BAPU: Efficient and Practical Bunching of Access Point Uplinks

Tao Jin[1], Triet D. Vo-Huu[2], Erik-Oliver Blass[3], and Guevara Noubir[2]

[1] Qualcomm Corporate Research & Development, San Diego CA, USA,
[2] Northeastern University, Boston MA, USA,
[3] Airbus Group Innovations, 81663 Munich, Germany.

Abstract. Today's throttled uplink of residential broadband renders a broad class of popular applications such as HD video uploading and large file transfer impractical. Aggregation of WiFi APs is one way to bypass this limitation. Motivated by this problem, we present BAPU (**B**unching of **A**ccess **P**oint **U**plinks) to achieve two major goals: (1) support commodity clients by refraining from client modifications, (2) support both UDP and TCP based applications. We justify the need for client transparency and generic transport layer support and present new challenges. In particular, a naive multiplexing of a single TCP session through multiple paths results in a significant performance degradation. We describe BAPU's mechanisms and design. We developed a prototype of BAPU with commodity hardware, and our extensive experiments show that BAPU aggregates up to 95% of the total uplink capacity for UDP and 88% for TCP.

1 Introduction

Today, mobile devices are equipped with high-resolution cameras and are quickly becoming the primary device to generate personal multimedia content. Such fast growth of User Generated Content (UGC) naturally leads to an ever increasing demand of instant sharing of UGC through online services, e.g., YouTube and Dailymotion, or in an end-to-end manner. In addition, there is a trend of instantly backing up personal files in "Cloud Storage" like Dropbox or iCloud. All these services require sufficient uplink bandwidth for fast data transfer. While today's ISPs offer high-speed downlink, uplink bandwidth is usually throttled. As a result, instant sharing of HD content or fast data backup in the "Cloud" is still impractical in today's residential broadband. Consequently, there is a need to scale backhaul uplinks.

1.1 Aggregating AP to Bypass Broadband Limitations

Given that WiFi capacity typically exceeds the broadband uplink capacity by at least one order of magnitude, a single client WiFi can communicate with multiple APs in range and aggregate the idle bandwidth behind them. Several AP aggregation solutions (e.g., FatVAP [17] and THEMIS [12]) have been proposed in the past few years. Their rationale is to route TCP/UDP sessions through different APs such that the traffic load splits across multiple broadband links, thereby achieving a higher aggregated throughput. Yet, a single TCP/UDP connection is assigned to a single AP, in which case the

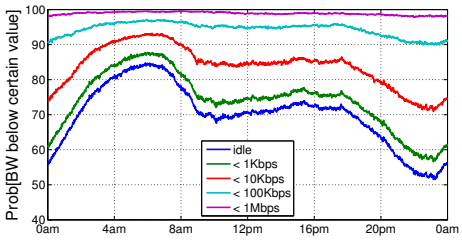


Fig. 1: Residential uplink bandwidth usage.

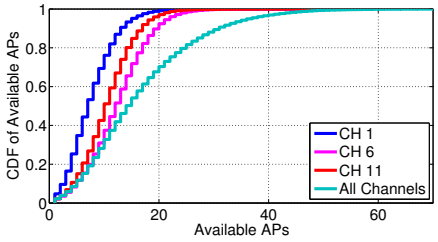


Fig. 2: Available APs in Wardriving.

connection throughput cannot exceed the single broadband link capacity. Since most uplink hogging applications such as iCloud establish single transport layer connections for data transfer, current AP aggregation solutions are not suitable for single session uplink scaling, unless the application is redesigned to adapt to the AP aggregation technology. Recently, Link-Alike [15] multiplexes single UDP flow across multiple APs. However, Link-Alike's design is specific to UDP file transfer, resulting pieces of files to arrive in out-of-order sequence, which prohibits TCP based applications (e.g., HD video streaming) that require a strictly in-order delivery and deadline meeting. Besides, multiplexing single TCP sessions through multiple paths is a challenging problem (and will be discussed later). Moreover, client modifications are required to support TCP. In this work, we require a new AP aggregation solution offering complete transparency on the client with generic support for either TCP or applications.

1.2 Feasibility of AP Aggregation

While WiFi aggregation allows bypassing broadband limitations, it is yet unclear whether aggregation is practical in reality. We now present our recent study on urban WiFi and broadband resources, revealing several interesting insights regarding the feasibility of AP aggregation in residential broadband.

Mostly idle broadband uplinks: Since Feb. 2011, we have developed and deployed a WiFi testbed in Boston's urban area, aiming to monitor the usage pattern of residential broadband. This testbed consists of 30 home WiFi APs running customized OpenWRT firmware with two major broadband ISPs, Comcast and RCN. During a 18 month period, we have collected over 70 million records. Figure 1 shows the uplink bandwidth usage during a 24 hour time window. Throughout the day, there is at least 50% chance that uplink is completely idle. Even during peak hours, there is over 90% chance that the uplink bandwidth usage is below 100 Kbps. Consequently, there exists a considerable amount of idle uplink bandwidth, making AP aggregation a viable approach.

WiFi densely deployed in residential area: Our recently conducted Wardriving measurements in 4 residential areas in Boston identify 22000 APs, 14.2% of which are unencrypted. As shown in Figure 2, there are on average 17 APs available at each location, with an average 7 to 12 APs on each channel. This enormous presence of WiFi justifies the feasibility of AP aggregation in urban area.

WiFi becoming open and social: Driven by the increasing demand of ubiquitous Internet access, there is a new trend that broadband users share their bandwidth as public

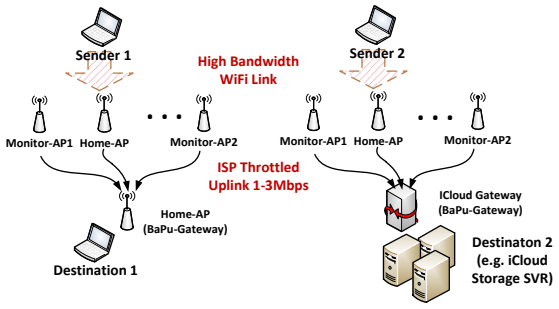


Fig. 3: BAPU system architecture and example application scenarios. Scenario 1 (left): Sender 1 shares an HD video with a remote end user. Scenario 2 (right): Sender 2 backs up a large file to iCloud. The uplink aggregation is enabled via BAPU-enabled Home-AP and Monitor-APs.

WiFi signal to mobile users. Mainstream home WiFi APs, e.g., LinkSys and D-Link, already offer a standard feature which hosts two SSIDs, one encrypted for private use, the other unencrypted for public sharing. FON [9], a leading company in this area, claims to have over 7 million social hotspots worldwide. Given this trend of WiFi becoming social and cloud-powered, a software solution on APs allows much easier progressive adoption of AP aggregation technologies compared to a few years ago.

Based on this discussion, we present BAPU, a complete software solution for WiFi APs allowing broadband uplink aggregation. BAPU features complete transparency to client devices and high aggregated throughput for both TCP and UDP, even in lossy wireless environment. Our major contributions are summarized as follows:

Transparency to client: BAPU does not require any modification to clients. The client device conducts regular 802.11 communications with its home AP while AP aggregation happens in a "transparent" way. Also, all legacy network applications benefit from such transparency and seamlessly utilize BAPU.

Efficient aggregation for both TCP and UDP: Multiplexing a single TCP flow through multiple paths raises many technical challenges, making efficient aggregation non-trivial. We propose a novel mechanism called *Proactive-ACK*. Through an in-depth analysis of TCP stack behavior, we show how *Proactive-ACK* performs efficient TCP multiplexing. BAPU achieves high aggregated throughput for *both* TCP and UDP.

Prototype with commodity hardware: We have prototyped our complete BAPU system on commodity 802.11n WiFi APs with OpenWRT firmware.

Evaluation: We conduct an extensive set of experiments to evaluate BAPU in various realistic network settings. Our results show that BAPU efficiently achieves over 95% and 88% of total uplink bandwidth for UDP and TCP transmissions, respectively.

2 System Overview

For ease of understanding, we first introduce two typical application scenarios that benefit from BAPU – see Figure 3.

Scenario 1. Instant Sharing of HD Video: In order to retain the control of personal content, Sender 1 streams his HD video in real time directly from his hard drive to

Destination 1. Both users are connected to their own Home-APs. Sender 1's uplink is throttled by his ISP to 1 $\sim$ 3Mbps, preventing him to handle the 8Mbps HD video in real time. However with BAPU, the idle uplink of the neighboring Monitor-APs are exploited to boost uplink throughput. BAPU-*Gateway*, the Home-AP of Destination 1, aggregates and forwards multiplexed traffic to Destination 1.

Scenario 2. Instant Backup of Large File: Sender 2 wishes to backup his HD video clip to some cloud storage service such as iCloud. With the 3Mbps uplink rate, it takes over an hour to upload a 30 minute HD video. With BAPU, uploading time is greatly reduced by deloying a BAPU-*Gateway* in front (or part) of the cloud storage servers for handling parallel uploads from Home-AP and neighboring Monitor-APs.

BAPU Protocol Description

In BAPU, *Sender* is associated with its *Home-AP*, and the uploading of data is aggregated via unencrypted wireless link. The data, however, is protected with some end-to-end security mechanism (e.g., SSL/TLS). *Home-AP* and *Monitor-AP* are configured to run in *both* WiFi AP mode and WiFi monitor mode[4]. The WiFi link between the *Sender* and its *Home-AP* generally provides high bandwidth, up to hundreds of Mbps with 802.11n. The link between a BAPU-*AP* and the *Destination*, however, is throttled by the ISP. At the remote end, we place a BAPU-*Gateway* immediately before the *Destination*. The connection between the BAPU-*Gateway* and the *Destination* is either wired or wireless high-speed link. Note that being in proximity, unicasts between *Sender* and *Home-AP* (AP mode) can be overheard by (some of) the *Monitor-AP*s (monitor mode). At a high level, BAPU is a centralized system with the controller residing at BAPU-*Gateway*, providing an uplink aggregation carried out as follows (Figure 4).

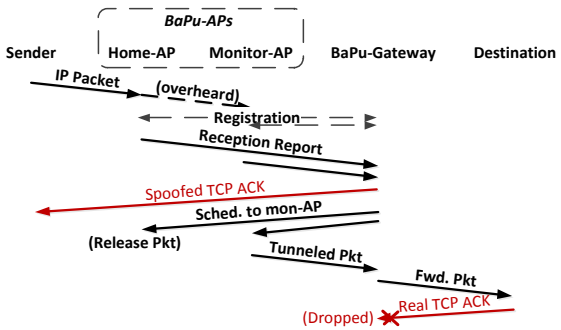


Fig. 4: BAPU Protocol Traffic Flow. The ACKs (red color) are managed for TCP only.

1. *Sender* starts a TCP/UDP upload to *Destination* through its *Home-AP* via WiFi.
2. *Home-AP* and *Monitor-AP* overhear WiFi packets and identify "BAPU" session by checking the destination IP and port.
3. BAPU-*AP*s register themselves to BAPU-*Gateway*.
4. *Home-AP* and *Monitor-AP* capture *Sender*'s packets in monitor mode, and collaborate to upload data for *Sender*, following a schedule determined by BAPU-*Gateway*.

[4] Modern WiFi drivers (e.g., ath9k) support multiple modes for one physical WiFi interface.

5. *Home-AP* and *Monitor-AP* send reports to BAPU-*Gateway* for each packet.
6. In an *UDP* session, BAPU-*Gateway* determines which BAPU-*AP* will forward the captured packet, and broadcast a scheduling message to all BAPU-*AP*s.
7. A *TCP* session is much more challenging to support than UDP. To properly multiplex *Sender*'s single TCP flow through multiple paths, we devise a new mechanism called *Proactive-ACK*: BAPU-*Gateway* sends spoofed TCP ACKs to *Sender* as BAPU session goes on. *Proactive-ACK* is designed to make BAPU work efficiently with legacy TCP congestion control.
8. The scheduled AP forwards packets to *Destination* tunnelled through BAPU-*Gateway*.

3 Uplink Aggregation

In this section, we discuss technical challenges and describe our solutions for BAPU system to achieve an efficient and practical aggregation system. We remark that BAPU shares some similarities in the high-level architecture with previous work (e.g., Link-alike [15], FatVAP [17]). However, from pure practicality aspects, the applicability of those systems is severely limited due to heavy modification of client devices or support for only specific applications (e.g., large file transfer, UDP). Contrary, BAPU targets *transparency* and *high-throughput* transmissions for *both* UDP and TCP applications.

3.1 Network Unicast

First, the transparency goal requires that legacy transport protocols must be usable for data transmission from *Sender* to *Destination*. Thus, *Sender* must be able to transmit data to *Destination* via *network unicast* through its *Home-AP*. Second, the network unicast is more reliable, because the MAC layer handles retransmissions in case of 802.11 frame loss. Consequently, network unicast between *Sender* and *Home-AP* is an essential requirement in BAPU, while prior work [15] chose broadcast for simplicity.

Packet Overhearing: In WiFi networks, network unicast and broadcast differ in the next-hop physical address in the MAC layer. This complicates the packet overhearing capability at *Monitor-AP*s, since the *Sender* uses its *Home-AP*'s physical address as the next-hop address in the 802.11 header, while *Monitor-AP*s automatically discard the packet due to a mismatched physical address. To allow *Monitor-AP*s to capture overheard packets, BAPU's solution is to configure BAPU-*AP*s to operate simultaneously in two modes: *AP mode* and *monitor mode*. The former mode is used for serving clients in the AP's own WLAN, whereas the latter is used for overhearing packets in raw format.

Packet Identification: Each packet sent from the *Sender* (BAPU protocol's step 1) contains the session information in the packet's IP header such as the protocol, the source and destination IP addresses and ports. With this information, *Home-AP* can uniquely identify the *Sender* (step 2). In contrast, *Monitor-AP*s may have ambiguity in identifying the *Sender*, as *Sender*s from different WLANs may (legally) use the same IP address. To resolve such conflict, we create a frame parser for the packet's MAC header to obtain the `BSSID` that identifies the WLAN the session belongs to. Therefore, any session in BAPU is now uniquely determined by the following 6-tuple `<BSSID, proto, srcIP, dstIP, srcPort, dstPort>`.

Duplicate Elimination: Unicasting a packet may involve a number of (MAC-layer) retransmissions due to wireless loss between the *Sender* and its *Home-AP*. This increases the transmission reliability. Nevertheless, it is possible that a nearby *Monitor-AP* can overhear more than one (re)transmission of the same packet and eventually forward unnecessary duplicates to *Destination*, flooding *Monitor-AP*'s uplink. To identify the duplicate packets, we keep records of `IPID` field of each overheard IP packet. Since `IPID` remains the same value for each MAC-layer retransmission, it allows *Monitor-AP*s to identify and discard the same packet. It is worth noting that in TCP transmission, the TCP sequence number (SEQ) is not a good indicator to identify the duplicate packets, as it is unique for TCP retransmission, but not for MAC-layer retransmissions.

3.2 Tunnel Forwarding

The transparency goals requires that the high-level application be unaware of the aggregation protocol in BAPU. A seemingly straightforward solution is that *Home-AP* and *Monitor-AP*s forward the *Sender*'s packets with spoofed IP addresses. It is, however, impractical for two reasons: 1) many ISPs block spoofed IP packets; 2) forwarded packets by *Monitor-AP*s are unreliable, because they are raw packets overheard from the air. Our approach is that each BAPU-*AP* conveys the *Sender*'s data via a separate TCP tunnel. Since we support a transparency for aggregation over multiple paths, the techniques for tunnelling and address resolving in each single path require a careful design at both BAPU-*AP*s and BAPU-*Gateway*.

Tunnel Connection: Once a BAPU-*AP* identifies a new *Sender-Destination* session (step 2) based on the 6-tuple, it establishes a tunnel connection to BAPU-*Gateway*. Regardless of the session protocol, a tunnel connection between the BAPU-*AP* and BAPU-*Gateway* is always a TCP connection. The choice of TCP tunnel is partially motivated by the *TCP-friendliness*. We desire to aggregate the idle bandwidth of BAPU-*AP*s without overloading the ISP networks. Besides, since TCP tunnel can provide a reliable channel, it helps keep a simple logic for handling a reliable aggregated transmission.

Forwarding: In the registration (step 3) to BAPU-*Gateway*, the BAPU-*AP* receives an `APID` as its "contributor" identifier for the new session. The `APID` is used in all messages in the protocol. Both control messages (registration, report, scheduling) and data messages are exchanged via the reliable TCP tunnel. On reception of a scheduling message with matching `APID`, the *Monitor-AP* encapsulates the corresponding *Sender*'s packet in a BAPU data message and sends it to BAPU-*Gateway* (step 8), which then extracts the original data packet, delivers to the *Destination*. In BAPU, short control messages only introduce small overhead in the backhaul.

NAT: In WLAN, the *Sender* is behind the *Home-AP*, typically a NAT box. In BAPU, the *Sender*'s data are conveyed to the *Destination* via separate tunnels from each participating BAPU-*AP*, which carries out NAT translation with NAT mapping records obtained from BAPU-*Gateway* in step 3. Besides, since the downlink capacity is enormous, we allow all reverse (downlink) traffic from *Destination* to *Sender* to traverse along the *default downlink path*. In addition, as there might be multiple tiers of NAT boxes in the middle, we must ensure that the NAT mapping for a session is properly installed on all NAT boxes along the path, and the first few packets of a new session are not tunnelled.

3.3 Scheduling

The bandwidth aggregation performance depends on the efficiency of multiplexing data among BAPU-*AP*s to best utilize the idle uplink bandwidth. In BAPU, we adopt a *centralized scheduler* at BAPU-*Gateway*. There are two main factors to select this design. *First*, it does not only simplify the implementation, but also allows easy extension of the design with extra logic to further optimize the scheduling strategy. *Second*, a scheduler usually requires complex processing and memory capability, which might overload the BAPU-*AP*s with much lower capability if scheduling decisions are distributedly performed by BAPU-*AP*s. Our scheduling strategy is based on received reports in step 6 and 7 of the protocol. Each report from a BAPU-*AP* contains a sending buffer size obtained from the Linux kernel (via `ioctl`). This value specifies how much a BAPU-*AP* can contribute to the aggregation. Based on reports, BAPU-*Gateway* applies First-Come-First-Served strategy to select a forwarder among BAPU-*AP*s who have captured the same packet. The rationale for choosing this approach are (1) *Fairness:* Sharing bandwidth takes into account the available bandwidth of participating BAPU-*AP*s because AP owners have different subscription plans. (2) *Protocol independence:* Scheduling decision is made based on the BAPU-*AP*s' sharing capability, not on the particular transport protocol.

4 TCP with Proactive-ACK

4.1 TCP Issues with Aggregation

Brief overview on TCP: TCP ensures successful and in-order data delivery between *Sender* and *Destination*. The *Sender* maintains a CWND (congestion window) during the on-going session, which determines the TCP throughput. The *Sender*'s CWND size is affected by acknowledgements from the *Destination*. First, the growth rate of CWND depends on the rate of receiving acknowledgements, i.e., the link latency. Second, missing acknowledgement within a RTO (retransmission timeout) causes the *Sender* to issue a *retransmission*. On reception of out-of-order sequences, the *Destination* sends a DUPACK (duplicate acknowledgement) to inform the *Sender* of missing packets. By default [3], the *Sender* will issue a *fast retransmission* upon receiving 3 consecutive DUPACKs. Both retransmission and fast retransmission cause the *Sender* to cut off the CWND accordingly to adapt to the congested network or slow receiver.

Performance issues with aggregation: TCP was designed based on the fact that the out-of-order sequence is generally a good indicator of lost packets or congested network. However, such assumption no longer holds in BAPU.

Out-of-order packets: In BAPU, packets belonging to the same TCP session are *intentionally* routed through multiple BAPU-*AP*s via diverse backhaul connections in terms of capacity, latency, traffic load, etc. This results in *serious* out-of-order sequence at BAPU-*Gateway*, which eventually injects the out-of-order packets to the *Destination*.

Double RTT: Also, due to the aggregation protocol, data packets in BAPU are delivered to the *Destination* with a double round-trip-time (RTT) compared to a regular link. This causes the *Sender*'s CWND to grow more slowly and peak at lower values. Consequently, with an *unplanned* aggregation method, the TCP congestion control

mechanism is *falsely* triggered, resulting in considerably low throughput. As we show later in Section 5, a simplified prototype of BAPU, which share similarities with the system in [15], gives poor TCP throughput.

Simple solution (SIMPLEBUFFER) does not work: To address the TCP performance issue, we investigate a simple approach: data packets forwarded by BAPU-*AP*s are buffered at BAPU-*Gateway* until a continuous sequence is received or a predefined buffering timeout is reached before delivering it to the *Destination*. This solution, however, encounters the following issues: 1) *Optimality:* Due to the difference in capacity, latency, loss rate among backhaul uplinks, it is unclear how to determine the optimal buffer size and timeout. 2) *Suboptimal RTT:* The buffering mechanism results in *double RTT* issue. 3) *Performance:* We implemented the buffering mechanism at BAPU-*Gateway*, and verified that it *does not* help improving the TCP throughput (Section 5.2).

4.2 BAPU's Solution

We introduce a novel mechanism called *Proactive-ACK* (step 7) to support TCP with uplink aggregation. The principle is to actively control the exchange of acknowledgements instead of relying on the default behaviour of the end-to-end session. By Proactive-ACK, we solve both *out-of-order packet* and *double RTT* issues. In the following paragraphs, we call acknowledgements actively sent by BAPU-*Gateway spoofed*, while the ones sent by the *Destination* are *real* acknowledgements.

Spoofing Proactive-ACK: In BAPU, most of out-of-order packets are caused by the aggregation mechanism via multiple BAPU-*AP*s. To avoid delivering out-of-order packets to the *Destination* and the resulting cut-off of the CWND at the *Sender*, we maintain a sequence map at BAPU-*Gateway*, indicating **reported**, **delivered** or **pending** sequence numbers. Once BAPU-*Gateway* collects a continuous range of reported sequence numbers, BAPU-*Gateway* sends a *spoofed* ACK back to the *Sender*. The intuition is that all the packets that are reported by some BAPU-*AP*s are currently stored in their buffer. Due to the reliability of the TCP tunnel, the reported packets will be eventually forwarded to BAPU-*Gateway* in reliable manner. Therefore, immediately sending a *spoofed* Proactive-ACK back to the *Sender* avoids false DUPACKs and helps maintain a healthy CWND growth at the *Sender*. Also, the RTT is not doubled.

Eliminating DUPACKs: Since *spoofed* ACKs keep the *Sender*'s CWND continuously grow, BAPU-*Gateway* can take time and buffer all out-of-order data packets forwarded from BAPU-*AP*s, and deliver *only* in-order packets to the *Destination*. Therefore, in BAPU, out-of-order packets and associated DUPACKs are eliminated from *Destination*.

Spoofing DUPACKs: It is possible that some packets are actually lost in the air between the *Sender* and BAPU-*AP*s. Concretely, if the report for an expected TCP sequence number is not received within a certain time, it is implied to be lost on all participating BAPU-*AP*s. Now that BAPU-*Gateway* sends a spoofed DUPACK back to the *Sender* in order to mimic the TCP fast retransmission mechanism for fast recovery.

Managing Real ACKs and TCP Semantics: Since BAPU-*Gateway* sends spoofed ACKs to the *Sender*, on reception of real ACKs from the *Destination*, BAPU-*Gateway* discards the real ACKs. However, BAPU-*Gateway* does save the TCP header fields in the real ACKs, such as advertised receiver window and timestamp, which maintains the TCP semantics and the state of the TCP connection. While BAPU-*Gateway* generates

the *spoofed* ACKs, it uses the latest header field values extracted from *real* ACKs to prepare the acknowledge segment.

We have one important remark on TCP semantics. If an AP which has been scheduled to forward a selected packet is suddenly offline, such packet lost would *not* be recognized by *Sender* because it has received spoofed ACK. In this case, we resort to *Home-AP* which carries out unicast between itself and *Sender* and should have a backup copy. Despite the slight difference in TCP semantics, we verify that Proactive-ACK gives a significantly improved TCP throughput. We present these results in Section 5.

5 Evaluation

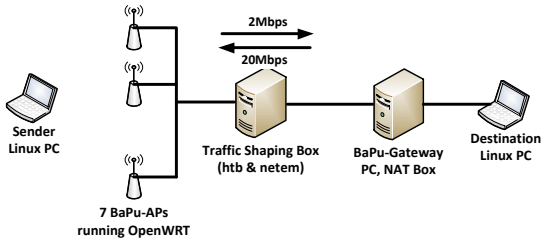


Fig. 5: BAPU Experiment Setup.

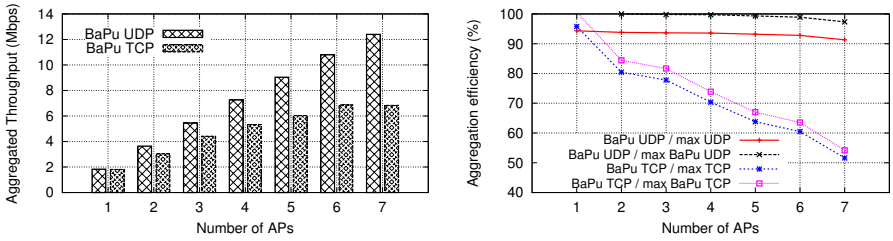
Table 1: Distance vs. Network RTT.

Regional: 500 - 1,000 mi	32ms [2]
Cross-continent: $\sim$ 3,000 mi	96ms [2]
Multi-continent: $\sim$ 6,000 mi	192ms [2]
Inter-AP in greater Boston	20ms $\sim$ 80ms

Inter-AP RTT are measured by our Open Infrastructure WiFi testbed [1] in greater Boston, covering Comcast, RCN, and Verizon.

In this section, we evaluate the performance of BAPU for UDP and TCP in various system settings. Our experiment setup is shown in Figure 5. Our testbed consists of a *Sender*, 7 BAPU-*AP*s, a BAPU-*Gateway*, a *Destination* and a traffic shaping box. All APs are Buffalo WZR-HP-G300NH 802.11n wireless routers. This model has a 400MHz CPU with 32MB RAM. We reflashed the APs with OpenWRT firmware, running Linux kernel 2.6.32 and `ath9k` WiFi driver. In our experiments, we select one BAPU-*AP* as a *Home-AP* which the *Sender* is always associated to, the other 6 BAPU-*AP*s act as *Monitor-AP*s to capture the traffic in monitor mode. The BAPU-*Gateway* runs on a Linux PC, and the *Destination* runs behind the BAPU-*Gateway*. The *Sender* and the *Destination* are both laptops with 802.11n WiFi card, running the standard Linux TCP/IP stack. To emulate traffic shaping as with residential broadband, we use the traffic shaping box between the BAPU-*AP*s and BAPU-*Gateway*. We use Linux' `iptables` and `tc` with the `htb` module to shape the downlink bandwidth to 20Mbps and the uplink to 2Mbps. Also, to emulate network latency between BAPU-*AP*s and BAPU-*Gateway*, we use `netem` to shape the RTT with different values. The bandwidth and latency parameter are selected to represent the typical bandwidth capacity and latency in residential cable broadband measured in Boston's urban area (Table 1).

In our experiments, we issue long-lived 30 minutes `iperf` flows (both TCP and UDP) from *Sender* to *Destination*. We choose 1350Byte as TCP/UDP payload size to make sure that the whole client IP packet can be encapsulated in one IP packet while an BAPU-*AP* sends it through its TCP tunnel. All throughput values reported in our experiment are the `iperf` throughput, which is the *goodput*. In the evaluation, we compare throughput of UDP and TCP in a variety of scenarios: A. BAPU – BAPU system without any buffering or Proactive-ACK mechanism; B. SIMPLEBUFFER – BAPU system



(a) UDP and TCP throughput

(b) UDP and TCP aggregation efficiency

Fig. 6: BAPU aggregation for UDP and TCP with 2Mbps 32ms RTT uplinks.

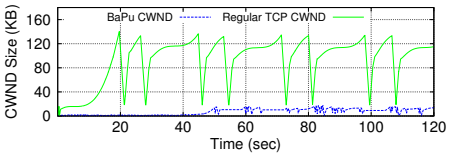


Fig. 7: Sender's TCP CWND growth: BAPU vs. Regular TCP.

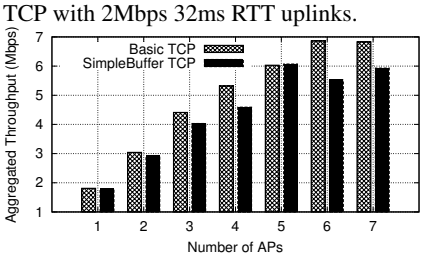


Fig. 8: BAPU vs. SIMPLEBUFFER.

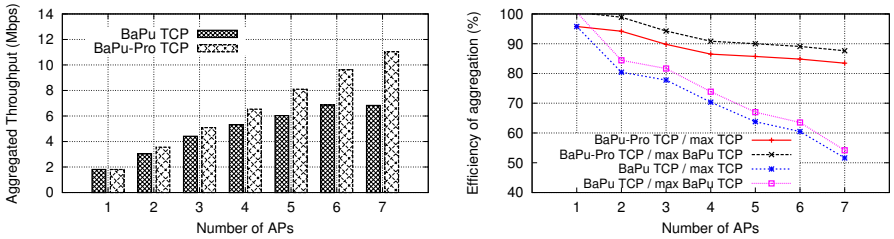
without Proactive-ACK, but enhanced by buffering at BAPU-*Gateway*; C. BAPU-PRO – this is the full BAPU system.

5.1 BAPU: Efficient UDP, Poor TCP

System efficiency with UDP throughput: We now first measure BAPU's efficiency by the throughput with UDP, as it provides a light-weight end-to-end transmission between *Sender* and *Destination*. Figure 6a shows the achieved aggregated UDP throughput with numbers of participating BAPU-*AP*s increasing from 1 to 7. We observe that the aggregated UDP throughput increases proportionally with the number of BAPU-*AP*s, and achieves 12.4Mbps with 7 BAPU-*AP*s. To put this figure into perspective, note that related work by Jakubczak et al. [15] achieves similar UDP throughput but *without* support for TCP or client transparency.

Low TCP throughput: We conduct the same experiments also for TCP transmission. Figure 6a shows that the aggregated TCP throughput does not benefit much when the number of BAPU-*AP*s increases. The TCP aggregated throughput is always lower than the UDP's in the same setup, and the gap between UDP and TCP performance increases along with the number of BAPU-*AP*s.

Aggregation efficiency: In addition to measuring aggregated throughput, we evaluate our system based on another metric, *aggregation efficiency*. We define *aggregation efficiency* as the ratio between practical throughput over the maximum theoretical goodput. Due to the TCP/IP header and BAPU protocol overhead, the actual goodput is less than the uplink capacity. With all protocol header overhead accounted, we derive the maximum theoretical goodput as the given backhaul capacity of 2Mbps. As shown in Figure 6b, BAPU UDP can harness close to 100% idle bandwidth. Even if we consider the extra overhead incurred by BAPU protocol messages, UDP aggregation efficiency



(a) Aggregated TCP throughput

(b) Aggregation efficiency

Fig. 9: BAPU-PRO vs. BAPU: comparison with 2Mbps 32ms RTT uplinks.

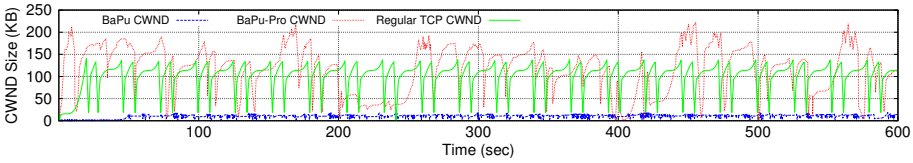


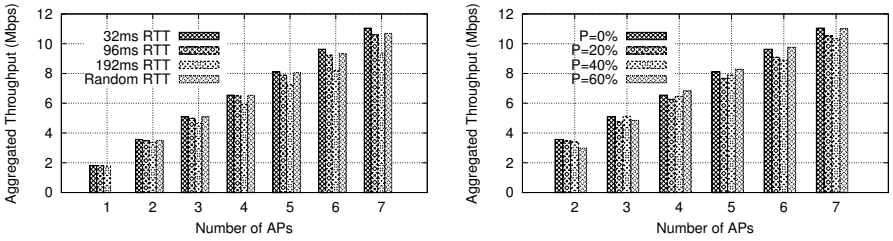
Fig. 10: Sender's TCP CWND growth: BAPU-PRO vs. BAPU vs. Regular TCP.

is still over 90% in all cases. In contrast, the aggregation efficiency for TCP degrades quickly as more BAPU-*AP*s join the cooperation. With 7 BAPU-*AP*s, BAPU transforms only 50% of idle bandwidth to effective throughput.

Discussion on BAPU's poor TCP performance: We can observe several factors in Section 4 that decrease the aggregated TCP throughput. In this section, we carry out an analysis on the *Sender*'s CWND size in BAPU. To justify our analysis, we inspect the TCP behavior by examining the Linux kernel TCP stack variables. We call `getsockopt()` to query the `TCP_INFO` data structure containing the system time stamp, *Sender*'s CWND, number of retransmissions, etc. We also modified the `iperf` code to log `TCP_INFO` for each call to send application data. Figure 7 shows the CWND growth in a 120 second `iperf` test with 7 BAPU-*AP*s (theoretical throughput is 2Mbps $\times$ 7 $=$ 14Mbps) in comparison with standard TCP through a single AP with 14Mbps uplink capacity. The *Sender*'s CWND remains at a very low level. Our captured packet trace at the *Sender* shows that lots of DUPACK packets and RTO incur a lot of retransmissions, resulting in low TCP throughput.

5.2 Does SIMPLEBUFFER help TCP performance?

As discussed in Section 4, a simple *buffering* mechanism does *not* solve the TCP performance issue due to difference in BAPU-*AP* uplink characteristics (latency, packet loss). In this section, we experimentally show that a buffering mechanism cannot help in improving the TCP throughput. Figure 8 depicts the throughput comparison between BAPU and SIMPLEBUFFER. Surprisingly, the throughput is even degraded with SIMPLEBUFFER. Our trace inspection shows a lot of TCP Timeout Retransmissions due to the packets being buffered at BAPU-*Gateway* for too long.



(a) Different RTT

(b) Different packet loss rate P on *Monitor-AP*s

Fig. 11: BAPU-PRO TCP throughput.

5.3 BAPU-PRO Performance

We now conduct a comprehensive set of experiments to evaluate the performance of BAPU-PRO. First, we validate our Proactive-ACK mechanism by comparing BAPU-PRO against BAPU. Second, we measure the performance of BAPU-PRO under a variety of network settings (network latency, wireless link quality, etc.). Finally, we demonstrate that BAPU-PRO is feasible for both, streaming and large file transfer applications.

TCP Throughput – BAPU-PRO vs. BAPU: We carry out the same `iperf` test as described in Section 5.1 with BAPU-PRO. As shown in Figure 9a, the aggregated TCP throughput of BAPU-PRO significantly outperforms the one of BAPU. With 7 BAPU-*AP*s, BAPU-PRO achieves 11.04Mbps, i.e., 62% improvement over BAPU. Furthermore, Figure 9b shows that BAPU-PRO achieves at least 88% aggregation efficiency in our setup, and it achieves at least 83% of the upper limit of standard TCP throughput. These results demonstrate that BAPU-PRO can achieve high aggregated throughput with high aggregation efficiency for TCP in practical settings.

Proactive-ACK benefit: To justify our Proactive-ACK mechanism, we adopt the same method as in Section 5.1 to examine the TCP CWND growth. Figure 10 shows that BAPU-PRO allows the CWND to grow to very high values, contributing to the high throughput. For convenience, we also run a regular TCP session with a throttled bandwidth 11Mbps (similar to the BAPU-PRO's resulted throughput). The CWND growth for BAPU-PRO and regular TCP shares a similar pattern, which implies that our design and implementation can efficiently and transparently aggregate multiple slow uplinks.

Impact of network latency: For TCP transmissions, RTT is an important factor that has impact on the throughput. We measure the performance of BAPU with different network latency settings listed in Table 1. Besides fixed latency values for each typical setting, we also assign to each BAPU-*AP* a random RTT value between 20ms and 80ms. We carry out this test for 10 runs and report the average throughput. As shown in Figure 11a, BAPU-PRO throughput slightly declines as network latency increases. In random latency setting, the resulted throughput shows no significant difference.

Impact of lossy wireless links: The wireless links in a real neighbourhood can be very lossy for a variety of reasons, such as cross channel interference and distant neighboring APs. Besides, since *Monitor-AP*s switch between transmit and receive mode, they cannot overhear all transmitted packets. To estimate the potential of BAPU highly lossy wireless environments, we emulate packet loss at *Monitor-AP*s by dropping received packets with a probability P. No losses were inflicted on *Home-AP*, because *Sender*

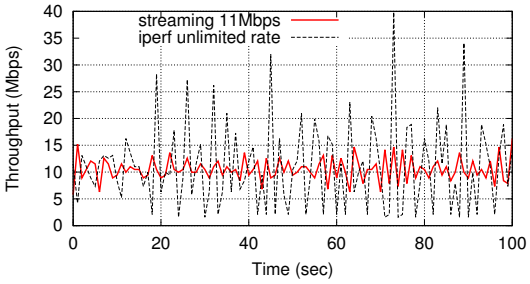


Fig. 12: Instantaneously received throughput: 11Mbps Streaming vs. Unlimited rate.

carries out unicast to *Home-AP*, and 802.11 MAC already handles packet loss and retransmissions automatically. We conduct the experiment with 3 values of P: 20%, 40%, and 60%. As indicated in Figure 11b, the throughput reduction on lossy wireless links is very limited in all cases. The good performance can be explained by the link diversity combined with the centralized scheduling mechanisms. The probability of some packet not overheard by *at least one Monitor-AP* is negligible small, especially in case of high number of participating APs. This also explains why 7 BAPU-*AP*s achieve higher throughput with $P = 60\%$ than with $P = 20\%$.

Streaming vs. large file transfer: One important goal in BAPU's design is to support instant sharing of high-bitrate HD videos directly between users using streaming. The motivation behind is that today the major online streaming services (e.g., Netflix) run on TCP based streaming technologies, such as HTTP based Adaptive Bitrate Streaming. Real time streaming generally requires *stable* instantaneous throughput. In this experiment, we study the potential of BAPU as a solution to high-bitrate real-time streaming. To emulate HD streaming, we use `nuttcp` to issue a TCP flow with a fixed 11Mbps sending rate. As shown in Figure 12, `nuttcp` achieves a reasonably stable instantaneous throughput during a 100 second session. It implies that BAPU can sustain high-bitrate streaming through aggregated uplinks. In comparison, the `iperf` flow with unlimited sending rate shows much higher fluctuation.

6 Related Work

While BAPU is inspired by design principles of previous work, it addresses unique constraints and goals and presents a set of novel techniques that achieve high efficiency. Previous research has addressed TCP performance improvements over wireless links by using intermediate nodes that assist in the recovery of lost packets, e.g., Snoop TCP [5], and Split TCP [18]. Multiple radio links for improving throughput have also been explored from several perspectives including traffic aggregation [17], multipath forwarding [15], and mitigation of wireless losses [21, 22]. In addition to systems that rely on multiple radio interfaces [4], other solutions and algorithms have been proposed for a single client radio interface that switches across multiple access points while providing upper layers of the network stack with a transparent access [8, 17, 20, 28]. Solutions to overcome limited APs backhaul through aggregation using such a virtualized radio interfaces include the initial Virtual-WiFi [20] system where two TCP connection are ser-

viced by two different APs, FatVAP [17] and ARBOR [28] that achieve fast switching by smart AP selection, and Fair WLAN [12] for fairness. These systems require techniques for fast switching across access points to reduce impact on TCP performance, e.g., delay and packet loss as discussed in Juggler [16] and WiSwitcher [11]. An analytical model [25] is proposed to optimize concurrent AP connections for highly mobile clients. They also implement Spider, a multi-AP driver using optimal AP and channel scheduling to improve the aggregated throughput. *Unlike* BAPU, these papers do not focus on aggregating the throughput for single transport layer connection, which is critical for *client transparency*. Divert [22] and ViFi [6] reduce path-dependent downlink loss from an AP to a client. However, rather than improving the wireless link quality, BAPU targets aggregation of the wired capacity behind APs. In BAPU, the sender regularly communicates with its home AP. As discussed, BAPU borrows ideas from Link-alike [15] where access points coordinate to opportunistically schedule the traffic over backhaul links. Contrary to Link-alike, BAPU does not require client devices to use broadcast. Moreover, BAPU transparently supports protocols like TCP. Being completely transparent to the clients and constraining each link AP-Destination flow to be TCP-friendly makes efficient multipath transport a key component of our system. We stress that, in contrast to BAPU, the large body of related work on multipath transport, cf. [7, 10, 13, 14, 19, 23, 24, 26, 27], does not support transparent, unmodified client devices and TCP/IP stacks while efficiently aggregating AP backhaul.

7 Conclusion

In this work, we present the design and implementation of BaPu, a complete software based solution on WiFi APs for aggregating multiple broadband uplinks. First, based on our large scale wardriving data and long term measurement in Boston's residential broadband, we show that the high AP density and under utilized broadband uplinks suit solutions that harness idle bandwidth to improve uplink throughput. Contrary to related work, BaPu offers a client transparent design, generic support for legacy devices, and a large variety of network applications. To this end, BAPU employs a novel mechanism (Proactive-ACK) to address the challenges of multiplexing single TCP sessions through multiple paths without degrading performance. To analyze the benefits of BaPu, we have carried out an extensive set of experiments for both UDP and TCP in a variety of realistic network settings. BaPu achieves over 95% aggregation efficiency for UDP and over 88% for TCP – even in lossy wireless environment. As a future work, it would be interesting to reproduce and compare the results in different neighborhoods and different countries. Also, incentive mechanisms and support from AP manufacturers and subscription providers need to be developed in order for BAPU to be useful for both AP owners and users.

References

[1] Open infrastructure: A wireless network research framework for residential networks. http://www.ccs.neu.edu/home/noubir/projects/openinfrastructure/.

[2] Akamai. Akamai HD Network. *Technical Report*, 2011. http://bit.ly/1xN2NNB.

[3] M. Allman, V. Paxson, and E. Blanton. Tcp congestion control, 2009.
[4] P. Bahl, A. Adya, J. Padhye, and A. Walman. Reconsidering wireless systems with multiple radios. *SIGCOMM Compututer Communication Review*, 34:39–46, 2004. ISSN 0146-4833.
[5] H. Balakrishnan, S. Seshan, E. Amir, and R.H. Katz. Improving TCP/IP performance over wireless networks. In *Proc. of MobiCom*, 1995.
[6] A Balasubramanian, R Mahajan, A Venkataramani, BN Levine, and J Zahorjan. Interactive wifi connectivity for moving vehicles. *Proc. of SigComm*, 2008.
[7] S. Barré, C. Paasch, and O. Bonaventure. MultiPath TCP: from theory to practice. In *Proceedings of NETWORKING*, 2011.
[8] R. Chandra and P. Bahl. Multinet: connecting to multiple ieee 802.11 networks using a single wireless card. In *Proc. of INFOCOM*, 2004.
[9] FON. FON, 2012. http://corp.fon.com/us/.
[10] A. Ford, C. Raiciu, M. Handley, and O. Bonaventure. TCP Extensions for Multipath Operation with Multiple Addresses. Internet-Draft, 2012.
[11] D. Giustiniano, E. Goma, A. Lopez, and P. Rodriguez. Wiswitcher: an efficient client for managing multiple aps. In *Proc. of PRESTO*, 2009.
[12] D. Giustiniano, E. Goma, A.L. Toledo, I. Dangerfield, J. Morillo, and P. Rodriguez. Fair WLAN backhaul aggregation. In *MobiCom*, 2010.
[13] H.-Y. Hsieh and R. Sivakumar. A transport layer approach for achieving aggregate bandwidths on multi-homed mobile hosts. In *Proc. of MobiCom*, 2002.
[14] H.-Y. Hsieh, K.-H. Kim, Y. Zhu, and R. Sivakumar. A receiver-centric transport protocol for mobile hosts with heterogeneous wireless interfaces. In *Proc. of MobiCom*, 2003.
[15] S. Jakubczak, D. G. Andersen, M. Kaminsky, K. Papagiannaki, and S. Seshan. Link-alike: using wireless to share network resources in a neighborhood. *SIGMOBILE Mobile Computing Communications Review*, 2008.
[16] Anthony J.N., Scott W., and B.D. Noble. Juggler: Virtual networks for fun and profit. *IEEE Transactions on Mobile Computing*, 9:31–43, 2010.
[17] S. Kandula, K.C. Lin, T. Badirkhanli, and D. Katabi. FatVAP: Aggregating AP backhaul capacity to maximize throughput. In *Proc. of NSDI*, 2008.
[18] S. Kopparty, S.V. Krishnamurthy, M. Faloutsos, and S.K. Tripathi. Split tcp for mobile ad hoc networks. In *GLOBECOM*, 2002.
[19] L. Magalhaes and R.H. Kravets. Transport Level Mechanisms for Bandwidth Aggregation on Mobile Hosts. In *Proc. of Conference on Network Protocols*, 2001.
[20] Microsoft Research. Virtual wifi, 2012. http://bit.ly/1IjD4iw.
[21] A. Miu, H. Balakrishnan, and C.E. Koksal. Improving loss resilience with multi-radio diversity in wireless networks. In *MobiCom*, pages 16–30, 2005.
[22] A.K. Miu, G. Tan, H. Balakrishnan, and J. Apostolopoulos. Divert: Fine-grained path selection for wireless lans. In *Proc. of MobiSys*, 2004.
[23] B. Radunović, C. Gkantsidis, D. Gunawardena, and P. Key. Horizon: Balancing TCP over multiple paths in wireless mesh network. In *MobiCom*, 2008.
[24] C. Raiciu, S. Barre, C. Pluntke, A. Greenhalgh, D. Wischik, and Ma. Handley. Improving datacenter performance and robustness with multipath TCP. In *SIGCOMM '11*, 2011.
[25] H. Soroush, P. Gilbert, N. Banerjee, B.N. Levine, M. Corner, and L. Cox. Concurrent Wi-Fi for mobile users: analysis and measurements. In *CoNEXT*, 2011.
[26] R. Steward. Stream control transmission protocol. IETF RFC 4960, 2007.
[27] D. Wischik, C. Raiciu, A. Greenhalgh, and M. Handley. Design, implementation and evaluation of congestion control for multipath TCP. In *Proc. of NSDI*, 2011.
[28] X. Xing, S. Mishra, and X. Liu. ARBOR: hang together rather than hang separately in 802.11 wifi networks. In *Proc. of INFOCOM*, 2010.