and all other nodes act as stations. By utilizing Linux virtual interfaces, this mechanism occurs completely transparently to the application. In particular, all opened sockets and TCP connections remain intact and the only side effect is a short added message transmission delay during the switch. As we show in extensive measurements, the throughput between the chosen access point and every station increases by as much as 40% vs. the throughput between these two nodes in the Ad-Hoc mode. This brings the throughput to the levels of normal BSS connections obtained with the same hardware, OS, and drivers.

Yet, once the switch occurs, the throughput for direct communication between stations drops by about 30%, due to the extra hop at the MAC level. Consider an example in Figure 1, which describes a network with three nodes, an access point and two stations. Messages between the stations must be transmitted at the MAC level first from the source station to the AP, and then from the AP to the destination station. Hence, our first scheme by itself is only useful in the two nodes network, or more generally, when the communication pattern mostly follows a star-like topology, e.g., when one node holds a very popular video clip or is sharing its presentation file with all attendees of a conference talk.

To extend the benefits of our scheme to general communication patterns, we complement the above with another scheme: we enable each node to be simultaneously a station or an access point in BSS mode and an Ad-Hoc node in Ad-Hoc mode. Consider the network in Figure 2. Node 1 acts both as an AP in BSS mode and as an Ad-Hoc node, while nodes 2 and 3 act both as stations in BSS mode and stations in Ad-Hoc mode. As a result, node 2 is able to communicate as a 802.11 station to the access point (node 1) when node 2 wishes to exchange messages with node 1 and concurrently communicate in 802.11 Ad-Hoc mode with all other stations in the local network (node 3).

All this is done by utilizing several standard Linux tools (Linux bridge and ebtables) to create a complete virtualization of the networking interfaces. Consequently, as before, the application as well as the kernel networking stack from the IP level and up are completely oblivious to our manipulations. To the best of our knowledge, we are the first to describe a MAC level virtualization (as opposite to IP level) for wireless cards. Moreover, all our code is executed in user space, and we do not change any line of kernel or driver code. This is commonly perceived as a significant advantage, as users are usually reluctant to recompile the kernel or add dynamic modules; user space implementations are also more robust to bugs. The performance measurements we have conducted indicate a very marginal drop in throughput

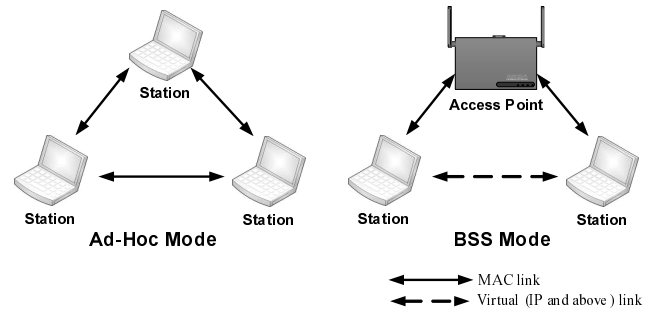


Figure 1: 802.11 protocol modes. In Ad-Hoc mode all nodes are connected to each other at MAC level. In BSS mode, although all nodes are connected at the physical (PHY) level, at the MAC level stations are only connected to the AP. At IP level and above all stations are virtually connected between themselves.

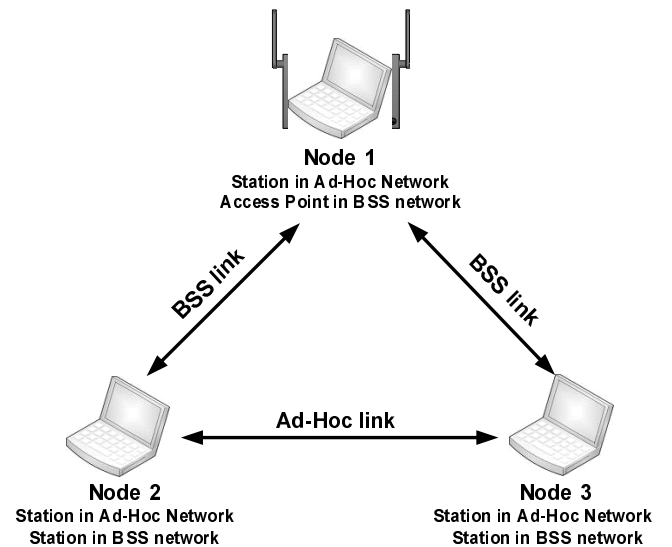


Figure 2: PeerBoost modes superposition.

compared to bare communication in 802.11 Ad-Hoc mode. Hence, the overall result of the combined scheme is that we dramatically improve the communication performance between every node and the chosen AP, while keeping the obtainable throughput between all other nodes almost the same.

Our implementation relies on the open source MadWifi driver [3] for Atheros wireless chipsets. This driver has two important features that made this project feasible. First is a Software Access Point, which allows to create access point with regular wireless card. The second is an ability to put a single network interface in more than one networking mode over the same wireless channel: the same node can be simultaneously an access point in one wireless network and an Ad-Hoc node in another network; or an access point in one network and a station in another network.

Prior work: Prior works [14, 16] have shown how to use a single wireless card to connect to multiple networks. These works differ from ours both technically and by their intended usage. Both [14, 16] allow a node to connect to different nodes in different networks over each of their wireless interfaces (usually, connect to multiple APs or to bridge

between different nodes in BSS and Ad-Hoc networks). On the other hand, PeerBoost’s goal is to connect the same set of nodes through multiple networks (create multiple wireless links between the same set of nodes over different virtual interfaces) with the goal of throughput enhancement.

In addition, [14, 16] use different channels to connect to multiple networks. Using multiple channels over the same card can increase the effective total network capacity beyond the single channel network, e.g., as done in [13], and thus has a great potential. However, there are also some significant limitations to this approach. First, usage of multiple channels cannot increase the capacity of small (two or three nodes) networks, since one needs at least four nodes to utilize two channels. More importantly, multiple channels do not come for free. Spectrum is a valuable limited network resource and there is a limited number of channels to use, some might not be free in a specific setup. Specific channels might even be restricted by network operators. In addition, utilizing multiple channels usually incurs switching and buffering overhead. In such settings, PeerBoost provides a significant throughput improvement while using a single channel, thus eliminating the switching overhead and not wasting additional channels. In fact, our approach can be seen complimentary to the usage of multiple channels. As there is only (at most) 11 channels (and not all of them can be used in practice), when there is a large number of nodes it is no longer possible to allocate a different channel for each pair of nodes. In such a setting our scheme can improve the throughput for each subset of nodes allocated to the same channel. More details comparing these works with ours can be found in Section 7.

Our contributions: In this paper we make the following contributions:

- We explore the throughput differences between Ad-Hoc and BSS modes and provide a detailed analysis.
- We present a system that increases the throughput of Ad-Hoc wireless links by up to 29% by turning these links into BSS (access-point/station) links.
- We present a system that increases the total network throughput in the presence of station-to-station wireless links in BSS mode by up to 37%, by turning these links into Ad-Hoc links.
- We describe a unified system that interposes the above two approaches and provides an always-superior throughput in all modes.
- We describe a virtualization technique based on standard Linux tools that allows to virtualize multiple network interfaces under the same MAC interface. The presented technique is not unique to Wi-Fi and can be used to virtualize any mixture of network interfaces.
- Our code is publicly available at [4] and can be used on any Linux distribution with MadWifi driver³.

³The MadWifi driver and the tools we have used are supported in all major Linux distributions. However, we’ve tested our system only on Ubuntu 8.10 distribution.

Paper road-map: We provide a brief summary of the relevant IEEE 802.11 MAC protocol details in Section 2. Section 3 describes the empirical study of the throughput characteristics of local Wi-Fi networks, in different scenarios and modes. This study motivates our proposed solutions. We describe our first scheme, which improves the network throughput in star communication patterns in Section 4. We describe our second scheme, which improves the network throughput in other scenarios in Section 5. In Section 6 we describe the throughput results obtained with our enhancements. We discuss relevant related work in Section 7 and conclude in Section 8.

2. BRIEF BACKGROUND

2.1 802.11 PCF and DCF Procedures

In order to coordinate access to the wireless medium, the 802.11 MAC protocol [18] employs one of the two basic procedures:

PCF: Point Coordination Function. This optional procedure relies on the presence of a single coordinator (access point). With PCF, the MAC procedure is centralized in the sense that an AP grants permissions to stations to transmit the data.

DCF: Distributed Coordination Function. In this fully distributed procedure all stations (and an access point, if present) compete for the wireless medium in a similar manner to standard CSMA protocols.

On the one hand, DCF is more robust and suits more general networks and setups. In particular, DCF can be used in both Ad-Hoc and BSS modes, while PCF can only be used in BSS. On the other hand, in BSS mode, PCF can achieve higher throughput since the central coordinator can reduce the amount of unsynchronized competition between the stations and subsequently reduce interference.

Unfortunately, PCF is rarely implemented in practice. Based on our informal investigation [5] we speculate that the reason is that PCF does not yield a significant throughput improvement in realistic setups. On the one hand, in large networks with hidden nodes the performance of PCF practically deteriorates to the performance of DCF. On the other hand, in small home networks there is hardly any contention and DCF performs well enough. Regardless of the reason, PCF is not implemented in the chipsets we were using. We have verified this by sniffing the beacon and association messages emitted by AP and stations, and checking that they do not support PCF, as detailed in their published capabilities list. We are unaware of any vendor that implements PCF.

2.2 Ad-Hoc vs. BSS Throughput in the DCF Procedure

In theory, a point to point connection (link) using the DCF procedure should have the same throughput in both Ad-Hoc and BSS modes. This is since there are no differences in these operation modes at the physical level and in the basic MAC procedure (the difference between the two modes is mainly in the way they are managed). In this paper we

| | Wireless Card | Chipset | Driver | OS |
|------------------------|--|------------------------------|-----------------------------------|-------------------|
| PC 1 (node1) | DLink DWL-G550 (HW version A1) PCI Wi-Fi card | Atheros AR5001X+ chip Rev 01 | MadWifi version v0.9.4, SVN trunk | Ubuntu Linux 8.10 |
| PC 2 (node 2) | DLink DWL-G550 (HW version A1) PCI Wi-Fi card | | version 3334 ⁴ | |
| Laptop (node 3) | DLink DWLG650 (HW version C4) Wireless CardBus Adapter | | | |
| | Intel Wireless chip 2200/2915 (HW Version 1.0.3) | Intel PRO/Wireless 2200BG | ipw2200 | |

Figure 3: Hardware equipment.

show that this is not the case in practice; for the NICs that we have tested, as reported below, BSS mode delivers much higher throughput. To the best of our knowledge (and as confirmed by our study in Section 3), this stems from the fact that vendors tend to implement more optimizations for the BSS mode than for the Ad-Hoc mode, since the latter is not considered “useful” enough to merit the effort [5].

3. WIRELESS NETWORKS THROUGHPUT CHARACTERISTICS

3.1 Setup

In this section we report on an extensive study of the throughput between Wi-Fi nodes. All measurements were taken in a three nodes network, in various scenarios. The nodes were placed about half a meter from each other, in the same room, without any obstacles between them. Thus, all nodes are in the transmission range of each other. To avoid interference with unrelated nodes we used a radio channel that had no other Wi-Fi nodes on at the moment the experiment was started. In addition, most measurements were taken late at night and during weekends to further minimize possible interference. Still, since the experiments were not performed in an RF isolated chamber, some interference from stray laptops could have existed.

3.2 Equipment and Tools

The equipment we have used is described in Figure 3. We have three nodes equipped with DLink and Intel cards. Node 3 has both DLink and a built-in Intel Wireless chip 2200/2915 [2]. Unless explicitly mentioned otherwise, we turn this chip off to avoid interference with the DLink adapter.

We have used Iperf [6] as a throughput measurement tool⁵. In all our experiments, we configure Iperf to send (TCP or UDP) packets on a specified link at a rate that exceeds the link bandwidth (54Mb/s). The actual rate reported by Iperf is therefore the actual maximal link throughput. We measure the throughput of different links in various scenarios (a single link with homogeneous and heterogeneous wireless cards, two and three simultaneous links). The latency differences are reported in Section 6.4.

⁴We did not explore the most recent ath5k and ath9k drivers, which were still under development at the time we started this study.

⁵We have discovered and fixed a number of bugs in Iperf version 2.0.4. The fixes were contributed back to the community.

We run every measurement for 60 seconds for 5 times and plot the average as well as the minimum and maximum link throughput in Mb/s. For full details of the driver configuration and more detailed measurements results refer to [7].

3.3 Driver Optimizations

The MadWifi driver implements several optional optimizations [8, 9], listed below.

Bursting: Frame Bursting is a transmission technique supported by 802.11e QoS specification [20]. Frame Bursting increases the throughput of any point-to-point link by reducing the transmission associated overhead. In this mode, the source and the destination capture a channel in turns for transmissions. After a first frame is transmitted and the acknowledgment is received, the sender does not wait for the required time interval. Instead, it waits only a very short fixed time period (without backoff) and then transmits the second data frame, etc. Thus, the sender does not give an opportunity for other stations to start their transmissions and they have to wait for the end of its burst. To eliminate starvation, the total burst time is limited. Eliminating the originally required time interval allows larger data chunks to be transmitted over the same period of time, thereby improving the channel throughput. Although all Atheros products support this technique [8], devices from other manufacturers may not support it. Thus, when communicating with a device that fails to acknowledge a burst transmission, the sender falls back to the base mode.

Fast Frames: Fast Frames technique, which is also based on the 802.11e standard [20], bundles two frames into a single larger frame. This eliminates extra overheads of the header of the second packet and inter-frame spaces.

Rather than restricting frames to the standard size, the frame size is negotiated between a transmitter and receiver. The maximum allowed size is 3000 bytes, which is twice as large as the maximum frame size of a standard Ethernet packet. Since in BSS mode all traffic goes via an access point, the frame negotiation protocol requires an access point that supports fast frames. While bursting increases the number of frames transmitted in a given transmission opportunity, fast frames allows for more information per frame to be transmitted. As with bursting, fast frame mode may not be supported by all manufacturers.

No-Ack policy: Although the 802.11 standard requires an acknowledgment for every unicast packet, the 802.11e extension allows transmissions without acks. This results in greater theoretical bandwidth since no time is spent on acks. Naturally, this can have severe side effects in a lossy environment as there would be less MAC retransmissions, leading to greater packet loss in UDP and more TCP retransmissions.

Adaptive radio: This technology provides performance-on-demand based on channel utilization. Adaptive Radio monitors the entire 802.11 band and automatically increases the throughput only when channels are available and the application demands additional bandwidth.

Additional optimizations: The driver supports the ability to “background scan” other channels in order to find a channel with fewer nodes. The driver spends about 10% of the

time listening to other channels and while doing so it cannot transmit or receive on the original channel. This can lead to throughput degradation and thus we turn it off.

The driver also supports an adaptive rate algorithm that changes the transmission rates according to channel conditions. Lowering the rate can potentially improve the throughput in lossy environments, as lower rate enables better signal capture. However, our empirical testing showed that adaptive rate usually resulted in lower throughput compared to the throughput achievable with maximal possible rate. Therefore, in all tests we fix the rate at the highest possible 54Mb/s .

The driver also supports a Turbo Mode: two channels are combined for transmission, thus theoretically offering double throughput (108 Mb/s). Turbo modes is a proprietary Atheros feature, which has to be supported by the AP and all stations in the same network, and it is not part of the 802.11e standard. In addition, its use is limited in certain countries [10]. Therefore, we do not use the turbo mode.

3.4 Results

3.4.1 Impact of driver optimizations on throughput

We start by exploring the impact of various driver optimizations by measuring the performance of a single link in a two nodes network. We consider every combination of 2 out of 3 nodes, in every direction (total of 6 possible links) and depict the average throughput as well as the maximal and minimal throughput out of the 6 possible combinations. These measurements enable us to evaluate the best obtainable results as adding more nodes can only degrade performance.

The results for same vendor NICs are depicted in Figures 4(a) and 4(b). We add one optimization at a time. We can see a significant difference between Ad-Hoc and BSS modes. Based on the measured throughput gap we can clearly see that: (i) in Ad-Hoc mode the driver implements only Bursting; (ii) in BSS mode the driver also implements Fast Frame and No-Ack. In all cases (TCP and UDP, with and w/o optimizations), BSS mode provides a significant throughput increase (between 23% and 43%) over Ad-Hoc mode. Some insignificant differences between the links can be attributed to small differences in physical locations (the distance between the nodes) and different hardware and environment changes (background noise). UDP mode is generally better than TCP since in a single link scenario there is no contention and thus the additional TCP overhead (acks, slow start) hurts the throughput.

Additionally, we have manually validated that Fast Frame and No-Ack are indeed not implemented in Ad-Hoc mode, but are implemented in BSS mode. We have sniffed the packets and saw that while in BSS mode nodes send large packets of 3000 bytes. This never happens in Ad-Hoc mode. We have also seen nodes that do not send acks in BSS mode, while this never happened in Ad-Hoc mode.

Yet another reason we have found for the throughput differences is that the contention window has different values in both modes. Specifically, the standard mandates nodes to use post backoff after every successful transmission. The backoff is a random number of slots of 10 microseconds

(802.11g) picked out of the contention window parameter, which is initialized to the pre-configured value CW_{min} . The 802.11e extension allows a node to change its CW_{min} value. We have seen in the MadWifi driver code that BSS mode uses CW_{min} of 7 while Ad-Hoc mode uses CW_{min} of 15. We have also seen in the sniffer that the idle period after a transmission is on average twice larger in Ad-Hoc mode than in BSS mode. The difference in CW_{min} value explains why even when all optimizations are turned off, Ad-Hoc exhibits worse throughput than BSS mode.

In Figures 4(c) and 4(d) we report the results of repeating the same measurements using the Intel Wi-Fi NIC on the laptop (node 3). This way we evaluate the performance using NICs from two different vendors (Intel and DLink). In general we can see the same trend of BSS mode achieving a superior throughput of 23% up to 50%. In addition, the absolute throughput of the BSS link is lower than in the case with homogenous HW. All this is explained by a poor interoperability of the driver optimizations.

In particular, we have found that: (i) there is no Fast Frame and (ii) no No-Ack optimizations between heterogeneous nodes, in both BSS and Ad-Hoc modes. As for the Bursting: (iii) it is implemented in both Atheros and Intel cards in BSS, but only in Atheros cards in Ad-Hoc. Therefore, when an Intel card sends packets to an Atheros card in Ad-Hoc mode we see that no optimization kicks in at all. When an Atheros card sends packets to an Intel card in Ad-Hoc mode Bursting does operate and we get a slight improvement. A personal conversation with Intel Wi-Fi engineers confirmed our findings [5].

After establishing the impact of driver optimizations we fix the used set to include Fast Frames, Bursting, and Adaptive radio (No-Ack is turned off). These are the default MadWifi driver settings. We call it a normal driver mode and run all our subsequent measurement using these settings.

3.4.2 Single link

We now measure a single link throughput again, this time having a third node active but idle. We use 2 DLinks NICs. Our goal is to measure the throughput degradation caused by adding another NIC to the environment. Notice that in this scenario, when running in BSS mode, link 2→3 is a virtual STA/STA link, whose traffic has to be sent over two physical links (2→1 and 1→3). Since we have already established the impact of different driver optimizations, measurements are done in the Normal mode, i.e., the Wi-Fi driver is used “as is” without any special configuration. This is the mode we use for all subsequent measurements.

Figures 5(a) depicts the throughput of the direct links (AP to station and station to AP links in the BSS mode and station to station links in the Ad-Hoc mode). The BSS throughput is better than the Ad-Hoc throughput by 30%, which is quite similar to the throughput of a two nodes network reported in Section 3.4.1. The minor differences can be attributed to the third present node that although is in the idle state still influences the medium (e.g., by an increased amount of management frames, such as association, probes, and PSM control frames).

Figures 5(b) depicts the throughput of the virtual link (sta-

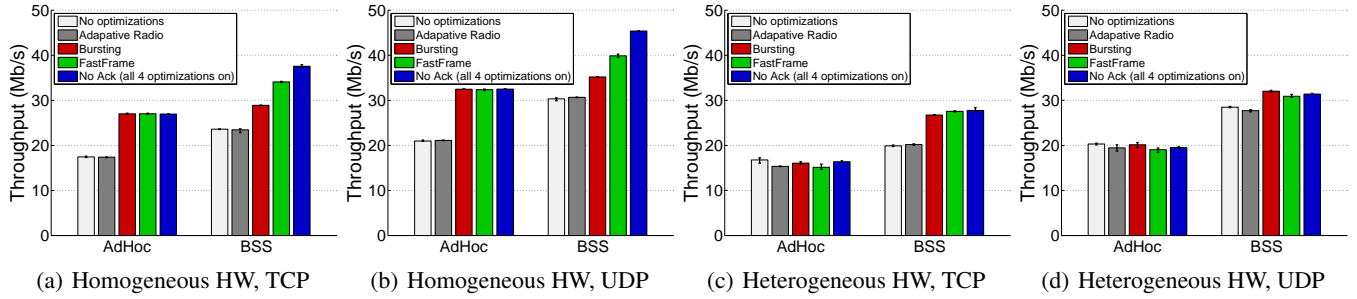


Figure 4: Cumulative optimizations impact. Single link, homogeneous (Dlink) and heterogeneous (Intel and Dlink) HW. For the same HW Ad-Hoc implements only Bursting, while BSS implements also Fast Frame and No-Ack. Heterogeneous HW has even poorer inter-operability. In total, BSS (AP/STA) links are better than Ad-Hoc links by 23%-43%.

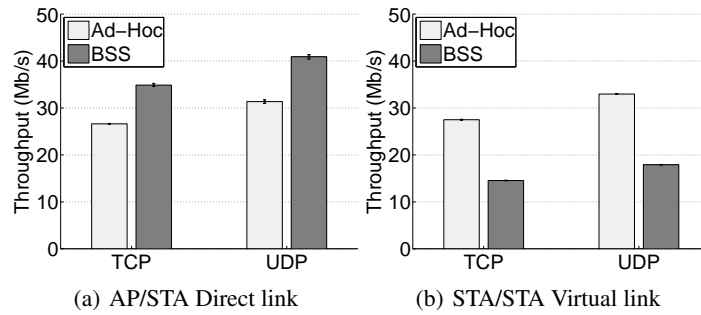


Figure 5: Single link throughput, homogeneous (Dlink) hardware, Normal driver optimization mode, 3 operational nodes. AP/STA links are better than Ad-Hoc links by 30%, virtual STA/STA link is worse than Ad-Hoc link by 47%.

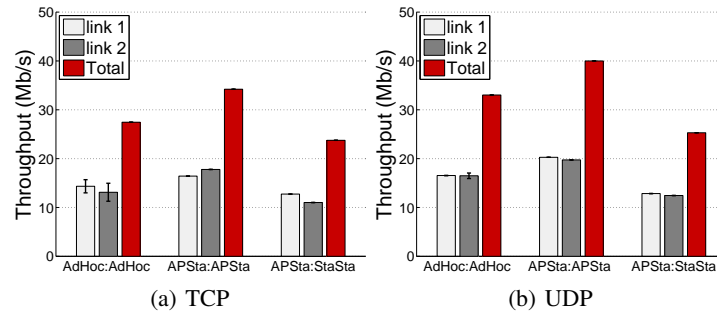


Figure 6: Two simultaneous links. In the scenarios without the virtual STA/STA link BSS outperforms Ad-Hoc by up to 26%. In the scenarios with the virtual link Ad-Hoc outperforms BSS by up to 18%. This is since the virtual STA/STA link in the BSS mode doubles the actual physical transmissions.

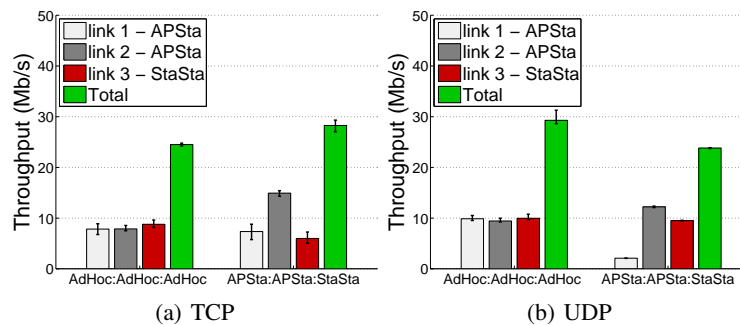


Figure 7: Three simultaneous links. Ad-Hoc achieves balanced fairness between the 3 links, while BSS reveals significant variations, attributed to different buffer management in the AP. The virtual STA/STA link in BSS is worse than Ad-Hoc link in TCP and UDP, while the direct AP/STA links are better than Ad-Hoc links in TCP, but worse in UDP.

tion to station link 2→3) in BSS mode vs. the same link in Ad-Hoc mode. The throughput in Ad-Hoc mode is almost twice than in BSS. Since there is no direct connection at the MAC layer between the station nodes in BSS mode, the traffic must pass through the AP (node 1) in order to get from node 2 to node 3 (2 → 1 → 3). Since the medium (radio channel) is shared between all nodes, the throughput drops in half. In Ad-Hoc mode this limitation does not exist and we have a direct MAC connection between nodes 2 and 3.

3.4.3 Two simultaneous links with 3 nodes

We now use all 3 NICs and test 2 simultaneous streams. That is, we have all 3 nodes operational all the time, pick 2 links out of all possible combinations of 6 links (3 nodes, up and down directions) and run simultaneous Iperf throughput streams on them. The results are depicted in Figure 6.

In the combinations that do not include the virtual STA/STA link (3→2 or 2→3) the BSS modes outperforms the Ad-Hoc mode approximately by the same amount as before (up to 26%). The total throughput of the 2 links in both modes (34.2 Mb/s in BSS and 26.9 Mb/s in Ad-Hoc) is only slightly lower than the throughput of a single link, due to air interference between the simultaneous links. The amount of interference differs for various combinations. For example, when the 2 streams originate from the same node (e.g., streams 1→2 and 1→3) there is actually no air interference at all, since the streams originate from the same network card that transmits only one packet a time. On the other hand, streams 2→1 and 3→1 do interfere. This explains the larger deviations between the combinations (longer error bars).

In the combinations that do include the virtual STA/STA link, denoted as APSta:StaSta pair in the figure, (i.e., one link is 3→2 or 2→3 and the other link is one of the 1→2, 2→1, 1→3, 3→1), Ad-Hoc mode resulted in a better throughput than BSS by up to 18% (total throughput of 28 vs. 23.7).

The total throughput of the APSta:StaSta pair (23.7 Mb/s) is as expected. The virtual StaSta stream translates into 2 physical streams and thus effectively with another APSta stream we have 3 simultaneous streams that share the same medium. The total throughput of 34.2 Mb/s is thus divided between the 3 physical streams and both APSta and StaSta streams have ~ 11.4 Mb/s, just as can be seen from Fig. 6(a).

3.4.4 Three simultaneous links with 3 nodes

We use all 3 Wi-Fi NICs and measure 3 simultaneous streams. The results are depicted in Figure 7.

Ad-Hoc mode achieves balanced fairness between the 3 links. The total throughput (sum of 3 links) is a bit smaller than in a single link scenario, due to interference. In Ad-Hoc mode UDP achieves better results than TCP since TCP is a lot more sensitive than UDP to packet loss that occurs when 3 simultaneous streams compete for the medium.

BSS mode, however, reveals quite significant throughput variations, in both TCP and UDP. We attribute this to different buffer management in the access point. The drastic difference between TCP and UDP in BSS mode is conjectured to the bad buffer management in UDP: the problematic virtual link creates too much pressure on the wireless medium and this deteriorates the situation even further for

UDP which lacks any congestion control mechanism. On the other hand, TCP is able to alleviate the pressure on the wireless link due to its built-in congestion control.

In total, the STA/STA link in BSS is worse than the Ad-Hoc link, while the AP/STA links are better than Ad-Hoc.

3.5 Final Measurements Conclusions

We can see that BSS mode consistently yields significantly better throughput in point to point scenarios. However, whenever there are multiple competing streams, such as in the (virtual) STA/STA stream in BSS mode, Ad-Hoc mode is generally better.

4. IMPROVING THE THROUGHPUT OF AD-HOC NETWORKS

This section describes a method to automatically transform an Ad-Hoc network into a BSS network. This includes two aspects: (i) a distributed leader-election protocol for selecting the node that will act as the access point, and (ii) a mechanism for switching all nodes to the BSS mode transparently to the networking layer. The leader is picked dynamically based on a policy driven configuration. We currently pick the node with the strongest average signal and the best connectivity to all other nodes (a more detailed discussion on how to pick the “best” leader appears in Section 4.3). The elected leader becomes an AP and all other nodes become stations.

Below, we make the following assumptions: (a) a connected Ad-Hoc network (i.e., all the nodes can hear each other), (b) we assume stability periods during which no single node repeatedly enters and leaves the networks, and (c) the RF channel is neither very noisy nor busy, and in particular the RF conditions do not change very quickly.

4.1 The Leader Election Protocol

The protocol overview: As in most leader election protocols, the basic idea is that every node initially attempts to become the leader. All nodes exchange periodic information through Hello message, which include who they currently support as a leader, the identity of their neighbors, and some “goodness” information about each neighbor. The latter reflects the estimate of the sender as to how “adequate” the corresponding neighbor is to become a leader. In our specific case, the information associated with each neighbor is its average measured RSSI.

Whenever a node p hears about a better candidate q for becoming a leader than the one p was supporting before, then p switches loyalty and starts supporting q . Assuming the “goodness” value is computed in a symmetric way, which is the case with average RSSI, eventually all nodes agree on the same leader. If this unanimously supported leader does not change for a few rounds, which we term a stability period, then the chosen leader can declare itself as the access point by sending a Kick message to all other nodes. In the response, all nodes switch to the AP/STA network in which the leader acts as an AP and all other nodes as stations.

We use the following notations:

(1) *Potential Peer (PP)*: A node, identified by its unique MAC id, whose transmissions (beacons and other packets)

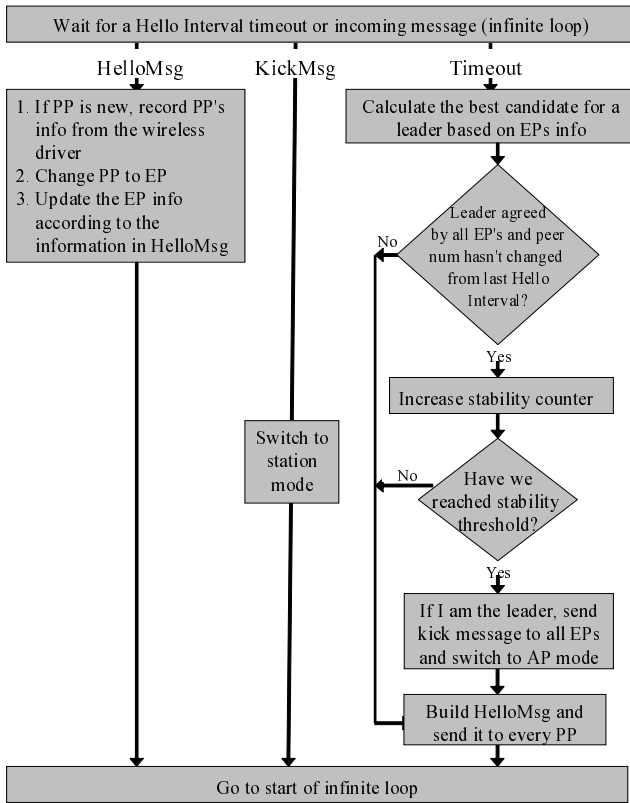


Figure 8: The leader election protocol

can be heard by the local wireless network adapter. Every pre-defined interval the driver is polled for this information.

(2) *Hello Message (HelloMsg)*: A periodic message sent from a node to its PPs. This message contains various data elements, as described below.

(3) *Hello Interval (HI)*: The time interval between two consecutive hello messages.

(4) *Effective Peer (EP)*: A node whose Hello message was received (at least once) during the last 2 HI. In each HI the list of EP's is updated.

(5) *Kick Message (KickMsg)*: A message sent from the selected leader to EPs informing them to switch state from Ad-Hoc to BSS.

Protocol details: The protocol is described in Figure 8 in an event based manner. I.e., the protocol continuously executes an infinite loop in which it waits for one of several events to occur. Whenever such an event occurs, it handles the event, and returns to waiting mode until the next event arrives, and so on. For lack of space, and given that leader election protocols are fairly standard, we only discuss here the unique aspects of our protocol.

Each node records the average RSSI of the messages it receives from every node. Hello messages, which every node transmits to all its PPs, include the recorded RSSI for each neighbor as well as the identity of the temporal candidate leader that the sender supports. Hence, as a result of receiving Hello messages, each node can calculate locally which node p in the system is being heard with the best average RSSI by all other nodes; p is deemed the best candidate for being the leader, as its transmissions are received, on av-

erage, with the best signal. As soon as the identity of the candidate leader reported by all nodes is the same and does not change for a certain threshold number of intervals, it declares itself a leader by sending all others a Kick message and switching to BSS mode as an AP. All other nodes that receive the Kick message, switch to BSS mode as stations.

Implementation details: First, for our mechanism to be independent of the IP layer and transparent to it, all messages are sent directly over the MAC layer via raw sockets. Notice that we get the MAC addresses of all neighbor nodes for free from the driver, as it collects them from all overhead packets. This saves us discovering IP addresses of other nodes.

Second, we send all messages (including hello messages) as unicast messages and not as MAC broadcasts. This is because unicast transmissions are more reliable due to the MAC layer retransmissions, which is not present for broadcasting. In addition, unicast packets can be sent at the maximum established link rate (up to 54 Mb/s) as opposed to broadcast message that are sent in the minimum basic rate (usually 1 or 2 Mb/s in 802.11b and 6Mb/s in 802.11g).

Remarks: Notice that in the protocol, as listed above, if a node p repeatedly fails to receive the Kick message from the leader, e.g., due to interference, it might occur that the leader and some nodes switch to BSS mode, while p remains in Ad-Hoc mode. One possible solution to this problem is to occasionally switch back to Ad-Hoc mode and restart the protocol. Yet, this is an expensive operation. Our complementing mechanism, reported in Section 5, enables nodes to remain in both modes concurrently. Hence, when utilizing the solution of Section 5, it is possible to enable nodes that managed to switch to BSS mode to communicate over that network, while communication between all other nodes takes place over the Ad-Hoc network. In addition, the leader selection protocol can be repeatedly executed over the Ad-Hoc network to enable as many nodes as possible to benefit from this optimization.

Finally, we have not tried to fine tune the various threshold parameters of the protocol. This is left for future work.

4.2 Transparent Interface Replacement

The described algorithm is implemented in C++ as a Linux daemon application (called PeerBoost), running in user space. Here we present an overview of the software system.

When an application opens a network connection, it expects it to stay open. When our daemon switches the state of the Wi-Fi network, it brings down the previous wireless interface (Ad-Hoc interface) and replaces it with a new interface (AP or STA interface in BSS mode). This will cause applications that had open wireless connections to receive error messages from the kernel during the transition informing them that the network is down.

We solve this problem by virtualizing the network interface. We use a bridge module as an abstraction layer between the TCP/IP stack and the MAC layer in the networking stack. Although originally this module is intended for connecting different networks we use it as an abstraction layer⁶. We assume the bridge module is installed at the sys-

⁶We use the bridge even more extensively in the next section.

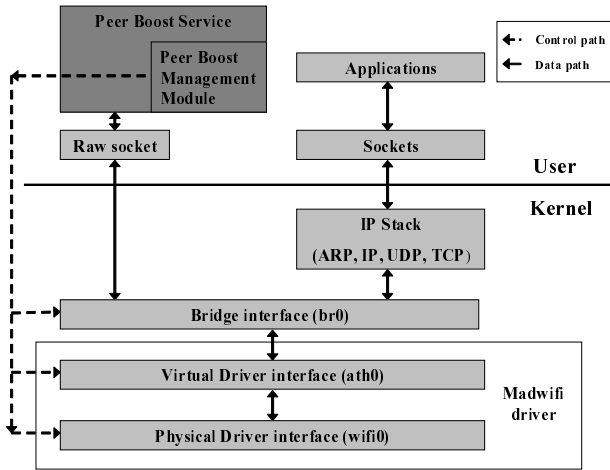


Figure 9: Networking stack with MAC virtualization.

tem prior to PeerBoost execution and the actual physical interface resides underneath it (this requirement can be viewed as part of PeerBoost installation procedure). When applications open network connections, they will do so over the bridge interface. The bridge interface does not change even in face of any changes to the physical interfaces underneath it. We have verified that using the bridge with one physical interface does not impose any throughput degradation (up to a measurement inaccuracy of 0.1%).

To perform the switch, our daemon tears the old Ad-Hoc interface underneath the bridge and brings the new BSS interface (AP or station) up and connects it to the bridge. Since the entire networking stack resides above the bridge interface and the virtual bridge interface preserves the same IP address, the whole switch is transparent from applications and IP stack point of view. In particular, application socket connections will not be broken.

For the short period of time that the interface is down, both incoming and outgoing packets to the layer 2 interface are dropped. This phenomenon is depicted in Figure 16 in the measurement section. This happens since the bridge cannot receive or forward packets to a virtual interface when the virtual interface does not exist during the transition. In addition, the bridge is not able to buffer packets, as it is just a forwarding layer. As a result, TCP only experiences a short link disruption, which it deals with through its normal buffering and retransmission mechanisms. UDP experiences small packet loss, which does not violate its semantics.

Figure 9 presents an overview of the networking stack. The MadWifi driver detects the HW and creates a set of interfaces. There is one physical interface per NIC created with the name `wifi#` (`#` stands for a number, e.g., `wifi0`). Since in our setup we have only one Wi-Fi NIC per machine we see a single interface of this kind⁷. In addition, the driver creates a virtual interface called `ath#` (e.g., `ath0`). All management and configuration commands are performed on the virtual interface. The virtual interface can be configured to be an Ad-Hoc, a Station or an Access Point interface. The driver usually defaults to creating a Station interface, but this can

⁷The protocol can be easily extended to use multiple physical interfaces (work with multiple cards).

be changed via a configuration command. The reason the driver creates two interfaces, a physical and a virtual one, is that multiple virtual interfaces can exist over a single physical one. We discuss this interesting capability in Section 5.

The PeerBoost Management Module controls the creation, destruction, and configuration of the various interfaces. It also interacts with the virtual wireless interface to extract the information about the neighboring nodes (PPs) via a special driver command. These operations are represented by the dotted lines. The PeerBoost Application Data module is responsible for sending protocol messages (these are layer 2 frames, sent directly over the virtual interface). The data frames are represented by solid lines.

4.3 How to Choose a Leader?

There may be various ways to choose the leader. Our protocol as described above works with any deterministic mechanism, in which the nodes converge to a single decision. Trivial such mechanisms can be static selection, MAC id based selection, or a node that provides efficient external Internet connectivity. A more sophisticated adaptive approach may be to dynamically select the leader that will maximize the network throughput, e.g., the one that is most active. Yet, measuring this and continuously replacing the AP imposes a significant overhead that can reduce the average throughput. Consequently, we have not tried it out and base our selection on RSSI values. To minimize possible fluctuations we reconfigure the network with a new leader only if the previous leader has left or if its average RSSI dropped significantly and there is a node with considerably higher RSSI (by a pre-defined percentage). We leave exploration of adaptive leader election policies as a future work.

5. IMPROVING THE THROUGHPUT OF BSS NETWORKS

From the performance study of Section 3, it is clear that switching to BSS mode is most effective when the communication pattern is strongly biased towards one node that is significantly more active than the others. Its best performance is obtained when the communication pattern mimics a star topology. Yet, often the communication pattern is more general, in which case simply switching to BSS may degrade the overall network performance rather than improve it. Below, we present a complementing mechanism that enables some nodes to benefit from switching to BSS, without degrading the performance of the others.

Furthermore, the same mechanism can be used to improve the throughput of BSS networks, by allowing direct connection at the MAC level, which saves an extra hop of relaying the traffic through the access point.

5.1 Multiple Virtual Interfaces Over one Physical

The MadWifi driver has the capability of creating multiple different virtual interfaces over the same physical interface. Originally this functionality was added to the driver to support wireless mesh networks, when the same NIC can function both as an AP for client machines and as a station connected to another AP. The MadWifi driver assigns a unique

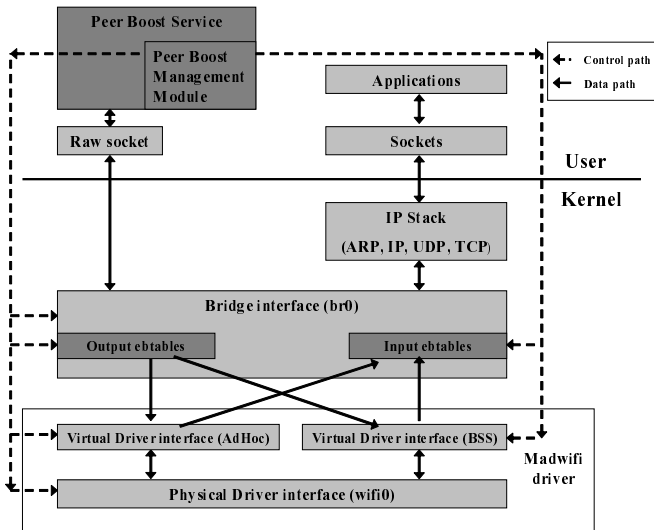


Figure 10: The virtualized networking stack with two MAC interfaces.

MAC address to each virtual interface. The moment the interface is created, it is treated as a fully functional regular network interface, supporting the same functionality of the original physical interface. In PeerBoost we make a different use of this capability. We create two virtual interfaces, one for Ad-Hoc and another for BSS mode, and connect all nodes through both. To the best of our knowledge, we are the first to try to connect all nodes in two complete overlapping network graphs, rather than connecting parts of a larger network, like in mesh.

Figure 10 presents an overview of the new networking stack. We retain the same structure as in Figure 9 for the previous protocol. We build two virtual interfaces to support two networks simultaneously. Since we aim at full application transparency, our solution configures the kernel to forward frames on the appropriate interface according to our specifications (rather than leaving this decision for the application). Every frame is forwarded on the interface that maximizes the network throughput for this transmission. In addition, we make sure our forwarding specifications do not result in forwarding loops.

We unify the two virtual interfaces under a bridge device [11] and configure the bridge to pass frames according to our rules, specified through an ebttables module [12]. This achieves full application and IP stack transparency.

5.2 Protocol Description

We use the leader election protocol as described before with two differences. First, when the kick message arrives to the STA node, instead of turning the old Ad-Hoc interface off and creating a new one, it creates an additional BSS network in which it acts as a station and configures the Linux bridge to work over both networks simultaneously. Second, the leader that initiates the kick also creates an additional BSS network in which it acts as an access point and configures its bridge.

The forwarding configuration rules ensure that every MAC layer frame will be forwarded on the appropriate virtual in-

terface according to its destination. A frame that is AP bound will be sent via the BSS interface. A frame that is destined to another peer will be sent over the Ad-Hoc interface.

Note that although we maintain two separate wireless networks, they operate over the same physical medium on the same channel. Frames on two networks compete over a shared medium, which triggers the usual Wi-Fi networks contention.

5.3 Bridge Rules and Ebttables

Linux bridge is a Linux based implementation of the ANSI-/IEEE 802.1d standard [19]. The bridge transparently relays traffic between multiple network interfaces, based on MAC addresses.

Ebttables is a filtering tool in the Linux kernel [12], focused on the Link Layer Ethernet frame fields (in the same spirit of iptables for the IP layer). Apart from filtering, it also allows to alter the Ethernet MAC addresses and ARP packet fields. Ebttables work with Linux bridge. We use ebttables to route the MAC frames over the correct interface.

We present the bridge rules that maintain the two networks in Figure 11. Each node has just one IP address that is assigned to the bridge interface. The virtual interfaces do not have IP addresses and are therefore inaccessible to the IP stack. We configure the bridge via the ebttables module to switch the source and destinations addresses of frames coming in and out of the bridge. As explained earlier, each virtual interface has a unique MAC address. By changing the addresses in the data frames, we control the network through which they are sent.

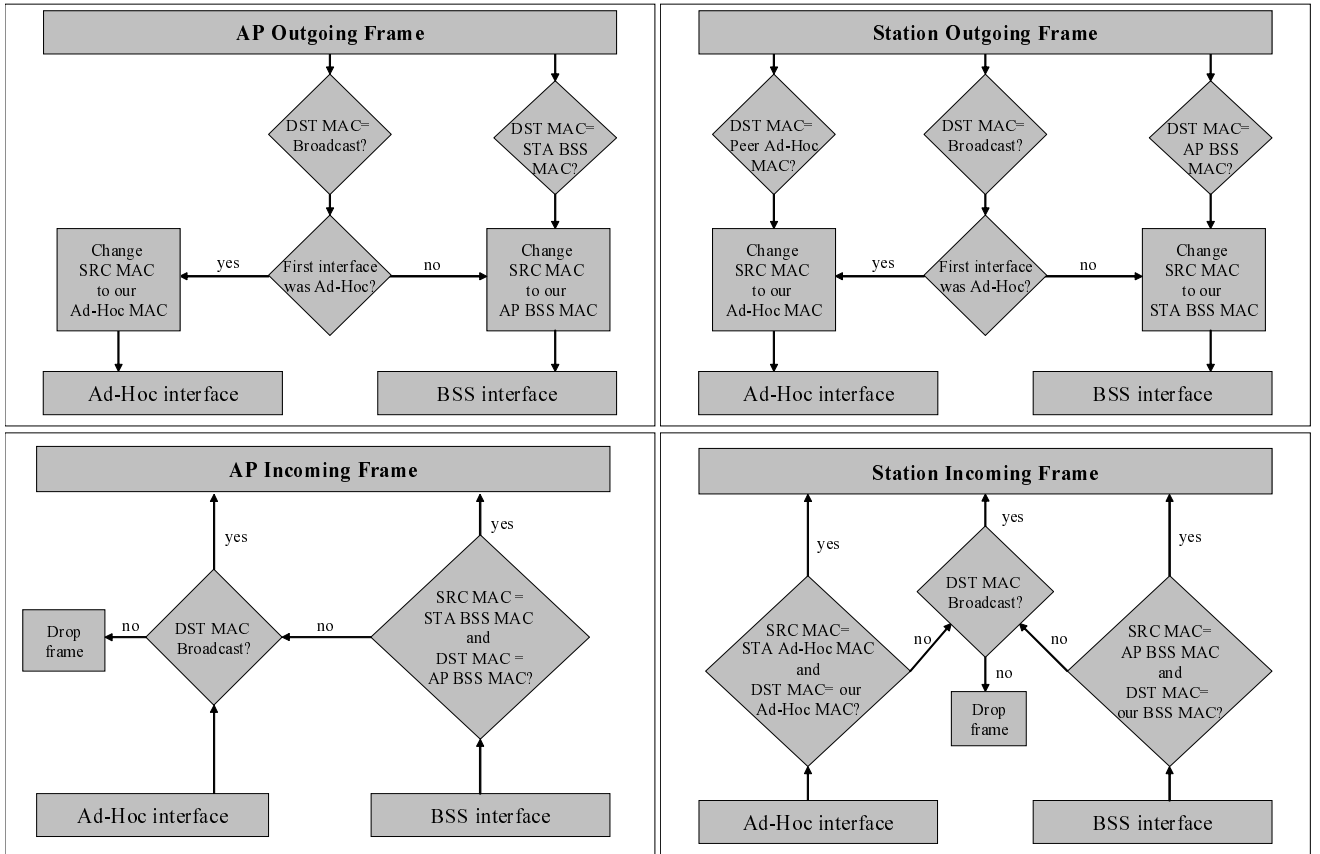
Ebtable Rules: For unicast frames the access point uses only the BSS interface. Other nodes communicate with the access point via the BSS interface and between themselves via the Ad-Hoc interface. In addition, we need to handle two special cases: broadcast and ARP messages. For broadcast messages both AP and the stations will use the original first interface that was operational when PeerBoost started. In fact, we could have just picked one of the interfaces that ensures connectivity with all the nodes and use this one.

ARP frames receive a special treatment: we duplicate the outgoing ARP requests and send them over both interfaces. We also change the addresses inside the ARP request payload, in addition to the regular MAC header changes. The reason we duplicate ARP requests is that we want to allow every node to bind the source MAC address of the ARP request originator to the correct interface. That is, an AP node will bind stations' MAC addresses to its BSS interface, while stations will bind stations' MAC addresses to the Ad-Hoc interface. This is handled inside the receiver ebttables logic.

5.4 Application and IP Layer Transparency

Our solution creates a true abstraction layer that allows all applications to behave as if there is just one standard interface - the bridge, being unaware of the actual two virtual interfaces beneath the bridge. By using a set of pre-configured rules on the bridge we are able to maintain these two coexisting networks. The result is full application transparency.

We maintain one IP address for all virtual interfaces. This IP address is assigned to the bridge and the virtual interfaces



(a) AP input and output ebtbles

(b) Station input and output ebtbles

Figure 11: Bridge ebtbles rules

do not have any IP at all. Our mechanism also does not make any use of the IP addresses of other nodes, as all communication is performed at layer 2 only and ebtbles rules do not use IP addresses as well.

To the best of our knowledge we are the first to build a system that allows superposition of two different physical networks in a way that is transparent both to the applications and the IP stack. That is, we provide a transparent network virtualization at the MAC level.

5.5 Security

PeerBoost does not weaken the security of the overall system. Since we add new network to an already existing network, if the first original network was secured, all participants have already authenticated in order to participate in the network. The second added network can simply continue to use the same keys / encryption methods that were used before the switch in the new network.

6. BRIDGE MODE RESULTS

We now describe the impact of our mechanisms on the performance of the system. We present the results for the bridge mode and compare them to those obtained earlier. We repeat the measurements described earlier (single link, two, and three simultaneous links) and do not repeat the study of the driver optimization. We also investigate the impact on latency.

6.1 Single Link Throughput

Figure 12 depicts the result for a single link measurements (homogeneous (Dlink) hardware, Normal driver optimization mode, 3 operational nodes). A throughput of a direct link (AP/station links) is depicted in Figure 12(a). We can see that the bridge mode achieves almost the same throughput as a pure BSS mode. A small “penalty” of 3.5% throughput degradation is due to the overhead of maintaining two MAC networks. As for the STA/STA link, we can see a clear improvement in Figure 12(b). Previously, this virtual link suffered from low throughput in BSS mode, since it used two physical links. In the bridge mode this link uses an Ad-Hoc network, and as a result its throughput increases significantly (it almost equals the throughput of the same link in pure Ad-Hoc mode; the small differences can be attributed again to the increased management overhead of two networks).

6.2 Two Simultaneous Links Throughput

Two simultaneous flows results are depicted in Figure 13 (for convenience, Figures 13(a) and 13(b) depict the original result identical to Figures 6(a) and 6(b)). We can see that for the APSta:APSta pair of links the bridge mode achieves almost the same results as the original BSS mode. Clear improvements are seen in the scenario that involves traffic on the virtual StaSta link. When examining the APSta:StaSta pair, we can see that the APSta link throughput increases by more than 75% (22.404 Mb/s in bridge mode vs. 12.741

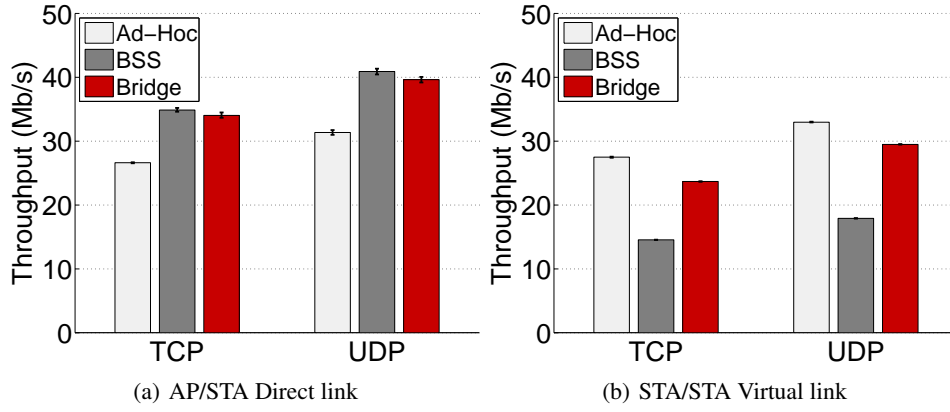


Figure 12: Single link throughput. For AP/STA links the bridge mode has almost the same throughput as BSS mode. For STA/STA links the bridge achieves a better throughput than BSS mode by almost 60% and almost the same as Ad-Hoc mode. On average, the bridge mode is superior to both BSS and Ad-Hoc modes.

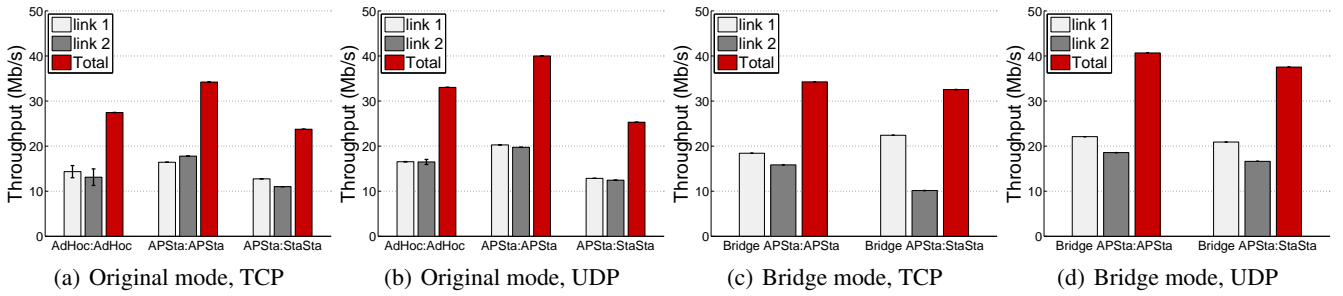


Figure 13: Two simultaneous links. For the APSta:APSta pair bridge mode is very close to the original BSS mode. In the scenario with the virtual StaSta link the total throughput for the 2 links in the bridge mode is 37% higher than BSS and 18% than Ad-Hoc.

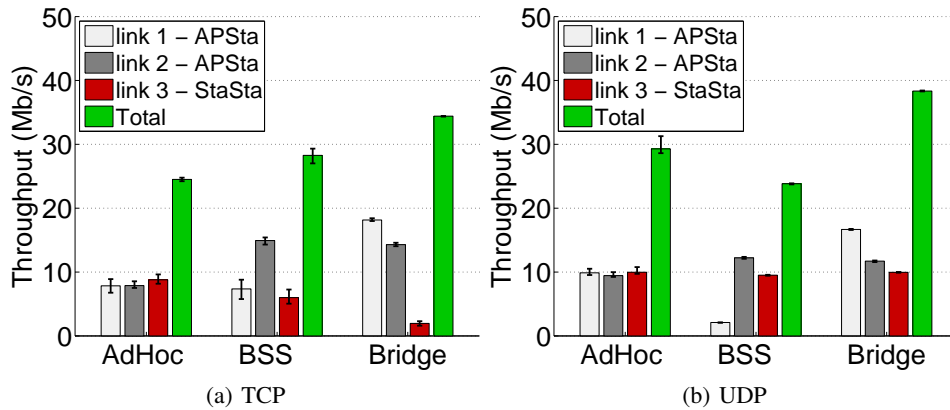


Figure 14: Three simultaneous links. The bridge mode is superior than both BSS and Ad-Hoc modes for total throughput: 34.94 Mb/s in the bridge mode vs. 28.65 Mb/s in BSS (21% increase) and 24.61 Mb/s in Ad-Hoc (41% increase).

Mb/s in BSS in TCP). As for the StaSta link, its throughput is only slightly smaller (10.143 Mb/s vs. 11.009 Mb/s, a degradation of 7.7%), which is due to the asymmetric nature of the links. The total throughput for the 2 links scenario (APSta:StaSta combination) is 32.5470 Mb/s in the bridge mode vs. 23.75 Mb/s in BSS mode (37% increase) and 27.446 Mb/s in Ad-Hoc mode (18% increase).

6.3 Three Simultaneous Links Throughput

Three simultaneous flows results are depicted in Figure 14. The total throughput for three links is 34.9410 Mb/s in the bridge mode vs. 28.6540 Mb/s in BSS (21% increase) and 24.6150 Mb/s in Ad-Hoc (41% increase). Despite the overall throughput increase, we notice an extremely asymmetric nature of some links (18.140 Mb/s vs. 2.2 Mb/s in the bridge mode). We have also seen similar phenomenon in the BSS mode. We attribute this to an inefficient queue management

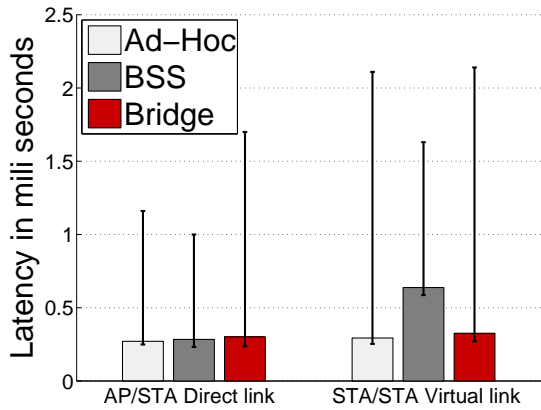


Figure 15: Latency (median and 0.95 percentile). For the direct link all three modes have the same latency. For the virtual link, Bridge and Ad-Hoc have almost the same latency, while BSS is almost double.

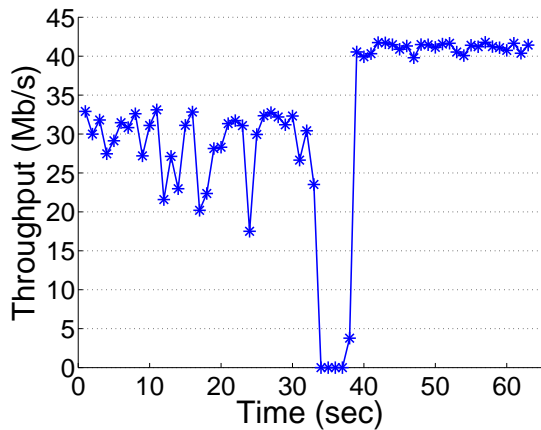


Figure 16: Throughput during reconfiguration from Ad-Hoc to BSS. The throughput drops to zero for 4 seconds and then increases from 28.95 to 41.02 Mb/s (41%).

in the driver which results in severe unfairness. Future research is required to validate this.

6.4 Latency

Figure 15 depicts the median as well as the 0.95 percentile of latency, as measured by executing a ping command every second for the total duration of 5 minutes. We can see that for the direct (AP/STA in BSS or Ad-Hoc link) all three modes have the same latency. For the virtual STA/STA link Bridge has the same latency as Ad-Hoc, while the latency in BSS is almost double. This is of course since in BSS the STA/STA link translates into 2 MAC transmissions.

6.5 Throughput During Reconfiguration

Figure 16 depicts the throughput of a single link in the 2 nodes network during reconfiguration from Ad-Hoc to BSS mode. Before the switch the Ad-Hoc average throughput was 28.95 Mb/s. The switch takes 4 seconds during which the throughput is zero (it also includes ARP as we flush the ARP cache while changing interface). After the switch the throughput increases by 41% to 41.02 Mb/s.

6.6 Results Summary

We have empirically demonstrated the ability of PeerBoost to increase the throughput of small wireless networks by up to 40%. The bridge mode increases the throughput of direct Ad-Hoc links by up to 29% by turning them into BSS (access-point/station) links. It increases the throughput of station-to-station links in BSS mode by up to 37%, by turning them links into Ad-Hoc links. The total throughput of 3 simultaneous streams (all-to-all communication) also increases by 21% vs. BSS and 41% vs. Ad-Hoc (Figure 14). The latency improves in the same manner.

7. RELATED WORK

MultiNet [14]: MultiNet and a subsequent VirtualWifi [17] project allow connecting to multiple wireless networks with a single wireless card. MultiNet exposes multiple virtual adapters, one for each wireless network and uses a network hopping scheme to switch the wireless card across the multiple networks. The switching is application transparent, such that the wireless card appears to be connected simultaneously to all the wireless networks.

There are three major differences between MultiNet and PeerBoost. First, although in MultiNet the wireless card appears to be connected simultaneously to all the wireless networks, this is a virtual connectivity. Physically, at every single point of time the card is connected to a different network on a different channel. Thus, sending traffic over different networks simultaneously is not possible without costly switching and buffering. On the other hands, PeerBoost relies on the built-in ability of the MadWifi driver and the Atheros chipset to be connect to multiple wireless networks simultaneously over the same channel. Thus, no switching and buffering is required.

The second difference relates to application transparency. MultiNet exposes the connectivity to different networks over different virtual adapters, leaving to the application the decision over which adapter (and IP) to send every packet. On opposite, PeerBoost makes the connection to different networks completely transparent to applications by exposing only a single virtual interface with a single IP address for all connected networks.

The third difference relates to the indented usage. MultiNet can be viewed as a low level enabling technology, which allows connecting to multiple wireless networks over a single card. PeerBoost on the other hand is a higher level usage of this technology. Its aim is to increase network throughput. In fact, PeerBoost could be implemented on Windows over VirtualWifi in a very similar way to how we do it on Linux over the MadWifi driver.

FatVAP [16]: The closest to our work is FatVAP, a MadWifi based system that allows to connect to multiple APs simultaneously and aggregate their backhaul bandwidth to maximize throughput. PeerBoost however differs from FatVAP significantly. First, FatVAP connects to different APs on different channels⁸, thus it needs to switch between the

⁸FatVAP is also able to connect to different APs if they are on the same channel, but in practice nearby APs are usually configured on different channels.

networks. The switching in FatVAP is faster than in Multi-Net, but still imposes some non-negligible overhead. PeerBoost connects to different networks on the same channel, eliminating the need to switch. Second, FatVAP performs network virtualization at the IP level, while PeerBoost does it on the MAC layer. This is simpler (e.g., no need to recompute IP and TCP/UDP checksums), allows other systems that manipulate the IP stack (e.g., IPtables) to continue working seamlessly (this could have been different if we have been manipulating the IPtables as well), and allows greater flexibility (the virtualization is performed at a lower level and thus transparent to a larger part of the networking stack). Most importantly, FatVAP's goal is to unify the backhaul bandwidth from different networks, while PeerBoost's goal is to increase the throughput of a single network (basically, make a more efficient use of an air time of a single channel).

Direct Link Setup: Direct Link Setup (DLS) [20] is part of the 802.11e QoS extension that allows direct station-to-station frame transfer within a BSS. The establishment of this direct link is performed via an AP.

PeerBoost provides the same functionality as DLS (connectivity to an AP in BSS mode and simultaneously establishing direct physical links between the stations via Ad-Hoc mode). However, DLS requires a MAC protocol change and a new MAC SW in the stations and in the Access Point. PeerBoost is able to provide the same functionality transparently from an AP, without MAC protocol changes, and with existing HW. Indeed, although already present for a number of years, the DLS implementation is not ubiquitous. We have not been able to identify an access point that supports DLS. Another subtle but critical difference lies in the fact that DLS requires AP authorization for a direct STA/STA communication (some commercial APs might not allow direct communication, e.g., to be able to better control and charge for communication). Our solution is not controlled by the AP.

In addition, our system implements a dynamic reconfiguration of Ad-Hoc network comprised of commodity client Wi-Fi cards into a better throughput oriented BSS network. This is not provided by the DLS protocol.

Performance anomalies: Performance anomalies in 802.11 were also reported in [14, 15]. The work in [14] noticed (without explanation) a significant difference of 32% between the average throughput of a BSS network with commercial AP and an isolated two node Ad-Hoc network. The work in [15] reported a different 802.11b anomaly - one mobile host having a lower bit rate than the others can considerably degrade the throughput of all other hosts transmitting at a higher rate, even below the level of the lower rate. The authors explained this phenomenon by analyzing the CSMA/CA channel access method underlying DCF.

8. CONCLUSIONS AND FUTURE EXTENSIONS

We have presented two complementing mechanisms to boost the performance of wireless Ad-Hoc networks by increasing network throughput. We have implemented both in Linux with the MadWifi driver for the Atheros chipsets and have demonstrated the efficiency of our solution in various

scenarios and communication topologies. The combination of the two schemes increases the total network throughput by as much as 40% in many practical cases.

An important application scenario of our work is improving the throughput of infrastructure networks in big social events. Consider a conference with tens or hundreds of people in one room. The participants use Wi-Fi to surf the web, but some of them also communicate directly between themselves, for example to pass presentations, share files etc. In such a scenario, all the traffic will be directed to one (or more) pre-configured access points. In addition to loading the access point, this also halves the potential throughput, as we explained in this work. Using our software, the participants can establish an Ad-Hoc network between themselves, in addition to participating in the infrastructure network. This way, they can use the Ad-Hoc network for the direct communication between themselves. The result will be reduced load on the access point, preventing inefficient STA/STA links, and subsequently improving both the infrastructure and the Ad-Hoc networks.

There are several additional optimizations that can be added. For example, according to its spec, the driver supports compression, but we did not test this capability thoroughly. In general, in a kernel module we can implement the following optimizations: aggregation, queuing and scheduling, compression (independent from the chip) and selective ack (also suggested by 802.11e but not implemented).

9. REFERENCES

- [1] <http://www.wipeer.com>.
- [2] <http://ipw2200.sourceforge.net>.
- [3] <http://madwifi-project.org>.
- [4] <http://www.cs.technion.ac.il/Labs/dsl/projects/peerboost/>.
- [5] Private communication with engineers in Intel's wireless drivers team, Intel Israel Development Center.
- [6] <http://sourceforge.net/projects/iperf>.
- [7] www.cs.technion.ac.il/Labs/dsl/projects/peerboost/PeerBoost-Appendix-graph.rar.
- [8] <http://www.atheros.com/>.
- [9] <http://madwifi-project.org/wiki/ChipsetFeatures>.
- [10] <http://madwifi-project.org/wiki/UserDocs/TurboMode>.
- [11] <http://www.linuxfoundation.org/en/Net:Bridge>.
- [12] <http://ebtables.sourceforge.net/>.
- [13] P. Bahl, C. Ranveer, and D. John. SSCH: Slotted Seeded Channel Hopping for Capacity Improvement in IEEE 802.11 Ad-Hoc Wireless Networks. In *Proc. of MobiCom*, 2004.
- [14] R. Chandra, Paramvir Bahl, and Predeep Bahl. MultiNet: Connecting to Multiple IEEE 802.11 Networks Using a Single Wireless Card. In *Proc. of IEEE INFOCOM*, March 2004.
- [15] M. Heusse, F. Rousseau, G. Berger-Sabbatel, and A. Duda. Performance Anomaly of 802.11b. In *Proc. of IEEE INFOCOM*, March 2003.
- [16] S. Kandula, K. Ching-Ju Lin, T. Badirkhanli, and D. Katabi. FatVAP: Aggregating AP Backhaul Capacity to Maximize Throughput. In *Proc. of the 5th USENIX NSDI*, April 2008.
- [17] Microsoft. VirtualWiFi: Connecting to multiple IEEE 802.11 networks with one WiFi card. <http://research.microsoft.com/en-us/um/redmond/projects/virtualwifi/>.
- [18] IEEE Computer Society. 802.11: Wireless LAN Media Access Control (MAC) and Physical Layer (PHY) Specifications. <http://standards.ieee.org/getieee802/>.
- [19] IEEE Computer Society. 802.1d: Media Access Control (MAC) Bridges.
- [20] IEEE Computer Society. 802.11e: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications: MAC Enhancements for Quality of Service (QoS), January 2005.