Outline

Motivation
Existing approaches
Design principles
System architecture
OneProbe
HTTP/OneProbe
Evaluation
Motivation

1. Measuring millions of arbitrary (e2e) paths
   - Active measurement
   - Non-cooperative (active) measurement
Motivation

1. Measuring millions of arbitrary (e2e) paths
   - Active measurement
   - Non-cooperative (active) measurement

2. Reliable measurement
   - Work for all systems and paths
   - Measuring data-path quality
   - Adequate sampling rate and pattern
Motivation

1. Measuring millions of arbitrary (e2e) paths
   - Active measurement
   - Non-cooperative (active) measurement

2. Reliable measurement
   - Work for all systems and paths
   - Measuring data-path quality
   - Adequate sampling rate and pattern

3. Use as many quality metrics as possible.
   - Beyond round-trip metrics
   - More than one to two types of metrics
   - Metrics a function of packet size
Motivation

1. Measuring millions of arbitrary (e2e) paths
   ▶ Active measurement
   ▶ Non-cooperative (active) measurement

2. Reliable measurement
   ▶ Work for all systems and paths
   ▶ Measuring data-path quality
   ▶ Adequate sampling rate and pattern

3. Use as many quality metrics as possible.
   ▶ Beyond round-trip metrics
   ▶ More than one to two types of metrics
   ▶ Metrics a function of packet size

4. Others:
   ▶ Low measurement overhead
   ▶ Lightweight
1. Do not work for many systems and paths.
   - ICMP, TCP SYN, TCP RST subject
   - Anomalous probe packets
   - Invalid assumptions (e.g., consecutive IPID)
Reliable measurement

1. Do not work for many systems and paths.
   - ICMP, TCP SYN, TCP RST subject
   - Anomalous probe packets
   - Invalid assumptions (e.g., consecutive IPID)

2. Do not measure data-path quality.
   - ICMP and data packets
   - TCP SYN and TCP data packets
Reliable measurement

1. Do not work for many systems and paths.
   - ICMP, TCP SYN, TCP RST subject
   - Anomalous probe packets
   - Invalid assumptions (e.g., consecutive IPID)

2. Do not measure data-path quality.
   - ICMP and data packets
   - TCP SYN and TCP data packets

3. Do not support adequate sampling rate and pattern.
Metric-rich measurement

1. Round-trip measurement limitation
   - Asymmetric network paths
   - Asymmetric data traffic
Metric-rich measurement

1. Round-trip measurement limitation
   - Asymmetric network paths
   - Asymmetric data traffic

2. Most tools offer one to two metrics.
   - Tulip: loss, reordering, and queueing delay
   - Using multiple tools
Metric-rich measurement

1. Round-trip measurement limitation
   - Asymmetric network paths
   - Asymmetric data traffic
2. Most tools offer one to two metrics.
   - Tulip: loss, reordering, and queuing delay
   - Using multiple tools
3. Do not support different response packet sizes.
Design principles of OneProbe

1. Use data packets for measurement
Design principles of OneProbe

1. Use data packets for measurement
2. Use normal and basic data transmission mechanisms
Design principles of OneProbe

1. Use data packets for measurement
2. Use normal and basic data transmission mechanisms
3. Use normal application protocol mechanisms
Design principles of OneProbe

1. Use data packets for measurement
2. Use normal and basic data transmission mechanisms
3. Use normal application protocol mechanisms
4. Elicit response packets that contain sufficient information.
Design principles of OneProbe

1. Use data packets for measurement
2. Use normal and basic data transmission mechanisms
3. Use normal application protocol mechanisms
4. Elicit response packets that contain sufficient information.
5. Control the size of the response packets
What can OneProbe measure?
OneProbe’s features

1. Per-packet round-trip time (RTT)
OneProbe’s features

1. Per-packet round-trip time (RTT)
2. Forward- and reverse-path packet loss rate
OneProbe’s features

1. Per-packet round-trip time (RTT)
2. Forward- and reverse-path packet loss rate
3. Forward- and reverse-path packet reordering rate
OneProbe’s features

1. Per-packet round-trip time (RTT)
2. Forward- and reverse-path packet loss rate
3. Forward- and reverse-path packet reordering rate
4. Forward- and reverse-path packet capacity (still in progress)
OneProbe’s features

1. Per-packet round-trip time (RTT)
2. Forward- and reverse-path packet loss rate
3. Forward- and reverse-path packet reordering rate
4. Forward- and reverse-path packet capacity (still in progress)
5. RTT and loss and reordering rates as a function of packet sizes
OneProbe’s features

1. Per-packet round-trip time (RTT)
2. Forward- and reverse-path packet loss rate
3. Forward- and reverse-path packet reordering rate
4. Forward- and reverse-path packet capacity (still in progress)
5. RTT and loss and reordering rates as a function of packet sizes
6. Support Poisson and periodic sampling
OneProbe’s path metrics

- Overall data-path quality
  - Forward-path quality
    - Forward-path loss rate
    - Number of first-probe-packet loss
    - Number of first-response-packet loss
  - Reverse-path quality
    - Reverse-path loss rate
    - Forward-path reordering rate
      - Number of probes sent without probe packet loss
      - Number of reordered probe packets
    - Reverse-path reordering rate
      - Number of probes sent without response packet loss
      - Number of reordered response packets
Diurnal RTT and loss patterns

Olympic Games

22 Aug 20:37 UTC

RTT (milliseconds)

Round-trip Loss Rate (%)
Discrepancy between Ping and OneProbe RTTs

OneProbe RTT

Ping RTT

12 Aug 16:39 UTC
Asymmetric loss rates and loss-pair RTTs

(a) Forward path

(b) Reverse path
Effect of packet size on reordering rates

(c) Forward-path reordering

(d) Reverse-path reordering
System architecture

User

HTTP

Probe and response packet sizes (e.g., 1500 and 240 bytes)

URL (e.g., http://usenix.org)

Sampling rate (e.g., 2Hz) and sampling pattern (e.g., Poisson)

OneProbe

Find qualified URLs

Prepare HTTP GET requests

HTTP GET requests

TCP

Probe packets

Response packets

Network

HTTP/OneProbe

HTTP helper

Rocky K. C. Chang

Department of Computing The Hong Kong Polytechnic University

A Tutorial on OneProbe (oneprobe.org)
Two layers

1. OneProbe (TCP)
   - Prepare and send probe TCP data packets.
   - Receive response TCP data packets.
   - Obtain the measurement from the response packets.
Two layers

1. OneProbe (TCP)
   - Prepare and send probe TCP data packets.
   - Receive response TCP data packets.
   - Obtain the measurement from the response packets.

2. Application layer (HTTP helper for HTTP/OneProbe)
   - Find qualified URLs.
   - Prepare HTTP GET requests
The probe design

1. Send two (back-to-back) probe data packets.
   - Two is a minimum for forward-path packet reordering measurement.
   - The second packet gives a “hint” about the first probe packet’s delivery status.
The probe design

1. Send two (back-to-back) probe data packets.
   - Two is a minimum for forward-path packet reordering measurement.
   - The second packet gives a “hint” about the first probe packet’s delivery status.

2. Elicit a minimum of two new response data packets.
   - Two is a minimum for reverse-path packet reordering measurement.
The probe design

1. Send two (back-to-back) probe data packets.
   - Two is a minimum for forward-path packet reordering measurement.
   - The second packet gives a “hint” about the first probe packet’s delivery status.

2. Elicit a minimum of two new response data packets.
   - Two is a minimum for reverse-path packet reordering measurement.

3. The response packets are distinguishable and predeterminable for different delivery statuses of the probe packets.
The probing process

1. \( Cm|n \): probe data packet, where \( m \) and \( n \) are the TCP data segment’s sequence and acknowledgment numbers.
2. \( Sm|n \): response data packet.
3. Full-sized TCP data segments (i.e., the maximum segment size, MSS).
4. Use \( m = 1, 2, \cdots \) for server’s TCP data segments and \( 1', 2', \cdots \) for OneProbe’s data segments.
Transmission pattern for inordered probe

1. The server can send only one new data segment after receiving a probe packet.
2. Each probe packet acknowledges only one data segment from the server.
3. The servers TCP send window size is set to two segments by the probe packet’s advertised window.
Per-packet RTT measurement

1. Based on a probe packet and its induced new data packet (e.g., $C3'|1$ and $S3|3'$)
2. Two RTT observations in a probe round
3. Use only the first-probe-packet-RTT for measurement.
Packet loss and reordering events

1. Five possible cases on forward path:
   - F0: Both probe packets arrive at the server with the same order.
   - FR: Both probe packets arrive at the server with a reverse order.
   - F1: The first probe packet is lost, but the second arrives at the server.
   - F2: The first probe packet arrives at the server, but the second is lost.
   - F3: Both probe packets are lost.
Packet loss and reordering events

1. Five possible cases on forward path:
   - F0: Both probe packets arrive at the server with the same order.
   - FR: Both probe packets arrive at the server with a reverse order.
   - F1: The first probe packet is lost, but the second arrives at the server.
   - F2: The first probe packet arrives at the server, but the second is lost.
   - F3: Both probe packets are lost.

2. Distinguishable new response data packets for F0-F3.
Packet loss and reordering events

1. Five possible cases on forward path:
   - F0: Both probe packets arrive at the server with the same order.
   - FR: Both probe packets arrive at the server with a reverse order.
   - F1: The first probe packet is lost, but the second arrives at the server.
   - F2: The first probe packet arrives at the server, but the second is lost.
   - F3: Both probe packets are lost.

2. Distinguishable new response data packets for F0-F3.

3. Five similar cases for reverse path: R0, RR, R1, R2, and R3.
18 possible loss-ordering scenarios

<table>
<thead>
<tr>
<th></th>
<th>R0</th>
<th>RR</th>
<th>R1</th>
<th>R2</th>
<th>R3</th>
</tr>
</thead>
<tbody>
<tr>
<td>F0</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
</tr>
<tr>
<td>FR</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
</tr>
<tr>
<td>F1</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>–</td>
</tr>
<tr>
<td>F2</td>
<td>–</td>
<td>–</td>
<td>√</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>F3</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
</tbody>
</table>
The scenario of FR-R0

Server

Measuring node

C3'|1 C4'|2

Timeout

Time

Rocky K. C. Chang
Department of Computing The Hong Kong Polytechnic University

A Tutorial on OneProbe (oneprobe.org)
The scenario of F1-R0
The scenario of F2-R0

Server

Measuring node

Timeout

C3'|1 C4'|2 S3'|3' S2'|3'
The scenario of F3-R0

Timeout

Server

Measuring node

C3'|1 C4'|2

Time

Rocky K. C. Chang
Department of Computing  The Hong Kong Polytechnic University
### Response packets for the loss-ordering scenarios

<table>
<thead>
<tr>
<th>Path events</th>
<th>1st response packets</th>
<th>2nd response packets</th>
<th>3rd response packets</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. F0×R0</td>
<td>S3</td>
<td>3′</td>
<td>S4</td>
</tr>
<tr>
<td>2. F0×RR</td>
<td>S4</td>
<td>4′</td>
<td>S3</td>
</tr>
<tr>
<td>3. F0×R1</td>
<td>S4</td>
<td>4′</td>
<td>S3</td>
</tr>
<tr>
<td>4. F0×R2</td>
<td>S3</td>
<td>3′</td>
<td>S3</td>
</tr>
<tr>
<td>5. F0×R3</td>
<td>S3</td>
<td>4′</td>
<td>–</td>
</tr>
<tr>
<td>6. F1×R0</td>
<td>–</td>
<td>–</td>
<td>S3</td>
</tr>
<tr>
<td>7. F1×RR</td>
<td>–</td>
<td>–</td>
<td>S3</td>
</tr>
<tr>
<td>8. F1×R1</td>
<td>S4</td>
<td>2′</td>
<td>S3</td>
</tr>
<tr>
<td>9. F1×R2</td>
<td>S3</td>
<td>2′</td>
<td>S3</td>
</tr>
<tr>
<td>10. F1×R3</td>
<td>S3</td>
<td>4′</td>
<td>–</td>
</tr>
<tr>
<td>11. F2×R0</td>
<td>S3</td>
<td>3′</td>
<td>S2</td>
</tr>
<tr>
<td>12. F2×R1</td>
<td>S2</td>
<td>3′</td>
<td>–</td>
</tr>
<tr>
<td>13. F3</td>
<td>S1</td>
<td>2′</td>
<td>–</td>
</tr>
</tbody>
</table>
1. All 18 cases can be distinguished except for
   A1. \( F1 \times R2 \) and \( F1 \times R3 \): \( S3|2' \) and \( \hat{S}3|2' \) are identical, and the server may retransmit more than once.
   A2. \( F1 \times RR \) and \( F1 \times R1 \): Similar to that for A1.
   A3. \( F0 \times R3 \) and \( FR \times R3 \): The same response packet \( \hat{S}3|4' \).
Path event distinguishability

1. All 18 cases can be distinguished except for
   - **A1.** $F1 \times R2$ and $F1 \times R3$: $S3|2'$ and $\hat{S}3|2'$ are identical, and the server may retransmit more than once.
   - **A2.** $F1 \times R R$ and $F1 \times R1$: Similar to that for A1.
   - **A3.** $F0 \times R3$ and $FR \times R3$: The same response packet $\hat{S}3|4'$.

2. A1 and A2 make the delivery status of $S3|2'$ uncertain.
1. All 18 cases can be distinguished except for
   A1. F1×R2 and F1×R3: S3|2’ and $\hat{S}3|2’$ are identical, and the server may retransmit more than once.
   A2. F1×RR and F1×R1: Similar to that for A1.
   A3. F0×R3 and FR×R3: The same response packet $\hat{S}3|4’$.
2. A1 and A2 make the delivery status of S3|2’ uncertain.
3. A3 makes the probe’s order of arrival uncertain.
Resolving the ambiguities

1. A1 and A2: use RTT to differentiate between non-retransmission packets and retransmission packets.
Resolving the ambiguities

1. A1 and A2: use RTT to differentiate between non-retransmission packets and retransmission packets.
   - C4’s timestamp for the case of F0×R3.
   - C3’s timestamp for the case of FR×R3.
Assistance from TCP ACKs

1. Two types of TCP ACKs: out-of-ordered-packet ACK (OOP-ACK) and filling-a-hole ACK (FAH-ACK).
   - An early arrival of $C_4'|2$ could immediately trigger an OOP-ACK,
   - A late arrival of $C_3'|1$ could immediately trigger an FAH-ACK.
   - Some systems did not return the OOP-ACK, but all the systems tested returned the FAH-ACK.
Assistance from TCP ACKs

1. Two types of TCP ACKs: out-of-ordered-packet ACK (OOP-ACK) and filling-a-hole ACK (FAH-ACK).
   - An early arrival of $C4'|2$ could immediately trigger an OOP-ACK,
   - A late arrival of $C3'|1$ could immediately trigger an FAH-ACK.
   - Some systems did not return the OOP-ACK, but all the systems tested returned the FAH-ACK.

2. Use the FAH-ACK to accelerate the detection of the forward-path reordering events without waiting for the data retransmissions.
Assistance from TCP ACKs

1. Two types of TCP ACKs: out-of-ordered-packet ACK (OOP-ACK) and filling-a-hole ACK (FAH-ACK).
   - An early arrival of $C4'|2$ could immediately trigger an OOP-ACK,
   - A late arrival of $C3'|1$ could immediately trigger an FAH-ACK.
   - Some systems did not return the OOP-ACK, but all the systems tested returned the FAH-ACK.

2. Use the FAH-ACK to accelerate the detection of the forward-path reordering events without waiting for the data retransmissions.

3. Use the FAH-ACK to disambiguate A3.
The FR-R0 scenario with a FAH-ACK
Successive probe rounds

1. Send two probe TCP data packets
2. Receive a response TCP data packet
3. Determine the packet delivery statuses?
   - yes: Perform some preparation works
   - no: Repeat from step 1
1. A new probe could be dispatched immediately after receiving two response packets for events 1-2.
Starting a new probe round in the same TCP connection

1. A new probe could be dispatched immediately after receiving two response packets for events 1-2.
2. For others, a new probe is withheld to elicit a retransmission packet.
   - Send one or more new TCP ACKs to increase the server’s cwnd back to two for path events 3-18.
   - After receiving two new data segments, OneProbe dispatches a new probe: \( \{C5', C6'\} \) for events 3-10, \( \{C4', C5'\} \) for events 16-17, and \( \{C3', C4'\} \) for event 18.
   - Handling events 11-15 is more complicated.
1. Use multiple concurrent TCP connections to achieve the user-configured sampling pattern and rate.
Using multiple TCP connections

1. Use multiple concurrent TCP connections to achieve the user-configured sampling pattern and rate.

2. Assign randomly selected source IP addresses from an address pool to the connections.
Using multiple TCP connections

1. Use multiple concurrent TCP connections to achieve the user-configured sampling pattern and rate.
2. Assign randomly selected source IP addresses from an address pool to the connections.
3. Our experience shows that 10 connections is sufficient for supporting periodic sampling with a rate of two probes per second.
Sampling the packet loss and reordering rates

1. Similar to the RTT measurement, OneProbe uses only the first packet for the loss measurement.
Sampling the packet loss and reordering rates

1. Similar to the RTT measurement, OneProbe uses only the first packet for the loss measurement.

2. After conducting 120 probe rounds over one minute, OneProbe computes the forward-path loss rate by dividing the number of the first-probe-packet-loss events by 120.
1. Similar to the RTT measurement, OneProbe uses only the first packet for the loss measurement.

2. After conducting 120 probe rounds over one minute, OneProbe computes the forward-path loss rate by dividing the number of the first-probe-packet-loss events by 120.

3. Similar sampling for the reverse-path loss rate and reordering rates.
HTTP/OneProbe

User

Probe and response packet sizes (e.g., 1500 and 240 bytes)

URL (e.g., http://usenix.org)

Sampling rate (e.g., 2Hz) and sampling pattern (e.g., Poisson)

HTTP

Probe packets

Find qualified URLs

Prepare HTTP GET requests

HTTP GET requests

OneProbe

Response packets

Network

HTTP helper

Rocky K. C. Chang
Department of Computing, The Hong Kong Polytechnic University

A Tutorial on OneProbe (oneprobe.org)
1. Before sending the first probe \( \{C_1', C_2'\} \), HTTP/OneProbe sends an ACK to ramp up the server’s cwnd to two segments.

2. Therefore, \( C_0' \) and \( S_1-S_3 \) are not used for OneProbe measurement.
The HTTP helper

1. Finding qualified http URLs
   - A qualified http URL: its HTTP GET request can be retrofitted into a probe packet, and the GET request can induce at least 5 response packets from the server.
   - Verifying whether a user-specified URL meets the size requirement for the response packets.
   - Besides, the HTTP GET request for a qualified URL must induce a 200(OK) response.
1. Finding qualified http URLs
   - A qualified http URL: its HTTP GET request can be retrofitted into a probe packet, and the GET request can induce at least 5 response packets from the server.
   - Verifying whether a user-specified URL meets the size requirement for the response packets.
   - Besides, the HTTP GET request for a qualified URL must induce a 200(OK) response.

2. Preparing the HTTP GET requests
   - To craft a probe packet for an HTTP request, expand the packet size through the `Referer` field and a customized string.
   - Exploit HTTP/1.1’s request pipelining to include a GET message in each probe packet.
An OneProbe implementation

1. An OneProbe session generally consists of concurrent TCP connections.
An OneProbe implementation

1. An OneProbe session generally consists of concurrent TCP connections.

2. Two consecutive phases in each connection: preparation and probing
An OneProbe implementation

1. An OneProbe session generally consists of concurrent TCP connections.
2. Two consecutive phases in each connection: preparation and probing.
3. The preparation phase: performing the ground works for the probing phase.
An OneProbe implementation

1. An OneProbe session generally consists of concurrent TCP connections.
2. Two consecutive phases in each connection: preparation and probing.
3. The preparation phase: performing the ground works for the probing phase.
4. The probing phase: dispatching the probes and analyzing the response packets.
The preparation and probing phases

Preparation phase:
- Configuring the probe and response packet sizes
- Ramping up the server's cwnd

Probing phase:
- Sending the probe and analyzing the results
- Getting the next probe task
- Preparing for the next probe task
- Terminating the TCP connection

Exception or Done:
- Exception or Done
  - OK
  - No exception
  - No probe task
Diagnosing self-induced packet losses

1. Perform a self-diagnosis to confirm that the measurement is free of self-induced packet losses.
Diagnosing self-induced packet losses

1. Perform a self-diagnosis to confirm that the measurement is free of self-induced packet losses.

2. Forward path: Uses libpcap to verify the delivery of each outgoing probe packet to the network.
Diagnosing self-induced packet losses

1. Perform a self-diagnosis to confirm that the measurement is free of self-induced packet losses.

2. Forward path: Uses libpcap to verify the delivery of each outgoing probe packet to the network.

3. Reverse path: Monitor the `ps_drop` variable returned by the libpcap’s `pcap_stats()` function to detect losses.

Rocky K. C. Chang  
Department of Computing  The Hong Kong Polytechnic University  
A Tutorial on OneProbe  (oneprobe.org)
Validation of OneProbe

1. Designed Validator, a small, but just sufficient, suite of validation tests for OneProbe.
Validation of OneProbe

1. Designed Validator, a small, but just sufficient, suite of validation tests for OneProbe.
2. Validator constrains the server’s cwnd to two segments.
Validation of OneProbe

1. Designed Validator, a small, but just sufficient, suite of validation tests for OneProbe.

2. Validator constrains the server’s \textit{cwnd} to two segments.

3. Validator does not acknowledge the response data packets in order to simulate reverse-path losses.
Validation of OneProbe

1. Designed Validator, a small, but just sufficient, suite of validation tests for OneProbe.

2. Validator constrains the server’s \( cwnd \) to two segments.

3. Validator does not acknowledge the response data packets in order to simulate reverse-path losses.

4. The four validation tests V0-V2 that “simulate” the forward-path events F0-F2.

<table>
<thead>
<tr>
<th>Tests</th>
<th>Testing probes</th>
<th>Expected packets induced from server</th>
<th>Expected data retransmissions</th>
</tr>
</thead>
<tbody>
<tr>
<td>V0.</td>
<td>{C3', C4'}</td>
<td>{S3|3', S4|4'}</td>
<td>( \hat{S}3|4' )</td>
</tr>
<tr>
<td>VR.</td>
<td>{C4', C3'}</td>
<td>{S3|2', S4|2'}</td>
<td>( \hat{S}3|4' )</td>
</tr>
<tr>
<td>V1.</td>
<td>C4' only</td>
<td>{S3|2', S4|2'}</td>
<td>( \hat{S}3|2' )</td>
</tr>
<tr>
<td>V2.</td>
<td>C3' only</td>
<td>S3|3'</td>
<td>( \hat{S}2|3' )</td>
</tr>
</tbody>
</table>
## Validation on web server software and operating systems

<table>
<thead>
<tr>
<th>Systems tested in our lab.:</th>
<th>FreeBSD v4.5/4.11/5.5/6.0/6.2, Linux kernel v2.4.20/2.6.5/2.6.11/2.6.15/2.6.18/2.6.20, MacOSX 10.4 server, NetBSD 3.1, OpenBSD 4.1, Solaris 10.1, Windows 2000/XP/Vista</th>
</tr>
</thead>
<tbody>
<tr>
<td>Systems tested in the Internet:</td>
<td>AIX, AS/400, BSD/OS, Compaq Tru64, F5 Big-IP, HP-UX, IRIX, MacOS, NetApp NetCache, NetWare, OpenVMS, OS/2, SCO Unix, Solaris 8/9, SunOS 4, VM, Microsoft Windows NT4/98/Server 2003/2008</td>
</tr>
<tr>
<td>Servers tested in our lab.:</td>
<td>Abyss, Apache, Lighttpd, Microsoft IIS, Nginx</td>
</tr>
</tbody>
</table>
Validation on web servers in the wild

1. Tested 37,874 websites randomly selected.
Validation on web servers in the wild

1. Tested 37,874 websites randomly selected.
2. Successful (93.00%): These servers passed all tests.
Validation on web servers in the wild

1. Tested 37,874 websites randomly selected.
2. Successful (93.00%): These servers passed all tests.
3. Failures in the preparation phase (1.03%): These websites failed to return the expected \{S1, S2\}. Therefore, OneProbe could not start the probing phase.
Validation on web servers in the wild

1. Tested 37,874 websites randomly selected.
2. Successful (93.00%): These servers passed all tests.
3. Failures in the preparation phase (1.03%): These websites failed to return the expected \( \{S_1, S_2\} \). Therefore, OneProbe could not start the probing phase.
4. Failures in test V0 (0.26%): Most websites in this set replied with \( \{S_3|4', S_4|4'\} \), instead of the expected \( \{S_3|3', S_4|4'\} \). That is, they sent response packets after receiving both probe packets.
Validation on web servers in the wild

1. Tested 37,874 websites randomly selected.
2. Successful (93.00%): These servers passed all tests.
3. Failures in the preparation phase (1.03%): These websites failed to return the expected \{S1, S2\}. Therefore, OneProbe could not start the probing phase.
4. Failures in test V0 (0.26%): Most websites in this set replied with \{S3|4', S4|4'\}, instead of the expected \{S3|3', S4|4'\}. That is, they sent response packets after receiving both probe packets.
5. Failures in test VR (5.71%): Some websites appeared to have received an order-intact probe. Another showed that they did not receive the reordered \(C3'\).
Thanks