

MUST, SHOULD, DON'T CARE: TCP Conformance in the Wild

Mike Kosek, Leo Blöcher, Jan Rüth, Torsten Zimmermann, and
Oliver Hohlfeld[†]

Communication and Distributed Systems, RWTH Aachen University
{kosek, bloecher, rueth, zimmermann}@comsys.rwth-aachen.de

[†]Chair of Computer Networks, Brandenburg University of Technology
oliver.hohlfeld@b-tu.de

Abstract. Standards govern the SHOULD and MUST requirements for protocol implementers for interoperability. In case of TCP that carries the bulk of the Internets' traffic, these requirements are defined in RFCs. While it is known that not all optional features are implemented and non-conformance exists, one would assume that TCP implementations at least conform to the minimum set of MUST requirements. In this paper, we use Internet-wide scans to show how Internet hosts and paths conform to these basic requirements. We uncover a non-negligible set of hosts and paths that do not adhere to even basic requirements. For example, we observe hosts that do not correctly handle checksums and cases of middlebox interference for TCP options. We identify hosts that drop packets when the urgent pointer is set or simply crash. Our publicly available results highlight that conformance to even fundamental protocol requirements should not be taken for granted but instead checked regularly.

1 Introduction

Reliable, interoperable, and secure Internet communication largely depends on the adherence to standards defined in RFCs. These RFCs are simple text documents, and any specifications published within them are inherently informal, flexible, and up for interpretation, despite the usage of keywords indicating the requirement levels [20], e.g., SHOULD or MUST. It is therefore expected and known that violations—and thus non-conformance—do arise unwillingly. Nevertheless, it can be assumed that Internet hosts widely respect at least a minimal set of mandatory requirements. To which degree this is the case is, however, unknown.

In this paper, we shed light on this question by performing Internet-wide active scans to probe if Internet hosts and paths are conformant to a set of *minimum* TCP requirements that any TCP speaker MUST implement. This adherence to the fundamental protocol principles is especially important since TCP carries the bulk of the data transmission in the Internet. The basic requirements of a TCP host are defined in RFC 793 [47]—the core TCP specification. Since its over 40 years of existence, it has accumulated over 25 accepted errata described in RFC 793bis-Draft14 [27], which is a draft of a planned future update of the TCP

specification, incorporating all minor changes and errata to RFC 793. We base our selection of probed requirements on formalized MUST requirements defined in this drafted update to RFC 793.

The relevance of TCP in the Internet is reflected in the number of studies assessing its properties and conformance. Well studied are the interoperability of TCP extensions [21], or within special purpose scenarios [40, 41], and especially non-conformance introduced by middleboxes on the path [24, 35]. However, the conformance to basic mandatory TCP features has not been studied in the wild. We close this gap by studying to which degree TCP implementations in the wild conform to MUST requirements. Non-conformance to these requirements limits interoperability, extensibility, performance, or security properties, leading to the essential necessity to understand who does not adhere to which level of non-conformance. Uncovering occurrences of non-conformities hence reveal areas of improvement for future standards. A recent example is QUIC, where effort is put into the avoidance of such misconceptions during standardization [46].

With our large scale measurement campaign presented in this paper, we show that while the majority of end-to-end connections are indeed conforming to the tested requirements, a non-trivial number of end-hosts as well as end-to-end paths show non-conformities, breaking current and future TCP extensions, and even voiding interoperability thus reducing connectivity. We show that

- ▶ In a controlled lab study, non-conformance already exists at the OS-level: only two tested stacks (Linux and lwIP) pass all tests, where, surprisingly, others (including macOS and Windows) fail in at least one category each. Observing non-conformance in the wild can therefore be expected.
- ▶ In the wild, we indeed found a non-negligible amount of non-conformant hosts. For example, checksums are not verified in ~3.5% of cases, and middleboxes inject non-conformant MSS values. Worrisome, using reserved flags or setting the urgent pointer can render the target host unreachable.
- ▶ At a infrastructure level, 4.8% of the Alexa domains with and without `www.` prefix show different conformance levels (e.g., because of different infrastructures: CDN vs. origin server), mostly due to flags that limit reachability. The reachability of websites can thus depend on the `www.` prefix.

Structure. In Section 2 we present related work followed by our methodology and its validation in Section 3. The design and evaluation of our Internet-wide TCP conformance scans are discussed in Section 4 before we conclude the paper.

2 Related Work

Multiple measurement approaches have focused on the conformance of TCP implementations on servers, the presence of middleboxes and their interference on TCP connections, and non-standard conform behavior. In the following, we discuss similarities and differences of selected approaches to our work.

TCP Stack Behavior. One line of research aims at characterizing remote TCP stacks by their behavior (e.g., realized in the TCP Behavior Inference Tool

(TBIT) [45] in 2001). One aspect is to study the deployment of TCP tunings (e.g., the initial window configuration [48–50]) or TCP extensions (e.g., Fast Retransmit [45], Fast Open [39,44], Selective Acknowledgment (SACK) [36,43,45], or Explicit Congestion Notification (ECN) [18,36,37,42,43,45] and ECN++ [38] to name a few). While these works aim to generally characterize stacks by behavior and to study the availability and deployability of TCP extensions, our work specifically focuses on the *conformance* of *current* TCP stacks to *mandatory* behavior every stack must implement. A second aspect concerns the usage of behavioral characterizations to *fingerprint* TCP stacks (e.g., via active [30] or passive [19] measurements) and mechanisms to defeat fingerprinting (e.g., [53]).

Middlebox Interference. The end-to-end behavior of TCP not only depends on the stack implementations, but also on on-path middleboxes [22], which can tune TCP performance and security but also (negatively) impact protocol mechanisms and extensions (see e.g., [18,42,43]). Given their relevance, a large body of work studies the impact within the last two decades and opens the question if TCP is still extensible in today’s Internet. Answering this question resulted in a methodology for middlebox inference which is extended by multiple works to provide middlebox detection tools to assess their influence; By observing the differences between sent and received TCP options at controlled endpoints (TCPEXPOSURE [32]), it is observed that 25% of the studied paths tamper with TCP options, e.g., with TCP’s SACK mechanism. Similarly, `tracebox` [24] also identifies middleboxes based on modifications of TCP options, but as client-side only approach without requiring server control. Besides also identifying the issues with TCP’s SACK option, they highlight the interference with TCP’s MSS option and the incorrect behavior of TCP implementations when probing for MPTCP support. `PATHSpider` [35] extends `tracebox` to test more TCP options, e.g., ECN or differentiated services code point (DSCP). They evaluate their tool in an ECN support study, highlighting that some intermediaries tamper with the respective options, making a global ECN deployment a challenging task. Further investigating how middleboxes harm TCP traffic, a `tracebox`-based study [28] shows that more than a third of all studied paths cross at least one middlebox, and that on over 6% of these paths TCP traffic is harmed. Given the negative influence of transparent middleboxes, proposals enable endpoints to identify and negotiate with middleboxes using a new TCP option [34] and to generally cooperate with middleboxes [23]. While we focus on assessing TCP conformance to mandatory behavior, we follow `tracebox`’s approach to differentiate non-conforming stacks from middlebox interference causing non-conformity.

Takeaway: *While a large body of work already investigates TCP behavior and middlebox inference, a focus on conformance to mandatory functionality required to implement is missing—a gap that we address in this study.*

3 Methodology

We test TCP conformance by performing active measurements that probe for mandatory TCP features and check adherence to the RFC. We begin by explaining

how we detect middleboxes before we define the test cases and then validate our methodology in controlled testbed experiments.

3.1 Middlebox Detection

Middleboxes can alter TCP header information and thereby cause non-conformance, which we would wrongly attribute to the probed host without performing a middlebox detection. Therefore, we use the tracebox approach [24] to detect interfering middleboxes by sending and repeating our probes with increasing IP TTLs. That is, in every test case (see Section 3.2), the first segment is sent multiple times with increasing TTL values from 1 to 30 in parallel while capturing ICMP time exceeded messages. We limit the TTL to 30 since we did not observe higher hop counts in our prior work for Internet-wide scans [51]. To distinguish the replied messages and determine the hop count, we encode the TTL in the IPv4 ID and in the TCP acknowledgment number, window size, and urgent pointer fields. We chose to encode the TTL in multiple header fields since middleboxes could alter every single one. These repetitions enable us to pinpoint and detect (non-)conformance within the end-to-end path if ICMP messages are issued by the intermediaries quoting the expired segment. Please note that alteration or removal of some of our encodings does *not* render the path or the specific hop non-conformant. A non-conformance is only attested, if the actual tested behavior was modified as visible through the expired segment. Further, since only parts of the fields—all 16 or 32 bits in size—may be altered by middleboxes (e.g., slight changes to the window size), we repeat each value as often as possible within every field. Our TTL value of at most 30 can be encoded in 5 bits, and thus be repeated 3 to 6 times in the selected fields. Additionally, the TCP header option No-Operation (NOOP) allows an opaque encoding of the TTL. Specifically, we append as many NOOPs as there are hops in the TTL to the fixed-size header. Other header fields are either utilized for routing decisions (e.g., port numbers in load balancers) or are not opaque (e.g., sequence numbers), rendering them unsuitable. Depending on the specific test case, some of the fields are not used for the TTL encoding. For example, when testing for urgent pointer adherence, we do not encode the TTL in the urgent pointer field.

3.2 TCP Conformance Test Cases

Our test cases check for observable TCP conformance of end-to-end connections by actively probing for a set of *minimum* requirements that any TCP must implement. We base our selection on 119 explicitly numbered requirements specified in RFC 793bis-Draft14 [27], of which 69 are absolute requirements (i.e., *MUSTs* [20]). These *MUSTs* resemble minimum requirements for *any* TCP connection participating in the Internet—not only for hosts, but also for intermediate elements within the traversed path. The majority of these 69 *MUSTs* address internal state-handling details, and can therefore not be observed or verified via active probing. To enable an Internet-wide assessment of TCP conformance, we thus focus on *MUST* requirements whose adherence is

Checksum	PASS Condition
<i>ChecksumIncorrect</i> (2,3)	▶ When sending a SYN or an ACK segment with a non-zero but invalid checksum, a target must respond with a RST segment or ignore it
<i>ChecksumZero</i> (2,3)	▶ As above but with an explicit zeroed checksum
Options	PASS Condition
<i>OptionSupport</i> (4)	▶ When sending a SYN segment with EOL and NOOP options, a target must respond with a SYN/ACK segment
<i>OptionUnknown</i> (6)	▶ When sending a SYN segment with an unassigned option (# 158), a target must respond with a SYN/ACK segment
<i>MSSSupport</i> (4,14,16)	▶ When sending a SYN segment with an MSS of 515 byte, a target must not send segments exceeding 515 byte
<i>MSSMissing</i> (15,16)	▶ When sending a SYN segment without an MSS, a target must not send segments exceeding 536 byte (IPv4) or 1220 byte (IPv6, not tested)
Flags	PASS Condition
<i>Reserved</i> (no MUST)	▶ When Sending a SYN segment with a reserved flag set (# 2), a target must respond with a SYN/ACK segment with zeroed reserved flags ▶ Subsequently, when sending an ACK segment with a reserved flag set (# 2), a target must not retransmit the SYN/ACK segment
<i>UrgentPointer</i> (30,31)	▶ When sending a sequence of segments flagged as urgent, a target must acknowledge them with an ACK segment

Table 1. Requirements based on the MUSTs (number from RFC shown in brackets) as defined in RFC 793bis, Draft 14 [27]. Further, we show the precise test sequence and the condition leading to a PASS for the test.

observable by communicating with the remote host. We synthesize eight tests from these requirements, which we summarize in Table 1, and discuss them in the following paragraphs. Each test is in some way critical to interoperability, security, performance, or extensibility of TCP. The complexity involved in verifying conformance to other advanced requirements often leads to the exclusion of these seemingly fundamental properties in favor of more specialized research.

TCP Checksum. The TCP checksum protects against segment corruption in transit and is mandatory to both calculate and verify. Even though most Layer 2 protocols already protect against segment corruption, it has been shown [55] that software or hardware bugs in intermediate systems may still alter packet data, and thus, high layer checksums are still vital. Checksums are an essential requirement to consider due to the performance implications of having to iterate over the entire segment after receiving it, resulting in an incentive to skip this step even though today this task is typically offloaded to the NIC. Both the *ChecksumIncorrect* and the *ChecksumZero* test (see Table 1) verify the handling of checksums in the TCP header. They differ only in the kind of checksum used; the former employs a randomly chosen incorrect checksum while the latter, posing

as a special case, zeroes the field instead, i.e., this could appear as if the field is unused.

TCP Options. TCP specifies up to 40 bytes of options for future extensibility. It is thus crucial that these bytes are actually usable and, if used, handled correctly. According to the specification, any implementation is required to support the EOOL, NOOP, and MSS option. We test these options due to their significance for interoperability and, in the general case, extensibility and performance. The different, and sometimes variable, option length makes header parsing somewhat computationally expensive (especially in hardware), opening the door for non-conformant performance enhancements comparable to skipping checksum verification. Further, an erroneous implementation of either requirement can have security repercussions in the form of buffer overflows or resource wastage, culminating in a denial of service. The *OptionSupport* test validates the support of EOOL and NOOP, while the *OptionUnknown* test checks the handling of an unassigned option. The *MSSSupport* test verifies the proper handling of an explicitly stated MSS value, while the *MSSMissing* test tests the usage of default values specified by the RFC in the absence of the MSS option.

TCP Flags. Alongside the stated TCP options, TCP’s extensibility is mainly guaranteed by (im-)mutable control flags in its header, of which four are currently still reserved for future use. The most prominent “recent” example is ECN [29], which uses two previously reserved bits. Though not explicitly stated as a numbered formal MUST¹, a TCP must zero (when sending) and ignore (when receiving) unknown header flags, which we test with the *Reserved* test, as incorrect handling can considerably block or delay the adoption of future features.

The *UrgentPointer* test addresses the long-established URG flag. Validating the support of segments flagged as urgent, the test splits around 500 bytes of urgent data into a sequence of three segments with comparable sizes. Each segment is flagged as urgent, and the urgent pointer field carries the offset from its current sequence number to the sequence number following the urgent data, i.e., to the sequence number following the payload. Initially intended to speed up segment processing by indicating data which should be processed immediately, the widely-used Berkeley Software Distribution (BSD) socket interface instead opted to interpret the urgent data as out-of-band data, leading to diverging implementations. As a result, the urgent pointer’s usage is discouraged for new applications [27]. Nevertheless, TCP implementations are still required to support it with data of arbitrary length. As the requirement’s inclusion adds computational complexity, implementers may see an incentive to skip it.

Pass and Failure Condition Notation. For the remainder of this paper, we use the following notation to report passing or failing of the above-described tests. Connections that unmistakably conform are denoted as *PASS*, whereas not clearly determinable results (applies only to some tests) are conservatively stated as *UNK*. UNKs may have several reasons such as, e.g., hosts ceasing to respond

¹ RFC 793bis-Draft14 states: “Must be zero in generated segments and must be ignored in received segments, if corresponding future features are unimplemented by the sending or receiving host.” [27]

MUST Test as defined in Table 1	Linux 5.2.10	Windows 1809	macOS 10.14.6	uIP 1.0	lwIP 2.1.2	Seastar 19.06
<i>ChecksumIncorrect</i>	✓	✓	✓	✓	✓	✗
<i>ChecksumZero</i>	✓	✓	✓	✓	✓	✗
<i>OptionSupport</i>	✓	✓	✓	✓	✓	✓
<i>OptionUnknown</i>	✓	✓	✓	✓	✓	✓
<i>MSSSupport</i>	✓	✗	✓	✓	✓	✓
<i>MSSMissing</i>	✓	✓	✗	✓	✓	✓
<i>Reserved</i>	✓	✓	✓	✓	✓	✓
<i>UrgentPointer</i>	✓	✓	✓	✗	✓	✓

Table 2. Results of testbed measurements stating PASS (✓) and F_{Target} (✗)

to non-test packets after having responded to a liveness test. Non-conformities raised by the target host are denoted as F_{Target} , and non-conformities raised by middleboxes on the path rather than the probed host are denoted as F_{Path} .

3.3 Validation

To evaluate our test design, we performed controlled measurements using a testbed setup, thereby eliminating possible on-path middlebox interference. Thus, only F_{Target} can occur in this validation, but not F_{Path} . To cover a broad range of hosts, we verified our test implementations by targeting current versions of the three dominant Operating Systems (OSs) (Linux, Windows, and macOS) as well as three alternative TCP stacks (uIP [13], lwIP [7], and Seastar [8]).

We summarize the results in Table 2. As expected, we observe a considerable degree of conformance. Linux, as well as lwIP, managed to achieve full conformance to the tested requirements. Surprisingly, all other stacks failed in at least one test each. That is, most stacks do not fully adhere to these minimum requirements. uIP exposed the most critical flaw by crashing when receiving a segment with urgent data, caused by a segmentation fault while attempting to read beyond the segment’s size (see Section 3.2). Since the release of the tested Version of uIP, the project did not undergo further development, but instead moved to the Contiki OS project [3], where it is currently maintained in Contiki-NG [2]. Following up on Contiki, it was uncovered that both distributions are still vulnerable. Their intended deployment platform, embedded microcontrollers, often lack the memory access controls present in modern OSs, amplifying the risk that this flaw poses. Addressing this issue, we submitted a Pull request to Contiki-NG [1]. The remaining F_{Target} have much less severe repercussions. Seastar, which bypasses the Linux L4 network stack using *Virtio* [15], fails both checksum tests. While hardware offloading is enabled by default, Seastar features software checksumming, which should take over if offloading is disabled or unsupported by the host OS. However, host OS support of offloaded features is not verified, which can lead to mismatches between believed to be and actually enabled features. We reported this issue to the authors [9]. The tests pass if the unsupported hardware offloads are manually deselected. The F_{Target} failure for macOS in the *MSSMissing* test is a consequence of macOS defaulting to a 1024 bytes MSS regardless of the IP

version, thereby exceeding the IPv4 TCP default MSS, and falling behind that of IPv6. Windows 10 applies the MSS defaults defined in the TCP specification as a lower bound to any incoming value, overwriting the 515 bytes advertised in the *MSSSupport* test. Both MSS non-conformities could be mitigated by path maximum transmission unit (MTU) discovery, dynamically adjusting the segment size to the real network path.

Takeaway: *Only two tested stacks (Linux and lwIP) pass all tests and show full conformance. Surprisingly, all other stacks failed in at least one category each. That is, non-conformance to basic mandatory TCP implementation requirements already exists in current OS implementations. Even though our testbed validation is limited in the OS diversity, we can already expect to find a certain level of host non-conformance when probing TCP implementations in the wild.*

4 TCP Conformance in the Wild

In the following, we move on from our controlled testbed evaluation and present our measurement study in the Internet. Before we present and discuss the obtained results, we briefly focus on our measurement setup and our selected target sets.

4.1 Measurement Setup & Target Hosts

Measurement Setup. Our approach involves performing active probes against target hosts in the Internet to obtain a representative picture of TCP conformance in the wild. All measurements were performed using a single vantage point within the IPv4 research network of our university between August 13 and 22, 2019. As we currently do not have IPv6-capable scan infrastructure at our disposal, we leave this investigation open for future work. Using a probing rate of 10k pps on a distinct 10Gbit/s uplink, we decided to omit explicit loss detection and retransmission handling due to the increased complexity, instead stating results possibly affected by loss as UNK if not clearly determinable otherwise.

Target Hosts. To investigate a diverse set of end-to-end paths as well as end hosts, a total of 3,731,566 targets have been aggregated from three sources: *i*) the HTTP Archive [33], *ii*) Alexa Internet’s top one million most visited sites list [17, 52], and *iii*) Censys [25] port 80 and 443 scans.

The **HTTP Archive** regularly crawls about 5M domains obtained from the Chrome User Experience Report to study Web performance and publishes the resulting dataset. We use the dataset of July 2019. For this, we were especially interested in the Content Delivery Network (CDN) tagged URLs, as no other source provides URL-to-CDN mappings. Since no IP addresses are provided, we resolved the 876,835 URLs to IPv4 addresses through four different public DNS services of Cloudflare, Google, DNS.WATCH, and Cisco’s OpenDNS. Some domains contain multiple CDN tags in the original dataset. For these cases, we obtained the CDN mapping from the chain of CNAME resource records in the DNS responses and excluded targets that could still not be linked to only a single CDN. Removing duplicates on a per-URL basis, one target per resolved IPv4

address was selected. The resulting 4,116,937 targets were sampled to at most 10,000 entries per CDN, leading to 147,318 hosts in total. Removing duplicate IP addresses and blacklist filtering, we derived the final set of 27,795 CDN targets.

As recent research has shown [16], prefixing `www.` to a domain might not only provide different TLS security configurations and certificates than their non-`www` counterparts, but might also (re-)direct the request to servers of different Content Providers (CPs). To study this implications on TCP conformance, we used the **Alexa 1M list** published on August 10th, 2019, and resolved every domain with and without `www`-prefix according to the process outlined in the HTTP Archive. The resulting 3,297,849 targets were further sampled, randomly selecting one target with and without `www`-prefix per domain, removing duplicate IP addresses and blacklist filtering, leading to 466,685 Alexa targets.

Censys provided us research access to their data of Internet-wide port scans, which represent a heterogeneous set of globally distributed targets. In addition to the IPv4 address and successfully scanned port, many targets include information on host, vendor, OS, and product. Using the dataset compiled on August 8th, 2019, 10,559,985 Censys targets were identified with reachable ports 80 or 443, including, but not limited to, IoT devices, customer-premises equipment, industrial control systems, remote-control interfaces, and network infrastructure appliances. By removing duplicate IP addresses and blacklist filtering we arrive at 3,237,086 Censys target hosts.

Ethical Considerations. We aim to minimize the impact of our active scans as much as possible. First, we follow standard approaches [26] to display the intent of our scans in rDNS records of our scan IPs and on a website with an opt-out mechanism reachable via each scan IP. Moreover, we honor opt-out requests to our previous measurements and exclude these hosts. We further evaluated the potential implications of the uIP/Contiki crash observed in Section 3.3. Embedded microcontrollers, commonly used in IoT devices, are the primary use-case of uIP/Contiki. We could not identify hosts using this stack in the Censys device type data to exclude IPs, but assume little to very little use of this software stack within our datasets. We thus believe the potential implications to be minimal. We confirm this by observing that 100% of failed targets in the CDN as well as the Alexa dataset, and 99.35% of failed targets in the Censys dataset, are still reachable following *UrgentPointer* test case execution. We thus argue that our scans have created no harm to the Internet at large.

4.2 Results and Discussion

We next discuss the results of our conformance testing, which we summarize in Table 3. The table shows the relative results per test case for all reachable target hosts, excluding the unreachable ones. As the target data was derived from the respective sources multiple days before executing the tests (see Section 4.1), unreachable targets are expected. Except for minor variations, which can be explained by dynamic IP address assignment and changes to host configurations during test execution, ~12% of targets could not be reached in each test case and are removed from the results. While the CDN and Alexa datasets were derived

MUST Test as defined in Table 1	CDN $n = 27,795$			Alexa $n = 466,685$			Censys $n = 3,237,086$		
	UNK	F_{Target}	F_{Path}	UNK	F_{Target}	F_{Path}	UNK	F_{Target}	F_{Path}
<i>ChecksumIncorrect</i>	0.234	0.374	-	0.441	3.224	0.002	3.743	3.594	0.003
<i>ChecksumZero</i>	0.253	0.377	-	0.455	3.210	0.001	3.873	3.592	0.003
<i>OptionSupport</i>	-	0.040	-	-	0.470	0.009	-	1.410	0.313
<i>OptionUnknown</i>	-	0.026	0.011	-	0.585	0.053	-	1.477	0.019
<i>MSSSupport</i>	-	0.018	-	-	0.728	0.002	-	0.412	0.004
<i>MSSMissing</i>	0.026	-	0.018	0.303	0.299	0.136	1.423	0.388	0.416
<i>Reserved</i>	-	2.194	0.011	-	6.689	0.293	-	2.791	0.048
<i>Reserved-SYN</i>	-	0.138	0.011	-	1.297	0.309	-	1.849	0.049
<i>UrgentPointer</i>	0.150	0.330	0.022	0.804	3.179	0.208	3.815	7.300	0.042

Table 3. Overview of relative results (in %) per test case per dataset. Here, n denotes the number of targets in each dataset. For better readability, we do not show the PASS results and highlight excessive failure rates in bold.

from sources featuring popular websites, we expect a large overlap of target hosts, which is confirmed by 15,387 targets present in both datasets. Alexa and Censys share only 246 target hosts, while CDN and Censys do not overlap. All datasets are publicly available [5]. The decision to classify a condition as PASS, UNK, F_{Target} , or F_{Path} , does vary between test cases as a result of their architecture (see Section 3.2) and are discussed in detail next.

TCP Checksum. We start with the results of our checksum tests that validate correct checksum handling. As Table 3 shows, CDNs have a low failure rate for both tests, and we do not find any evidence for on-path modifications. In contrast, hosts from the Alexa and the Censys dataset show over $\sim 3\%$ F_{Target} failures. Drilling down on these hosts, they naturally cluster into two classes when looking at the AS ownership. On the one hand, we find AS (e.g., Amazon), where roughly 7% of all hosts fail both tests. Given the low share, these hosts could be purpose build high-performance VMs, e.g., for TCP-terminating proxies that do not handle checksums correctly. On the other hand, we find hosts (e.g., hosted in the QRATOR filtering AS) where nearly all hosts in that AS fail the tests. Since QRATOR offers a DDoS protection service, it is a likely candidate for operating a special purpose stack.

Takeaway: *We find cases of hosts that do not correctly handle checksums. While incorrect checksums may be a niche problem in the wild, these findings highlight that attackers with access to the unencrypted payload, but without access to the headers, could alter segments and have the modified data accepted.*

TCP Options. We next study if future TCP extensibility is honored by the ability to use TCP options. In our four option tests (see Table 3 for an overview), we observe overall the lowest failure rates—a generally good sign for extensibility support. Again, the Censys dataset shows the most failures, and especially the *OptionSupport* and the *MSSMissing* test have the highest F_{Path} (middlebox failures) across all tests. Both tests show a large overlap in the affected hosts and have likely the same cause for the high path failure rates. We observe that these hosts are all located in ISP networks. For the *MSSMissing* failures, we observe

that an MSS is inserted at these hosts—likely due to the ISPs performing MSS clamping, e.g., due to PPPoE encapsulation by access routers. These routers need to rewrite the options header (to include the MSS option), and as the *OptionSupport* fails when, e.g., some of the EOOL and NOOP are stripped, the exact number of EOOL and NOOP are likely not preserved. Still, inserting the MSS option alters the originally intended behavior of the sender, i.e., having an MSS of 536 byte for IPv4. In this special case, the clamping did actually increase the MSS, and thereby strip some of the EOOL and NOOP options.

Looking at the *OptionUnknown* test, where we send an option with an unallocated codepoint, we again see low F_{Path} failures, but still, a non-negligible number of F_{Target} fails. There is no single AS that stands out in terms of the share of hosts that fail this test. However, we observe that among the ASes with the highest failure rates are ISPs and companies operating Cable networks.

Lastly, the *MSSSupport* test validating the correct handling of MSS values shows comparably high conformance. As we were unable to clearly pinpoint the failures to specific ASes, the most likely cause can be traced to the non-conformant operating systems as shown by our validation (see Section 3.3), where Windows fails this test and likely others that we did not test in isolation.

Takeaway: *Our TCP options tests show the highest level of conformance of all tests, a good sign for extensibility. Still, we find cases of middlebox inference, mostly MSS injectors and option padding removers—primarily in ISP networks hinting at home gateways. Neither is inherently harmful due to path MTU discovery and the voluntary nature of option padding.*

TCP Flags. Besides the previously tested options, TCPs extensibility is mainly guaranteed by (im-)mutable control flags in its header to toggle certain protocol behavior. In the *Reserved* test, we identify the correct handling of *unknown* (future) flags by sending an unallocated flag and expect no change in behavior. Instead, we surprisingly observe high failure rates across all datasets, most notable CDNs. When inspecting the CDN dataset, we found ~10% of Akamai’s hosts to show this behavior. We contacted Akamai, but they validated that their servers do *not* touch this bit. Further analysis revealed that the reserved flag on the SYN was truthfully ignored, but *our test* failed as the final ACK of the 3-way handshake (second part of the test, see Table 1), which also contains the reserved flag, was seemingly dropped as we got SYN/ACK retransmissions. However, this behavior originates from the usage of Linux’s *TCP_DEFER_ACCEPT* socket option, which causes a socket to only wakeup the user space process if there is data to be processed [10]. The socket will wait for the first data segment for a specified time, re-transmitting the SYN/ACK when the timer expires in the hope of stimulating a retransmission of possibly lost data. Since we were not sending any data, we eventually received a SYN/ACK retransmission, seemingly due to the dropped handshake-completing ACK with the reserved flag set. Hence, we credited the retransmission to the existence of the reserved flag at first, later uncovering that the retransmission was unrelated to the reserved flag, but actually expected behavior using the *TCP_DEFER_ACCEPT* socket option. Following up with Akamai, they were able to validate our assumption by revealing that

parts of their services utilize this socket option. While it is certainly debatable if deliberately ignoring the received ACK is a violation of the TCP specification, our test fails to account for this corner case. Thus, connectivity is *not* impaired.

In contrast, connectivity *is* impaired in the cases where our reserved flag SYN fails to trigger a response at all, leaving the host unreachable (see *Reserved-SYN* in Table 3). The difference between both failure rates thus likely denotes hosts using the defer accept mechanism, as CDNs, in general, seem to comply with the standard. We also observe a significant drop in failures in the Alexa targets. While our results are unable to show if *only* defer accepts are the reason for this drop, they likely contribute significantly as TCP implementations would need to differentiate between a reserved flag on a SYN and on an ACK, which we believe is less likely. Our results motivate a more focused investigation of the use of socket options and the resulting protocol configurations and behavioral changes.

Lastly, the URG flag is part of TCP since the beginning to indicate data segments to be processed immediately. With the *UrgentPointer* test we check if segments that are flagged as urgent are correctly received and acknowledged. To confirm our assumption of this test having minimal implications on hosts due to the uIP/Contiki crash (see Section 3.3), we checked if the F_{Target} instances were still reachable after test execution. Our results show that of these failed targets, 99.35% of Censys, and 100% of CDN and Alexa, did respond to our following connection requests, which were part of the subsequent test case executed several hours later. While we argue that these unresponsive hosts can be explained by dynamic IP address assignment due to the fluctuating nature of targets in the Censys dataset, we recognize that the implicit check within the subsequent test case is problematic due to the time period between the tests and the possibility of devices and services being (automatically) restarted after crashing. We thus posit, that future research should include explicit connectivity checks directly following test case execution on a per target basis, and skip subsequent tests if a target’s connectivity is impaired.

Surprisingly, the *UrgentPointer* test shows the highest failure rate among all tests. That is, segments flagged as urgent are *not correctly* processed. In other words, flagging data as urgent limits connectivity. We find over ~7% of hosts failing in the Censys dataset, where ISPs again dominate the ranking. Only about 1.2% of these failures actively terminated the connection with a RST, while the vast majority silently discarded the data without acknowledging it. Looking at Alexa and CDNs, we again find an Amazon AS at the top. Here, we randomly sampled the failed hosts to investigate the kind of services offered by them. At the top of the list, we discovered services that were proxied by a *Vegur* [14], respective *Cowboy* [4], proxy server that seem to be used in tandem with the *Heroku* [6] cloud platform. Even though we were unable to find how Heroku precisely operates, we suspect a high-performance implementation that might simply not implement the urgent mechanism at all.

Takeaway: *While unknown flags are often correctly handled, they can reduce reachability, especially when set on SYNs. The use of the urgent pointer resulted in the highest observed failure rate by hosts that do not process data segments*

flagged as urgent. Thus, using the reserved flags or setting the urgent pointer limits connectivity in the Internet.

We therefore posit to remove the mandatory implementation requirement of the urgent pointer from the RFC to reflect its deprecation status, and thus explicitly state that its usage can break connectivity. Future protocol standards should therefore be accompanied by detailed socket interface specifications, e.g., as has been done for IPv6 [31, 54], to avoid RFC misconceptions. Moreover, we started a discussion within the IETF, addressing the issue encountered with the missing formal MUST requirement of unknown flags, which potentially led and/or will lead to diverging implementations [11]. Additionally, we proposed a new MUST requirement, removing ambiguities in the context of future recommended, but not required, TCP extensions which allocate reserved bits [12].

Alexa: Does `www.` matter? It is known that `www.domain.tld` and `domain.tld` can map to different hosts [16], e.g., the CDN host vs. the origin server, where it is often implicitly assumed that both addresses exhibit the same behavior. However, 4.89% (11.4k) of the Alexa domains with and without `www.` prefix show different conformance levels to at least one test. That is, while the host with the `www.` prefix can be conformant, the non-prefixed host could not, and vice versa. Most of these non-conformance issues are caused by TCP flags, for which we have seen that they can impact the reachability of the host. That is, 53.3% of these domains failed the reserved flags test, and 58% the urgent pointer test (domains can be in both sets). Thus, a website can be unreachable using one version and reachable by the other.

Takeaway: *While the majority of Alexa domains are conformant, the ability to reach a website can differ whether or not the `www.` prefix is used.*

5 Conclusion

This paper presents a broad assessment of TCP conformance to mandatory MUST requirements. We uncover a non-negligible set of Internet hosts and paths that do not adhere to even basic requirements. Non-conformance already exists at the OS-level, which we uncover in controlled testbed evaluations: only two tested stacks (Linux and lwIP) pass all tests. Surprisingly, others (including macOS and Windows) fail in at least one category each. A certain level of non-conformance is therefore expected in the Internet and highlighted by our active scans. First, we observe hosts that do not correctly handle checksums. Second, while TCP options show the highest level of conformance, we still find cases of middlebox inference, mostly MSS injectors and option padding removers—primarily in ISP networks hinting at home gateways. Moreover, and most worrisome, using reserved flags or setting the urgent pointer can render the target host unreachable. Last, we observe that 4.8% of Alexa-listed domains show different conformance levels when the `www.` prefix is used, or not, of which more than 50% can be attributed to TCP flag issues—which can prevent connectivity. Our results highlight that conformance to even fundamental protocol requirements should not be taken for granted but instead checked regularly.

Acknowledgments

This work has been funded by the DFG as part of the CRC 1053 MAKI within subproject B1. We would like to thank Akamai Technologies for feedback on our measurements, Censys for contributing active scan data, and our shepherd Robert Beverly and the anonymous reviewers.

References

1. Contiki-NG TCP URG Pull Request. <https://github.com/contiki-ng/contiki-ng/pull/1173>
2. Contiki-NG: The OS for Next Generation IoT Devices. <https://github.com/contiki-ng>
3. Contiki OS. <https://github.com/contiki-os>
4. Cowboyku, <https://github.com/heroku/cowboyku>
5. Dataset to "MUST, SHOULD, DON'T CARE: TCP Conformance in the Wild". <https://doi.org/10.18154/RWTH-2020-00809>
6. Heroku platform, <https://www.heroku.com/>
7. lwIP - A Lightweight TCP/IP stack. <http://savannah.nongnu.org/projects/lwip/>
8. Seastar. <https://github.com/scylladb/seastar>
9. Seastar: Virtio device reports features not supported by the OS. <https://github.com/scylladb/seastar/issues/719>
10. tcp(7) - linux man page, <https://linux.die.net/man/7/tcp>
11. TCPM Mailinglist: RFC793bis draft 14 Reserved Bits: Problem statement. https://mailarchive.ietf.org/arch/msg/tcpm/s0LtY3Ce3QBBAkJ_DuSH5VDNFM
12. TCPM Mailinglist: RFC793bis draft 14 Reserved Bits: Proposal. https://mailarchive.ietf.org/arch/msg/tcpm/_jpUQx0AjByR3U0gyX88RwoTxL0
13. uIP. <https://github.com/adamdunkels/uiP>
14. Vegur: Http proxy library, <https://github.com/heroku/vegur>
15. Virtio: Paravirtualized drivers for kvm/Linux. <https://www.linux-kvm.org/page/Virtio>
16. Alashwali, E.S., Szalachowski, P., Martin, A.: Does "www." Mean Better Transport Layer Security? In: ACM International Conference on Availability, Reliability and Security (ARES) (2019). <https://doi.org/10.1145/3339252.3339277>
17. Alexa Internet: About us, <https://www.alexa.com/about>
18. Bauer, S., Beverly, R., Berger, A.: Measuring the state of ECN readiness in servers, clients, and routers. In: ACM Internet Measurement Conference (IMC) (2011). <https://doi.org/10.1145/2068816.2068833>
19. Beverly, R.: A Robust Classifier for Passive TCP/IP Fingerprinting. In: Passive and Active Measurement Conference (PAM) (2004). https://doi.org/10.1007/978-3-540-24668-8_16
20. Bradner, S.O.: Key words for use in RFCs to Indicate Requirement Levels. RFC 2119 (Mar 1997). <https://doi.org/10.17487/RFC2119>
21. Cardwell, N., Cheng, Y., Brakmo, L., Mathis, M., Raghavan, B., Dukkupati, N., Keng Jerry Chu, H., Terzis, A., Herbert, T.: packetdrill: Scriptable Network Stack Testing, from Sockets to Packets. In: USENIX Annual Technical Conference (ATC) (2013), <https://www.usenix.org/conference/atc13/technical-sessions/presentation/cardwell>

22. Carpenter, B., Brim, S.: Middleboxes: Taxonomy and issues (2002). <https://doi.org/10.17487/RFC3234>
23. Craven, R., Beverly, R., Allman, M.: A middlebox-cooperative TCP for a non end-to-end internet. In: ACM SIGCOMM (2014). <https://doi.org/10.1145/2619239.2626321>
24. Detal, G., Hesmans, B., Bonaventure, O., Vanaubel, Y., Donnet, B.: Revealing Middlebox Interference with Tracebox. In: ACM Internet Measurement Conference (IMC) (2013). <https://doi.org/10.1145/2504730.2504757>
25. Durumeric, Z., Adrian, D., Mirian, A., Bailey, M., Halderman, J.A.: A Search Engine Backed by Internet-Wide Scanning. In: ACM Conference on Computer and Communications Security (CCS) (2015). <https://doi.org/10.1145/2810103.2813703>
26. Durumeric, Z., Wustrow, E., Halderman, J.A.: ZMap: Fast Internet-wide Scanning and Its Security Applications. In: USENIX Security Symposium (2013), <https://www.usenix.org/conference/usenixsecurity13/technical-sessions/paper/durumeric>
27. Eddy, W.: Transmission Control Protocol Specification. Internet-Draft draft-ietf-tcpm-rfc793bis-14, Internet Engineering Task Force (Jul 2019), <https://datatracker.ietf.org/doc/html/draft-ietf-tcpm-rfc793bis-14>, work in Progress
28. Edeline, K., Donnet, B.: A Bottom-Up Investigation of the Transport-Layer Ossification. In: Network Traffic Measurement and Analysis Conference (TMA) (2019). <https://doi.org/10.23919/TMA.2019.8784690>
29. Floyd, S., Ramakrishnan, D.K.K., Black, D.L.: The Addition of Explicit Congestion Notification (ECN) to IP. RFC 3168 (Sep 2001). <https://doi.org/10.17487/RFC3168>
30. Fyodor: Remote os detection via tcp/ip stack fingerprinting. <https://nmap.org/nmap-fingerprinting-article.txt> (1998)
31. Gilligan, R.E., McCann, J., Bound, J., Thomson, S.: Basic Socket Interface Extensions for IPv6. RFC 3493 (Mar 2003). <https://doi.org/10.17487/RFC3493>
32. Honda, M., Nishida, Y., Raiciu, C., Greenhalgh, A., Handley, M., Tokuda, H.: Is It Still Possible to Extend TCP? In: ACM Internet Measurement Conference (IMC) (2011). <https://doi.org/10.1145/2068816.2068834>
33. HTTP Archive: About HTTP Archive, <https://httparchive.org/about>
34. Knutsen, A., Ramaiah, A., Ramasamy, A.: Tcp option for transparent middlebox negotiation. <https://tools.ietf.org/html/draft-ananth-middisc-tcptopt-02> (2013)
35. Kühlewind, M., Walter, M., Learmonth, I.R., Trammell, B.: Tracing Internet Path Transparency. In: Network Traffic Measurement and Analysis Conference (TMA) (2018). <https://doi.org/10.23919/TMA.2018.8506532>
36. Khlewind, M., Neuner, S., Trammell, B.: On the state of ECN and TCP options on the internet. In: Passive and Active Measurement Conference (PAM) (2013). https://doi.org/10.1007/978-3-642-36516-4_14
37. Langley, A.: Probing the viability of TCP extensions. <http://www.imperialviolnet.org/binary/ecntest.pdf> (2008)
38. Mandalari, A.M., Lutu, A., Briscoe, B., Bagnulo, M., Alay, O.: Measuring ECN++: Good News for ++, Bad News for ECN over Mobile. IEEE Communications Magazine **56**(3), 180–186 (March 2018). <https://doi.org/10.1109/MCOM.2018.1700739>
39. Mandalari, A.M., Bagnulo, M., Lutu, A.: TCP Fast Open: initial measurements. In: ACM CoNEXT Student Workshop (2015)
40. Marinos, I., Watson, R.N., Handley, M.: Network Stack Specialization for Performance. In: ACM SIGCOMM (2014). <https://doi.org/10.1145/2619239.2626311>
41. Marinos, I., Watson, R.N., Handley, M., Stewart, R.R.: Disk, Crypt, Net: Rethinking the Stack for High-performance Video Streaming. In: ACM SIGCOMM (2017). <https://doi.org/10.1145/3098822.3098844>

42. Medina, A., Allman, M., Floyd, S.: Measuring Interactions between Transport Protocols and Middleboxes. In: ACM Internet Measurement Conference (IMC) (2004). <https://doi.org/10.1145/1028788.1028835>
43. Medina, A., Allman, M., Floyd, S.: Measuring the Evolution of Transport Protocols in the Internet. SIGCOMM Comput. Commun. Rev. **35**(2), 3752 (Apr 2005)
44. Paasch, C.: Network support for tcp fast open. Presentation at NANOG 67 (2016)
45. Padhye, J., Floyd, S.: On Inferring TCP Behavior. In: ACM SIGCOMM (2001). <https://doi.org/10.1145/383059.383083>
46. Piraux, M., De Coninck, Q., Bonaventure, O.: Observing the Evolution of QUIC Implementations. In: ACM CoNEXT Workshop on the Evolution, Performance, and Interoperability of QUIC (EPIQ) (2018). <https://doi.org/10.1145/3284850.3284852>
47. Postel, J.: Transmission Control Protocol. RFC 793 (Sep 1981). <https://doi.org/10.17487/RFC0793>
48. R uth, J., Hohlfeld, O.: Demystifying TCP Initial Window Configurations of Content Distribution Networks. In: Network Traffic Measurement and Analysis Conference (TMA) (2018). <https://doi.org/10.23919/TMA.2018.8506549>
49. R uth, J., Bormann, C., Hohlfeld, O.: Large-Scale Scanning of TCP’s Initial Window. In: ACM Internet Measurement Conference (IMC) (2017). <https://doi.org/10.1145/3131365.3131370>
50. R uth, J., Kunze, I., Hohlfeld, O.: TCP’s Initial Window - Deployment in the Wild and its Impact on Performance. IEEE Transactions on Network and Service Management (TNSM) (2019). <https://doi.org/10.1109/TNSM.2019.2896335>
51. R uth, J., Zimmermann, T., Hohlfeld, O.: Hidden Treasures — Recycling Large-Scale Internet Measurements to Study the Internet’s Control Plane. In: Passive and Active Measurement Conference (PAM) (2019). https://doi.org/10.1007/978-3-030-15986-3_4
52. Scheitle, Q., Hohlfeld, O., Gamba, J., Jelten, J., Zimmermann, T., Strowes, S.D., Vallina-Rodriguez, N.: A Long Way to the Top: Significance, Structure, and Stability of Internet Top Lists. In: ACM Internet Measurement Conference (IMC) (2018). <https://doi.org/10.1145/3278532.3278574>
53. Smart, M., Malan, G.R., Jahanian, F.: Defeating TCP/IP Stack Fingerprinting. In: USENIX Security Symposium (2000)
54. Stevens, W.R., Thomas, M., Nordmark, E., Jinmei, T.: Advanced Sockets Application Program Interface (API) for IPv6. RFC 3542 (Jun 2003). <https://doi.org/10.17487/RFC3542>
55. Stone, J., Partridge, C.: When the CRC and TCP Checksum Disagree. In: ACM SIGCOMM (2000). <https://doi.org/10.1145/347059.347561>