Experiments with TCP over WiFi

Rakhat Yskak

***Index Terms*—TCP, WiFi, Tests, Frame Aggregation.**

I. Literature Review

Avkhadeev evaluated delayed acknowledgment schemes in Transmission Control Protocol (TCP) over Wi-Fi [1]. He implemented TCP-Adaptive Acknowledgment Delay (TCP-AAD) in a Linux kernel. Simulations revealed that adaptive delays based on packet inter-arrival times improve throughput. This work identified optimal delay windows to balance acknowledgment overhead and burst-induced losses.

TCP stands as the key protocol for reliable online data transfer. It ensures packets arrive correctly and in sequence. Early standards established its core principles.

A. TCP Basics

RFC 793 defines TCP's essential mechanisms [2]. It explains connection establishment through a three-way handshake. Senders transmit segments with sequence numbers to track order. Receivers use acknowledgments to signal successful delivery. This setup provides reliability over unreliable networks.

RFC 1122 enhances these foundations for internet hosts [3]. It addresses implementation details like handling fragmented packets. The document also covers requirements for robust error detection. These refinements help TCP operate effectively in complex environments.

B. Congestion Management

Excess traffic can overwhelm networks. RFC 896 introduced congestion control concepts in IP/TCP systems [4]. It described scenarios where unchecked growth leads to collapse. The memo emphasized the need for senders to react to network signals.

Modern TCP employs specific algorithms to manage load. RFC 5681 specifies slow start to gradually increase sending rates [5]. Congestion avoidance follows to maintain steady flow. Fast retransmit and recovery handle detected losses quickly. These methods prevent persistent overload.

RFC 2018 introduces selective acknowledgments (SACK) [6]. This option lets receivers specify received segments precisely. Senders can retransmit only missing parts. SACK improves efficiency during bursts of losses.

CUBIC targets high-bandwidth, long-delay paths. RFC 8312 outlines its cubic function for window growth [7]. This approach achieves better scalability than traditional linear methods. CUBIC maintains stability while probing for available capacity.

C. High-Performance Additions

Paths with large bandwidth-delay products require expanded capabilities. RFC 7323 defines the window scale option to support bigger receive windows [8]. It also introduces timestamps for accurate round-trip time measurement. Timestamps enable protection against wrapped sequence numbers in fast networks.

These extensions allow TCP to utilize high-speed links fully. Larger windows prevent underuse of capacity. Timestamps aid in distinguishing old duplicates from new data.

D. Wireless Issues

Wireless environments pose unique challenges. Links suffer from bit errors and interference. Balakrishnan et al. evaluated techniques to boost TCP over wireless [9]. They compared end-to-end, link-layer, and split-connection approaches. Link-layer retransmissions proved effective in hiding losses from TCP.

Xylomenos et al. provided an overview of TCP/IP performance in wireless networks [10]. They discussed how random errors trigger unnecessary congestion responses. The paper highlighted the need for adaptations to distinguish error types.

Pokhrel et al. developed a model for TCP over Wi-Fi [11]. It accounts for both buffer overflows and channel losses. Their analysis reveals interactions between these factors. The model predicts throughput under varying conditions.

E. Delayed Acknowledgments

Delayed acknowledgments reduce control traffic. Receivers delay responses to batch confirmations.

Altman and Jiménez proposed innovative delayed ACK methods for multihop wireless networks [12]. They introduced variable delay coefficients based on sequence numbers. Simulations showed substantial throughput gains from ACK thinning.

Chen et al. investigated TCP with delayed ACK in wireless settings [13]. They observed that unrestricted delays do not always help. Optimal window sizes depend on path lengths. Longer paths increase interference from bursty transmissions.

Avkhadeev assessed delayed acknowledgment schemes over Wi-Fi [1]. He implemented TCP-AAD in a Linux kernel. The work evaluated performance through simulations. Results indicated adaptive delays improve throughput in various scenarios.

F. Wi-Fi Advances

IEEE 802.11n introduced enhancements for higher throughput. Xiao described features like MIMO and channel bonding [14]. MIMO uses multiple antennas to increase data rates. Bonding combines channels for wider bandwidth.

TABLE I
SUMMARY OF KEY LITERATURE ON TCP OVER WI-FI

Reference	Year	Key Contributions	Relevance to Thesis
Avkhadeev [1]	2025	Evaluates delayed ACK schemes, implements TCP-AAD in Linux.	Basis for real-world testing in this thesis.
RFC 793 [2]	1981	Defines core TCP mechanisms like acknowledgments and retransmissions.	Provides foundational understanding of TCP behavior.
RFC 1122 [3]	1989	Specifies host requirements for TCP/IP, refining protocol operations.	Ensures compliance in experimental setups.
RFC 896 [4]	1984	Introduces congestion control principles in TCP/IP networks.	Basis for handling congestion in Wi-Fi experiments.
RFC 5681 [5]	2009	Standardizes TCP congestion control algorithms.	Guides implementation of congestion avoidance in tests.
RFC 2018 [6]	1996	Introduces Selective Acknowledgments (SACK) for better loss recovery.	Enhances TCP resilience over error-prone Wi-Fi links.
RFC 7323 [8]	2014	Extensions for high-performance TCP, e.g., larger windows.	Supports high-throughput experiments over Wi-Fi.
RFC 8312 [7]	2018	CUBIC algorithm for fast long-distance networks.	Potential alternative congestion control for Wi-Fi scenarios.
Balakrishnan et al. [9]	1997	Compares mechanisms for TCP improvement over wireless links.	Identifies challenges like packet loss in Wi-Fi.
Xylomenos et al. [10]	2002	Overview of TCP/IP performance issues in wireless networks.	Contextualizes TCP pitfalls in Wi-Fi environments.
Pokhrel et al. [11]	2016	Models TCP throughput considering buffer and channel losses in Wi-Fi.	Aids in predicting performance in real scenarios.
Altman and Jiménez [12]	2003	Novel delayed ACK techniques for multihop wireless networks.	Builds on delayed ACK for performance gains.
Chen et al. [13]	2008	TCP with delayed ACK optimized for wireless networks.	Directly relates to extending delayed ACK schemes.
Xiao [14]	2005	Enhancements in IEEE 802.11n for higher Wi-Fi throughput.	Informs hardware considerations in experiments.
Deek et al. [15]	2011	Impact of channel bonding on 802.11n management.	Highlights Wi-Fi configuration effects on TCP.
Saha et al. [16]	2015	Power-throughput tradeoffs in 802.11n/ac for smartphones.	Considers energy aspects in mobile Wi-Fi tests.

Deek et al. analyzed channel bonding's effects on 802.11n networks [15]. They found wider channels boost capacity but heighten interference. The study emphasized careful management for optimal performance.

Saha et al. examined power and throughput tradeoffs in 802.11n/ac smartphones [16]. Higher features like MIMO consume more energy. Their experiments highlighted balances needed for mobile devices.

G. Summary of Literature

Table I summarizes the reviewed works. It lists each reference with its publication year. The table details key contributions and their ties to this thesis. Entries group into categories like fundamentals, congestion control, wireless challenges, delayed ACKs, and Wi-Fi enhancements. This structure shows the progression from basic TCP to specific Wi-Fi optimizations. The table aids quick reference to foundational and advanced sources.

This thesis extends Avkhadeev's simulation-based evaluation [1]. It applies the delayed ACK scheme in real hardware setups. Tools like tc control traffic shaping. iperf measures throughput in client-server tests.

II. METHODOLOGY

This section describes the experimental setup, test parameters, data collection procedures, and analysis methods used to evaluate TCP-AAD performance over Wi-Fi networks.

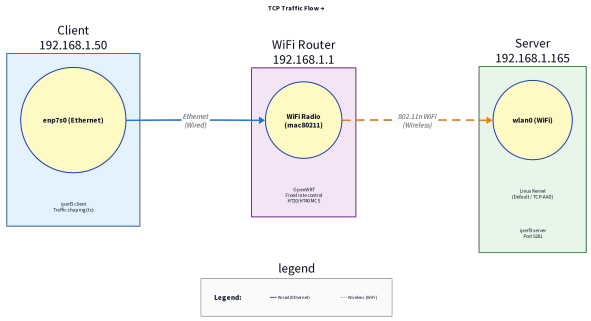

Fig. 1. Experimental network topology. The client connects via Ethernet to perform traffic shaping with tc, while the server connects over WiFi to measure TCP performance. The router runs OpenWRT with configurable WiFi rate control.

A. Experimental Setup

The experiments employ a controlled three-node topology consisting of a client, server, and wireless router. Figure 1 illustrates the network architecture.

The client machine connects to the network via Ethernet interface (enp7s0) at IP address 192.168.1.50. This wired connection ensures stable baseline conditions for initiating TCP flows. The client runs iperf3 in client mode to generate TCP traffic.

The server machine connects wirelessly through interface

TABLE II
EXPERIMENTAL PARAMETERS AND VALUES

Parameter	Values
Kernel Variant	Default, TCP-AAD
Bandwidth Limit	20, 50, 100 Mbps, unlimited
Network Delay	10, 50, 100 ms, no delay
WiFi Rate (HT20)	MCS 4, 5, 6, 12, 13, 14
WiFi Rate (HT40)	MCS 4, 5, 6, 12, 13, 14
Iterations	3 per configuration
Test Duration	30 seconds
Total Tests	1152 (576 per kernel)

wlan0 at IP address 192.168.1.165. This placement allows direct measurement of TCP performance over the Wi-Fi link. The server executes iperf3 in server mode to receive and measure incoming traffic on port 5201.

The wireless router operates at IP address 192.168.1.1. It runs an OpenWRT-based firmware that provides fine-grained control over Wi-Fi parameters. The router configuration enables manipulation of transmission rates through the mac80211 wireless subsystem.

Both client and server machines run Linux with kernel version 5.15 or later. The server can boot into two kernel variants: the standard Linux kernel or a modified kernel implementing TCP-AAD. This dual-boot capability enables direct comparison between conventional TCP behavior and the adaptive acknowledgment scheme.

B. Experimental Parameters

The experiments systematically vary multiple factors to characterize TCP-AAD performance across diverse network conditions. Table II summarizes the test parameters and their values.

1) Kernel Variants: Tests run with two kernel configurations on the server:

- **Default kernel**: Standard Linux TCP implementation without modifications
- **TCP-AAD kernel**: Modified implementation with adaptive acknowledgment delays based on packet inter-arrival times

2) Bandwidth Limits: Traffic shaping controls the available bandwidth at the client interface. Four bandwidth settings capture different network capacity scenarios:

- 20 Mbps: Simulates constrained mobile or rural broadband
- 50 Mbps: Represents moderate residential connections
- 100 Mbps: Models high-speed residential or enterprise access
- Unlimited (nolim): No artificial bandwidth constraint

3) Network Delays: Artificial delays emulate varying propagation times and queuing delays. Four delay configurations span typical wireless scenarios:

- 10 ms: Local area network or short-distance wireless
- 50 ms: Regional or moderate-distance connections
- 100 ms: Long-distance or satellite-like delays
- No delay (nodelay): Zero added latency beyond natural propagation

4) Wi-Fi Transmission Rates: The router fixes transmission rates to isolate rate adaptation effects. Twelve rate indices cover combinations of channel width and modulation coding scheme (MCS):

- **HT20 configurations**: 20 MHz channels with MCS 4, 5, 6, 12, 13, 14
- **HT40 configurations**: 40 MHz channels with MCS 4, 5, 6, 12, 13, 14

These rates span different spatial stream counts (single and dual stream) and modulation schemes. HT20 rates range from approximately 13 to 104 Mbps, while HT40 rates extend up to 216 Mbps under ideal conditions.

5) Iterations: Each unique configuration runs three times to capture performance variability. Multiple iterations enable statistical analysis and identification of outliers.

The complete parameter matrix yields $2 \times 4 \times 4 \times 12 \times 3 = 1152$ total tests (2 kernels, 4 bandwidths, 4 delays, 12 rates, 3 iterations).

C. Test Procedure

An automated script orchestrates the experiment execution. The script iterates through all parameter combinations in a systematic manner.

For each configuration, the procedure follows these steps:

1) **Router rate configuration**: The script establishes an SSH connection to the router and sets the fixed transmission rate via the mac80211 debugfs interface. The command writes the rate index to `/sys/kernel/debug/ieee80211/phy1/rc/fixed_rate`.
2) **Settlement delay**: The system waits 3 seconds after setting the rate to allow the wireless interface to stabilize on the new transmission parameters.
3) **Traffic shaping application**: The script applies bandwidth limits and delays on the client's Ethernet interface using Linux Traffic Control (tc). For bandwidth limiting, it employs the Token Bucket Filter (tbf) qdisc. For delay injection, it uses the Network Emulator (netem) qdisc. When both constraints apply, the script creates a hierarchical configuration with tbf as the root and netem as a child qdisc.
4) **Pre-test metadata capture**: Before the throughput test begins, the script collects system state information including WiFi station statistics from the router, TCP socket parameters from the server, and kernel version details. This metadata aids in validating test conditions and diagnosing anomalies.
5) **Throughput measurement**: The client initiates a 30-second iperf3 TCP transfer to the server. The tool measures instantaneous and aggregate throughput, retransmission counts, and other TCP metrics. Results save in JSON format for structured parsing.
6) **Traffic shaping removal**: The script removes all tc rules from the client interface to restore normal operation.
7) **Inter-test delay**: A 5-second pause separates consecutive tests to allow TCP connections to close completely and queues to drain.

All test outputs, including iperf3 JSON files, metadata snapshots, and execution logs, store in a hierarchical directory structure organized by kernel type and configuration parameters.

D. Data Collection

The experimental framework captures multiple data streams to enable comprehensive analysis.

1) Throughput Metrics: iperf3 records detailed performance statistics in JSON format. Key metrics include:

- **Bits per second**: Achieved goodput averaged over the test duration
- **Bytes transferred**: Total payload successfully delivered
- **Retransmissions**: Count of TCP segment retransmissions indicating packet loss
- **Test duration**: Actual elapsed time for the transfer

2) Wireless Statistics: The router provides Wi-Fi layer information through the iw utility and debugfs interfaces:

- **Station dump**: Connection quality metrics including signal strength, transmission rate, and frame statistics
- **Rate control status**: Current fixed rate index confirming correct configuration

3) Transport Layer State: The ss (socket statistics) command on the server captures TCP connection parameters:

- **Congestion window (cwnd)**: Current sending window size
- **Round-trip time (RTT)**: Smoothed and variance measurements
- **Receiver window**: Advertised window from the client

4) Test Metadata: Each test generates a metadata file containing:

- Timestamp of execution
- Kernel variant identifier
- Configuration parameters (bandwidth, delay, rate, iteration)
- System state snapshots

This metadata enables precise correlation between test conditions and observed performance.

E. Data Analysis

Python scripts process the collected data to extract insights and generate visualizations.

1) Result Parsing: The analysis pipeline begins by parsing iperf3 JSON output files. A custom parser extracts throughput, retransmission, and timing metrics from the end-of-test summaries. File naming conventions encode test parameters, allowing automatic association of results with experimental conditions.

2) Aggregation and Statistics: Results group by configuration tuple (bandwidth, delay, rate). For each configuration, the analysis computes:

- **Mean throughput**: Average across iterations
- **Standard deviation**: Measure of performance variability
- **Median throughput**: Robust central tendency metric
- **Retransmission rate**: Average packet loss indicator

3) Comparative Analysis: The comparison module pairs results from TCP-AAD and default kernel tests under identical conditions. For each configuration, it calculates:

- **Throughput ratio**: TCP-AAD performance relative to default
- **Absolute improvement**: Difference in Mbps
- **Percentage gain**: Relative performance change

4) Visualization: Plotting scripts generate publication-quality figures including:

- **Bar charts**: Throughput comparisons across rate configurations
- **Heatmaps**: Performance across bandwidth-delay matrices
- **Box plots**: Distribution of results showing variability
- **Time series**: Per-second throughput evolution during tests

These visualizations facilitate identification of trends, optimal configurations, and performance boundaries.

The systematic methodology ensures reproducible measurements and enables rigorous evaluation of TCP-AAD effectiveness across the parameter space representative of real-world Wi-Fi deployments.

References

[1] A. F. Avkhadeev, "Performance evaluation of delayed acknowledgment scheme of tcp over wifi," 2025. Supervisor: Nikola Zlatanov; Consultant: Shinnazar Seytnazarov.

[2] "Transmission Control Protocol." RFC 793, Sept. 1981.

[3] R. T. Braden, "Requirements for Internet Hosts - Communication Layers." RFC 1122, Oct. 1989.

[4] "Congestion Control in IP/TCP Internetworks." RFC 896, Jan. 1984.

[5] E. Blanton, V. Paxson, and M. Allman, "TCP Congestion Control." RFC 5681, Sept. 2009.

[6] S. Floyd, J. Mahdavi, M. Mathis, and A. Romanow, "TCP Selective Acknowledgment Options." RFC 2018, Oct. 1996.

[7] I. Rhee, L. Xu, S. Ha, A. Zimmermann, L. Eggert, and R. Scheffenegger, "CUBIC for Fast Long-Distance Networks." RFC 8312, Feb. 2018.

[8] D. Borman, R. T. Braden, V. Jacobson, and R. Scheffenegger, "TCP Extensions for High Performance." RFC 7323, Sept. 2014.

[9] H. Balakrishnan, V. N. Padmanabhan, S. Seshan, and R. H. Katz, "A comparison of mechanisms for improving tcp performance over wireless links," *IEEE/ACM Transactions on Networking*, vol. 5, no. 6, pp. 756–769, 1997.

[10] G. Xylomenos, G. Polyzos, P. Mahonen, and M. Saaranen, "Tcp/ip performance over wireless networks," June 2002.

[11] S. Pokhrel, M. Panda, H. Vu, and M. Mandjes, "Tcp performance over wi-fi: Joint impact of buffer and channel losses," *IEEE Transactions on Mobile Computing*, 2016.

[12] E. Altman and T. Jiménez, "Novel delayed ack techniques for improving tcp performance in multihop wireless networks," in *Personal Wireless Communications* (M. Conti, S. Giordano, E. Gregori, and S. Olariu, eds.), (Berlin, Heidelberg), pp. 237–250, Springer Berlin Heidelberg, 2003.

[13] J. Chen, M. Gerla, Y. Z. Lee, and M. Y. Sanadidi, "Tcp with delayed ack for wireless networks," *Ad Hoc Networks*, vol. 6, no. 7, pp. 1098–1116, 2008.

[14] Y. Xiao, "Ieee 802.11n: enhancements for higher throughput in wireless lans," *IEEE Wireless Communications*, vol. 12, no. 6, pp. 82–91, 2005.

[15] L. Deek, E. Garcia-Villegas, E. Belding, S.-J. Lee, and K. Almeroth, "The impact of channel bonding on 802.11n network management," in *Proceedings of the Seventh Conference on Emerging Networking Experiments and Technologies (CoNEXT '11)*, (New York, NY, USA), Association for Computing Machinery, 2011.

[16] S. K. Saha, P. Deshpande, P. P. Inamdar, R. K. Sheshadri, and D. Koutsonikolas, "Power-throughput tradeoffs of 802.11n/ac in smartphones," in *2015 IEEE Conference on Computer Communications (INFOCOM)*, pp. 100–108, 2015.