Software Defined Customization of Network Protocols with Layer 4.5

Dissertation Defense

LCDR Daniel Lukaszewski

Naval Postgraduate School
Department of Computer Science

Committee Members:

Dr. G. Xie
Dr. J. Kroll
Dr. M. Kölsch
Dr. J. Rohrer
Dr. P. Stanica
Dr. K. Wiegand

August 11, 2022

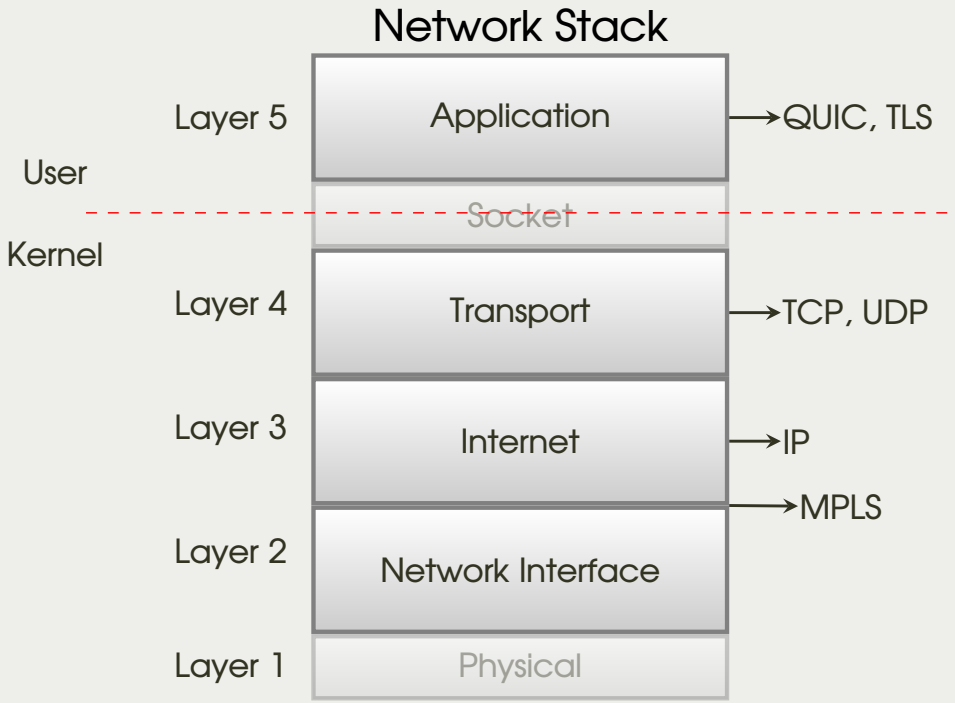
Thesis

A software defined Layer 4.5 protocol customization architecture is feasible and can provide continuous management capabilities, compatibility with modern encryption protocols, and rotating customizations on active application flows.

Outline

Outline

Protocol Customization



Protocol Customization: Traditionally extends existing protocols with new features, primarily for security and performance needs

- IP Options: Security, traceroute
- TCP Options: Selective acknowledgments, Multipath

Customization Distribution

Lightweight eBPF Application Framework (L3AF):

- Kernel function marketplace
- Configuration via L3AF daemon (per-machine)
- Lacks centralized control and continuous management

Application Protocol Plugins:

- Plugin repository
- Configuration via plugin negotiations
- Requires updates to protocol specifications
- Lacks centralized control and continuous management

Middlebox Interference Problem

Middlebox: "Any intermediary device performing functions other than the normal, standard functions of an IP router on the datagram path between a source host and destination host" (RFC 3234)

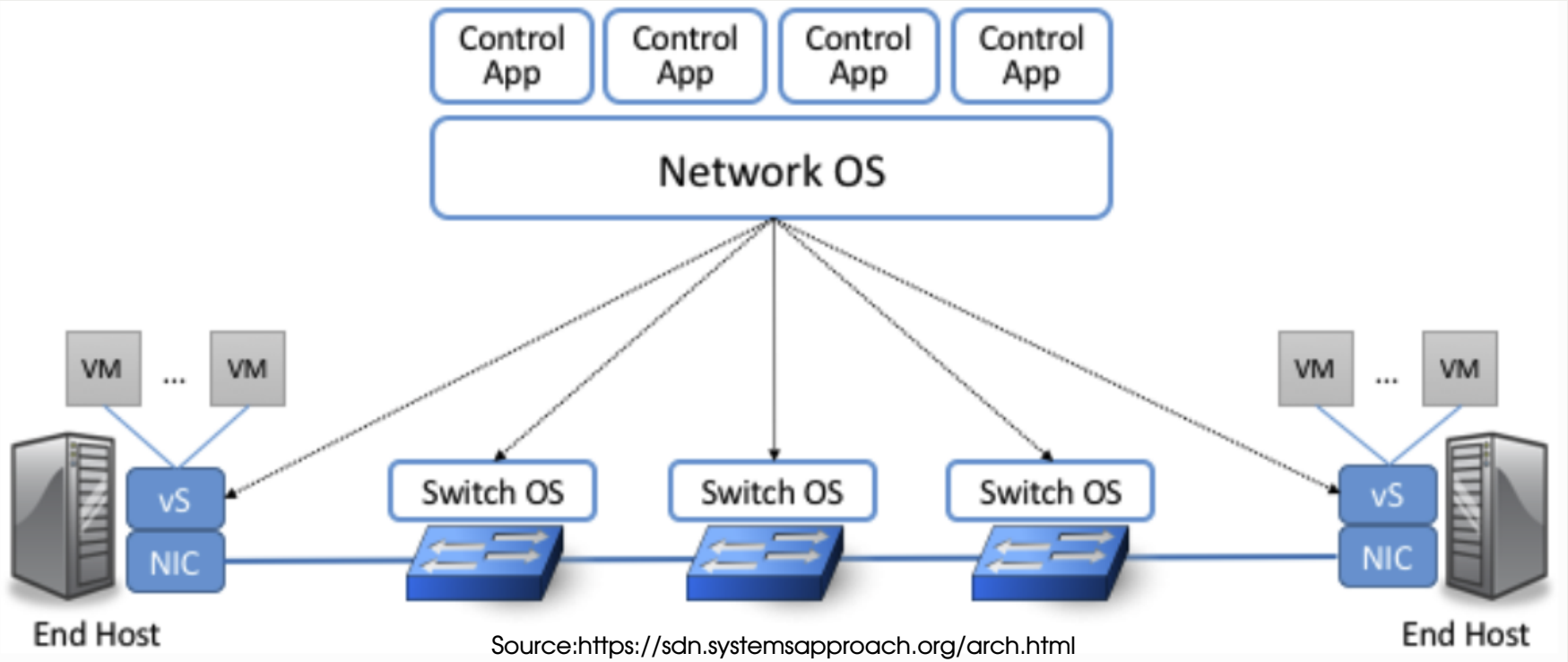
- Fallback:
  - Multipath TCP
  - QUIC
- Avoid:
  - Encryption:
    - TCPLS, QUIC
  - Tunneling:
    - VPN, IPSec

Source:https://www.snort.org

riverbed

Source:https://www.riverbed.com

Software-Defined Network (SDN)



Source:https://sdn.systemsapproach.org/arch.html

- Centralized control
  - Control channel with devices
  - Various apps run on controller

Outline

Problems with current methodology

1. Deployment is mostly ad hoc
   - manual configurations
   - specialized scripting
2. Lack the agility necessary to support the relatively high tempo of private customizations
   - Customization frequency (e.g, daily or hourly)
3. Middlebox interference is common
   - deployment burden

Outline

Thesis

A software defined Layer 4.5 protocol customization architecture is feasible and can provide continuous management capabilities, compatibility with modern encryption protocols, and rotating customizations on active application flows.

Significant Contributions

1. First software defined customization architecture (NETSOFT 2022)
   - 1.1 per-process protocol customization
   - 1.2 per-network security controls
   - 1.3 aid middlebox traversal
2. Improved understanding and flexible support for application transparent customization (e.g., encrypted flows)
3. Using the new capabilities of our architecture, we are the first to demonstrate the previously unsupported capability for active flow customization rotation

https://github.com/danluke2/software_defined_customization

Relevance to DoD: Publications

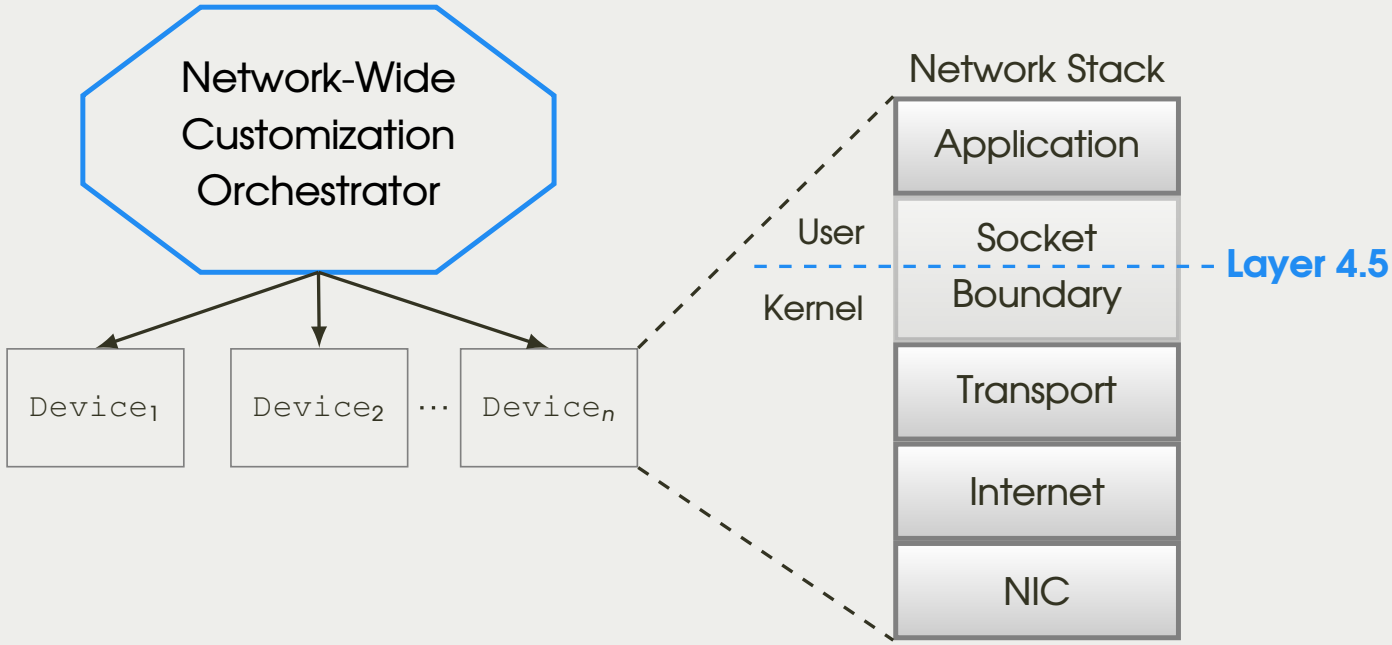
1. ``Strengthening SDN Security: Protocol Dialecting and Downgrade Attacks'' (NETSOFT 2021)
2. ``An Empirical Study of Application-Aware Traffic Compression for Shipboard SATCOM Links'' (MILCOM 2021)
3. ``Data Exfiltration via Flow Hijacking at the Socket Layer'' (HICSS-56 Under Review)

Relevance to DoD: Customizations

1. Traffic classification: Tag traffic with desired information
   - Quality of Service (QoS)
   - Aid network forensics
   - ML algorithm training (labeled data)
2. Rate-limiting: Add artificial delay
   - Brute force and DDoS attack mitigation
   - Web scraping
3. Link optimization: Reduce stress on network
   - WAN optimizers (e.g., riverbed)
   - MILCOM 2021

Outline

General Architecture

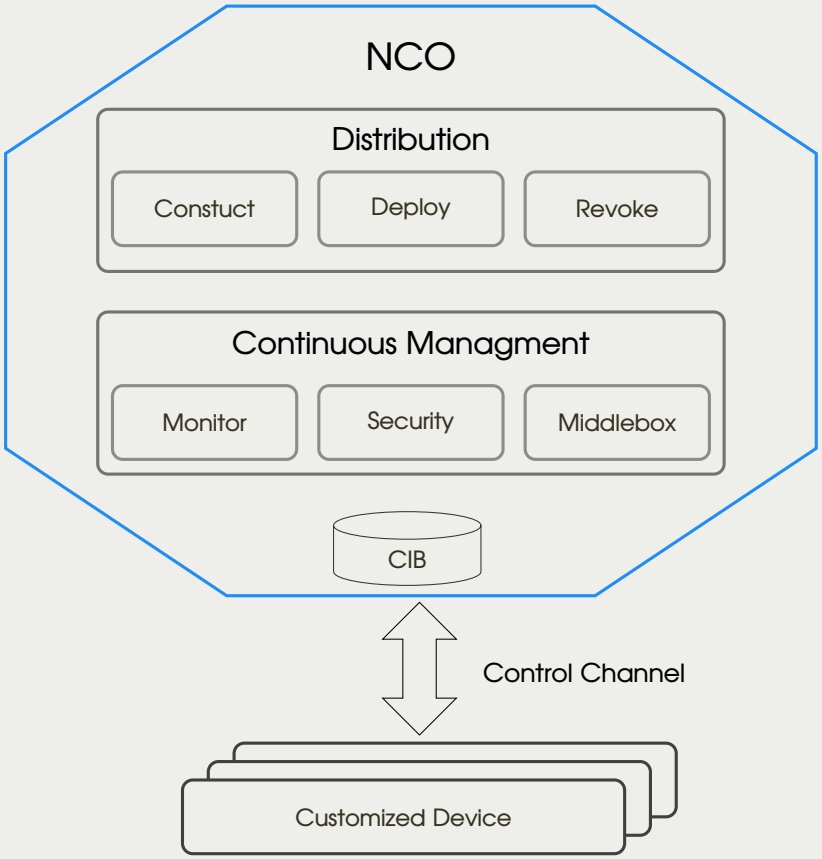


- A Network-Wide Customization Orchestrator (NCO) deploys customization modules as Layer 4.5 kernel extensions on devices
- Customization modules attached to matched application flows

Network-Wide Customization Orchestrator (NCO)

Distribution: Provide centralized control and deconfliction of the network customizations in use.

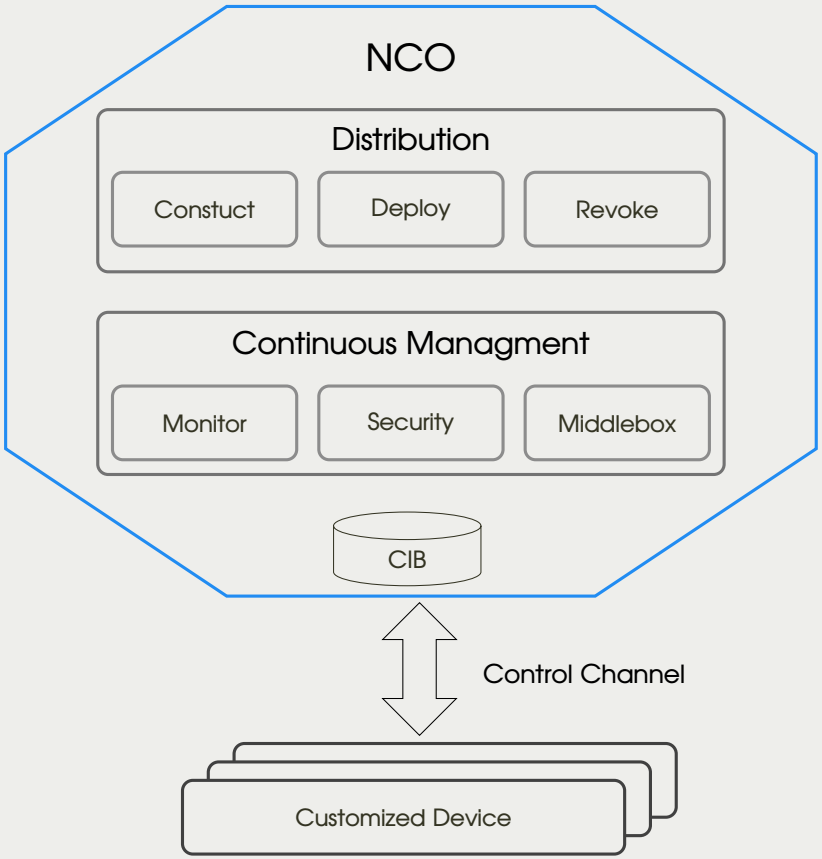
- Construct: Build module to match customized device
- Deploy: Transmit and install customization module over control channel
- Revoke: Remove customization module from device



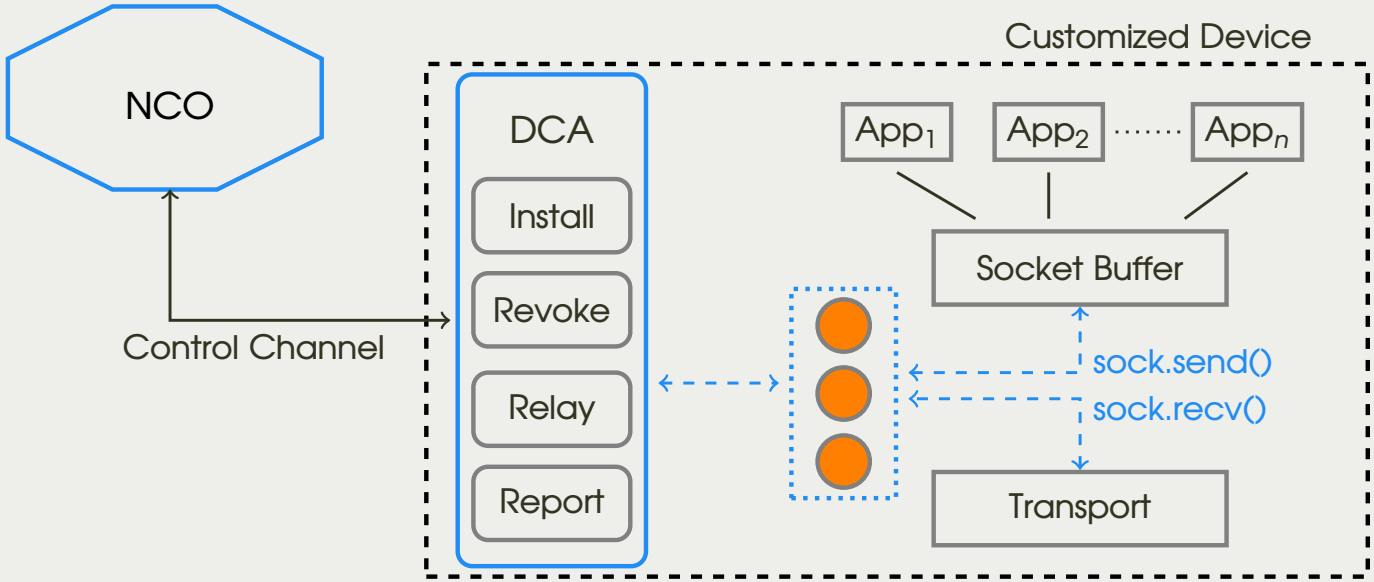
Network-Wide Customization Orchestrator (NCO)

Continuous Management: Monitor, validate, and coordinate use of customization modules on the network

- Monitor: receive reports from deployed customizations
- Security: mechanism for adding per-network module security requirements
- Middlebox: provide middlebox support to allow processing customized traffic

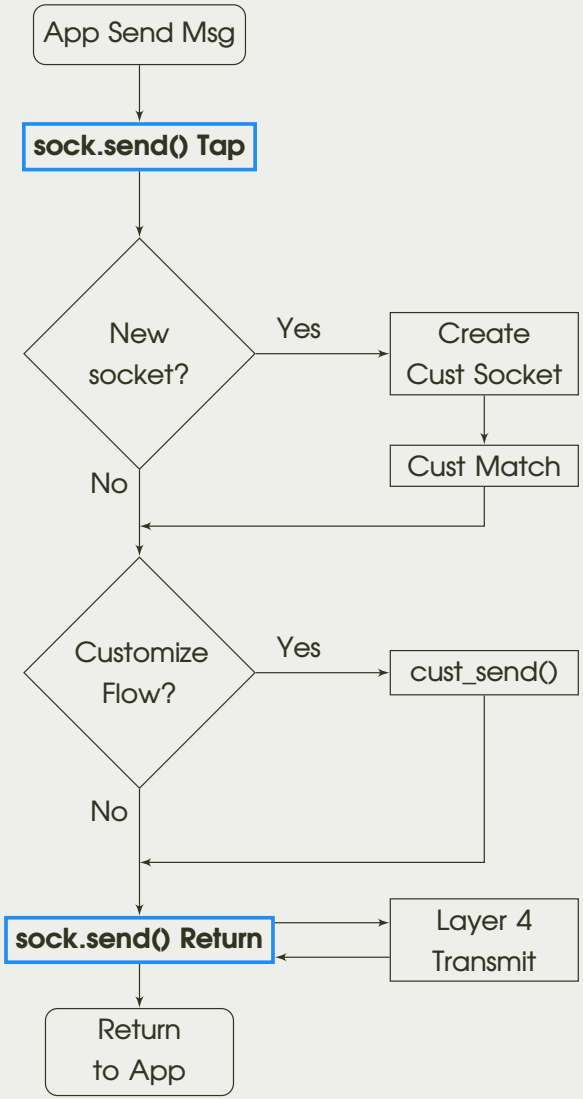


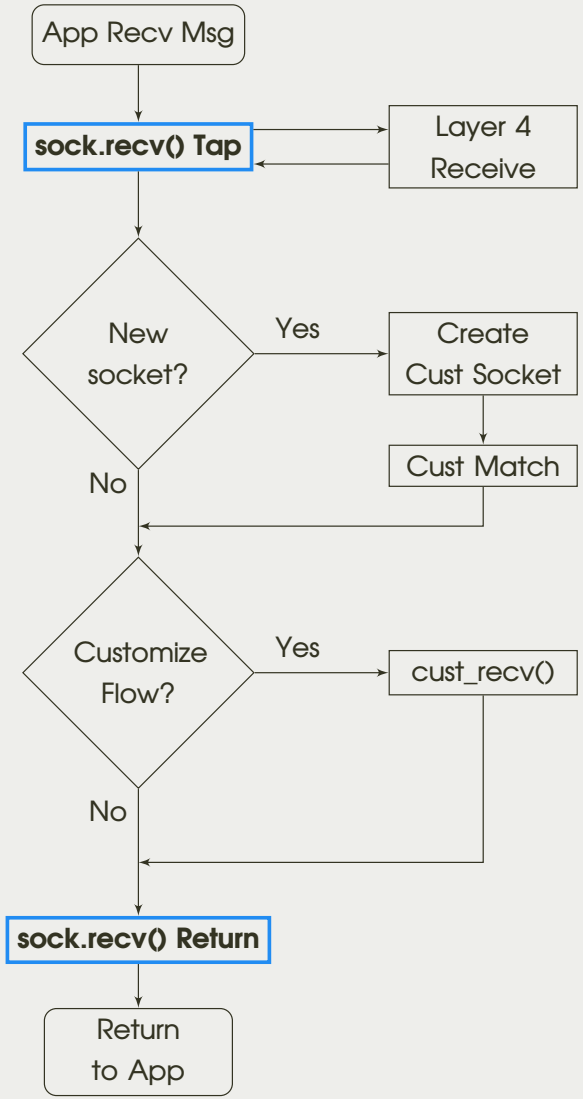
Device Customization Agent (DCA)



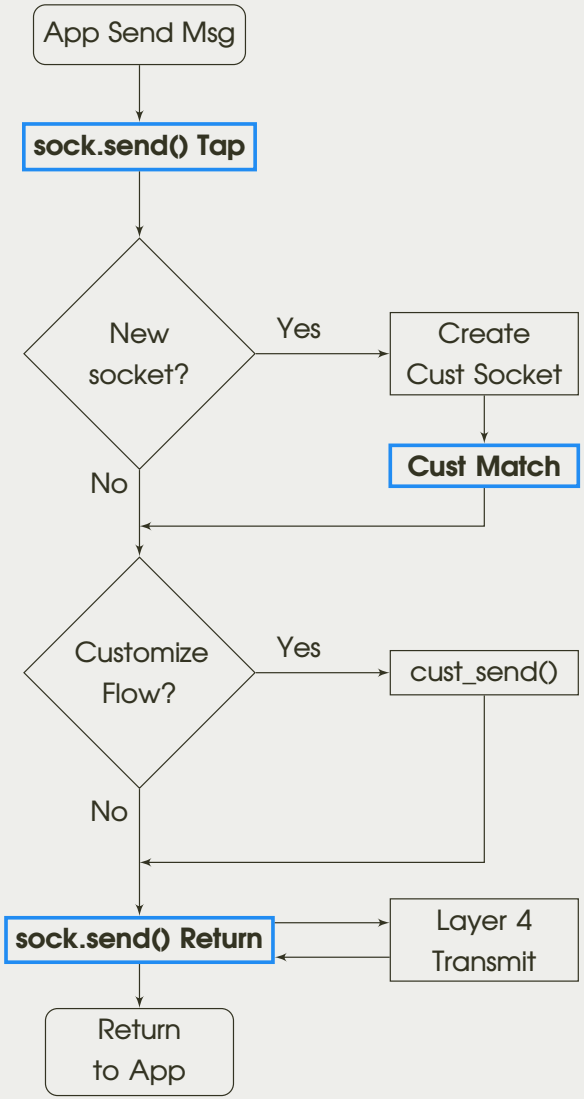
- DCA establishes control channel with NCO
- Install, revoke, and relay commands to customization modules (orange circles)
- Customizations invoked through the socket-transport tap

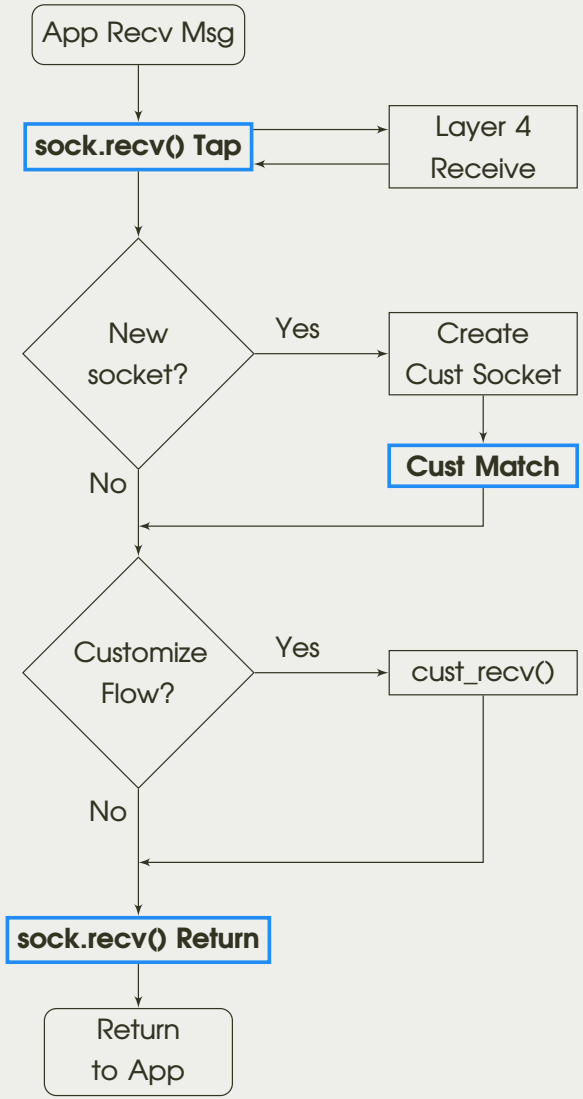
Layer 4.5 Tap Logic





Layer 4.5 Tap Logic





Customization Module Flow Matching

Examples:

1	Client:	1.1.1.1	**Chrome**	2.2.2.2	80	TCP
	Server:	2.2.2.2	80	1.1.1.1	**	TCP
2	Client:	**	**dig**	3.3.3.3	53	UDP
	Server:	3.3.3.3	53	**	**	UDP

Application Label:

- NCO can not predict dynamically generated port numbers
- Allows matching sockets before all 5-tuple values are known

Outline

Operational Network Customization

Traffic classification: Tag traffic with desired information

- Quality of Service (QoS)
- Aid network forensics
- ML algorithm training (labeled data)

Customization Module:

- UDP Message: Insert tag before each application header
  - Assuming each message is one IP packet of data
  - Message contains room for tag
- TCP Stream: Insert tag every 1000 bytes
  - Assuming MTU of 1500 bytes
  - At least one tag per transmitted packet

Prototype and Experiments

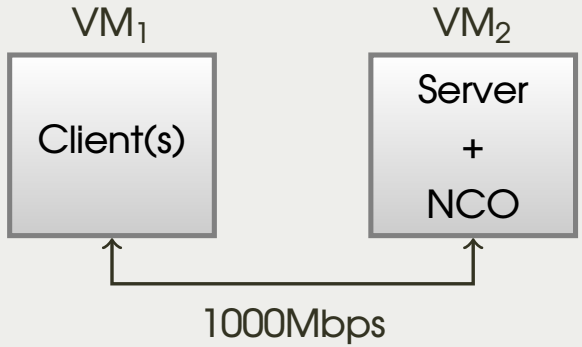
Prototype:

- NCO (1800 SLOC python)
- DCA (350 SLOC python, 1900 SLOC kernel module)
- Layer 4.5 customization modules (100-300 SLOC each)

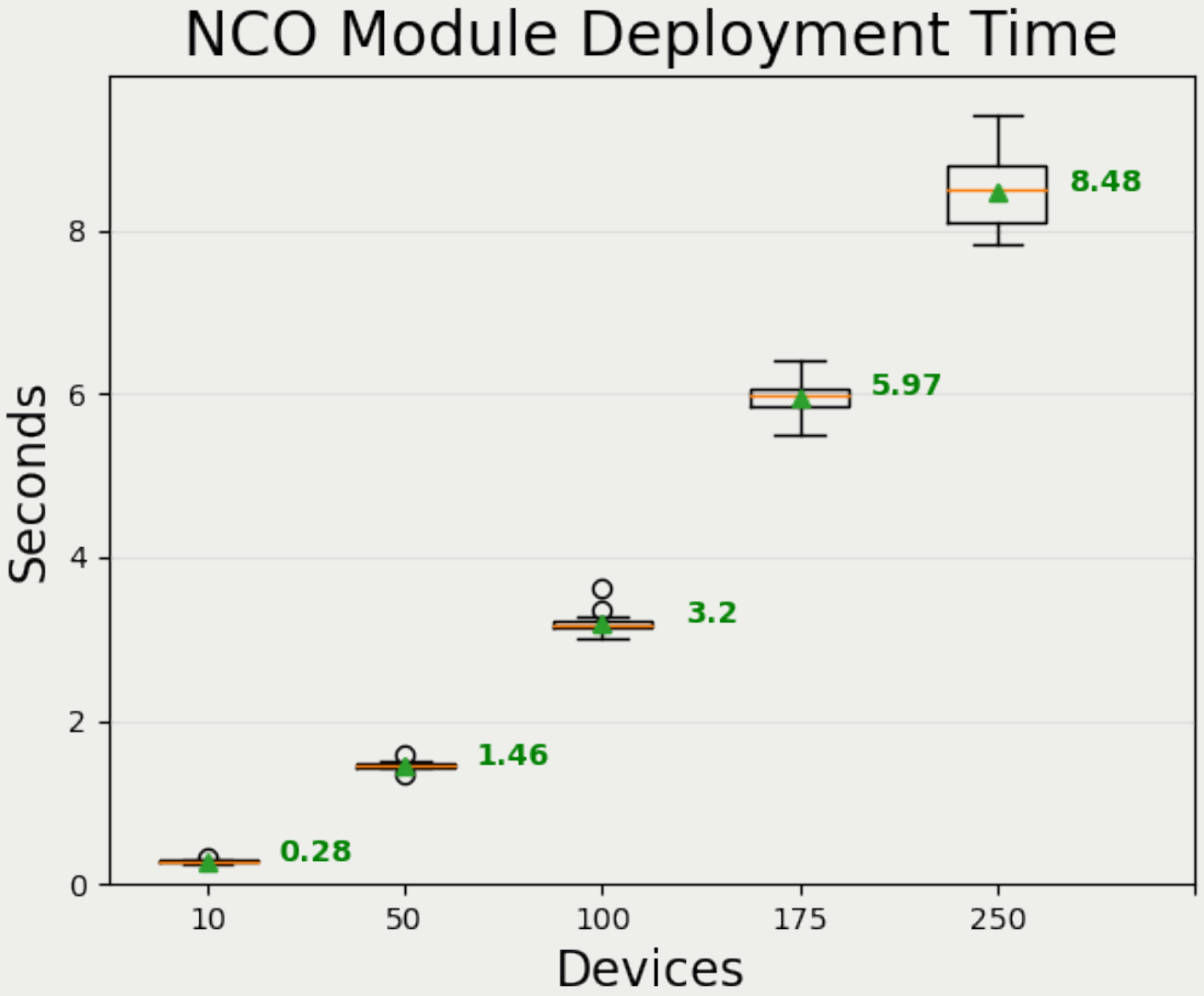
Experiments Performed:

- Distribution overhead
- Processing overhead
  - Layer 4.5 socket tap
  - Socket tap + customization
- NCO security functionality
- NCO middlebox functionality

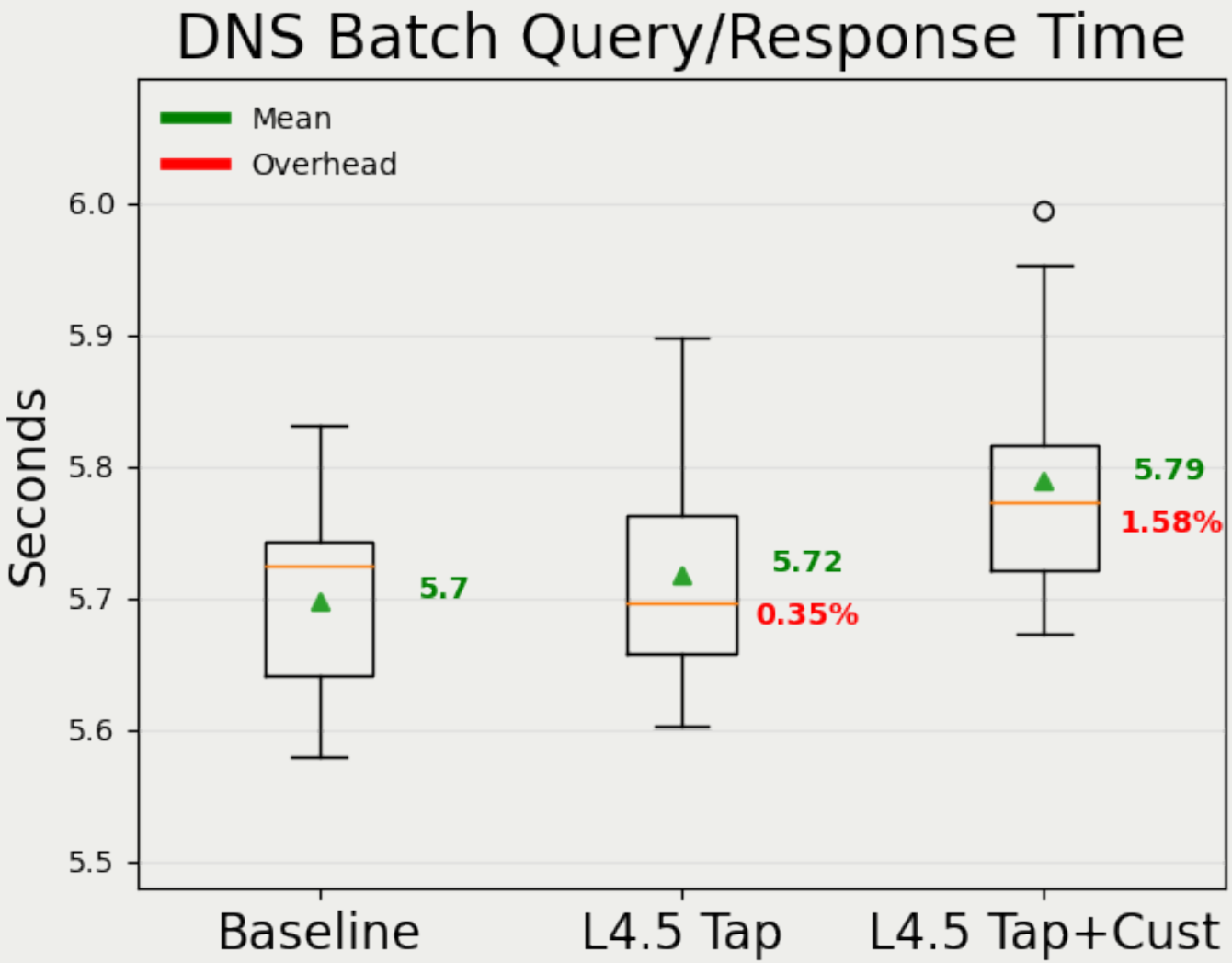
Virtual Testbed:



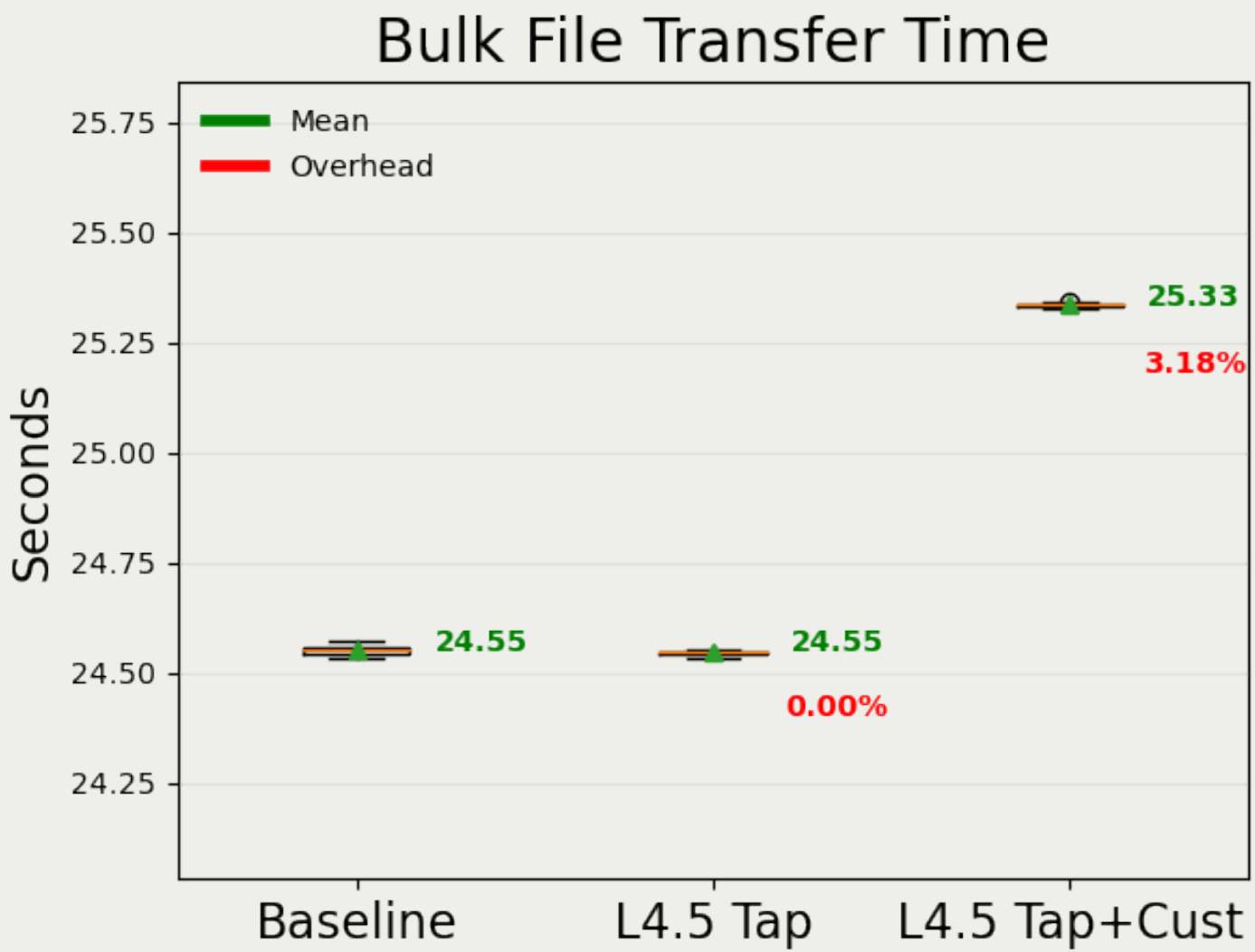
Distribution Overhead



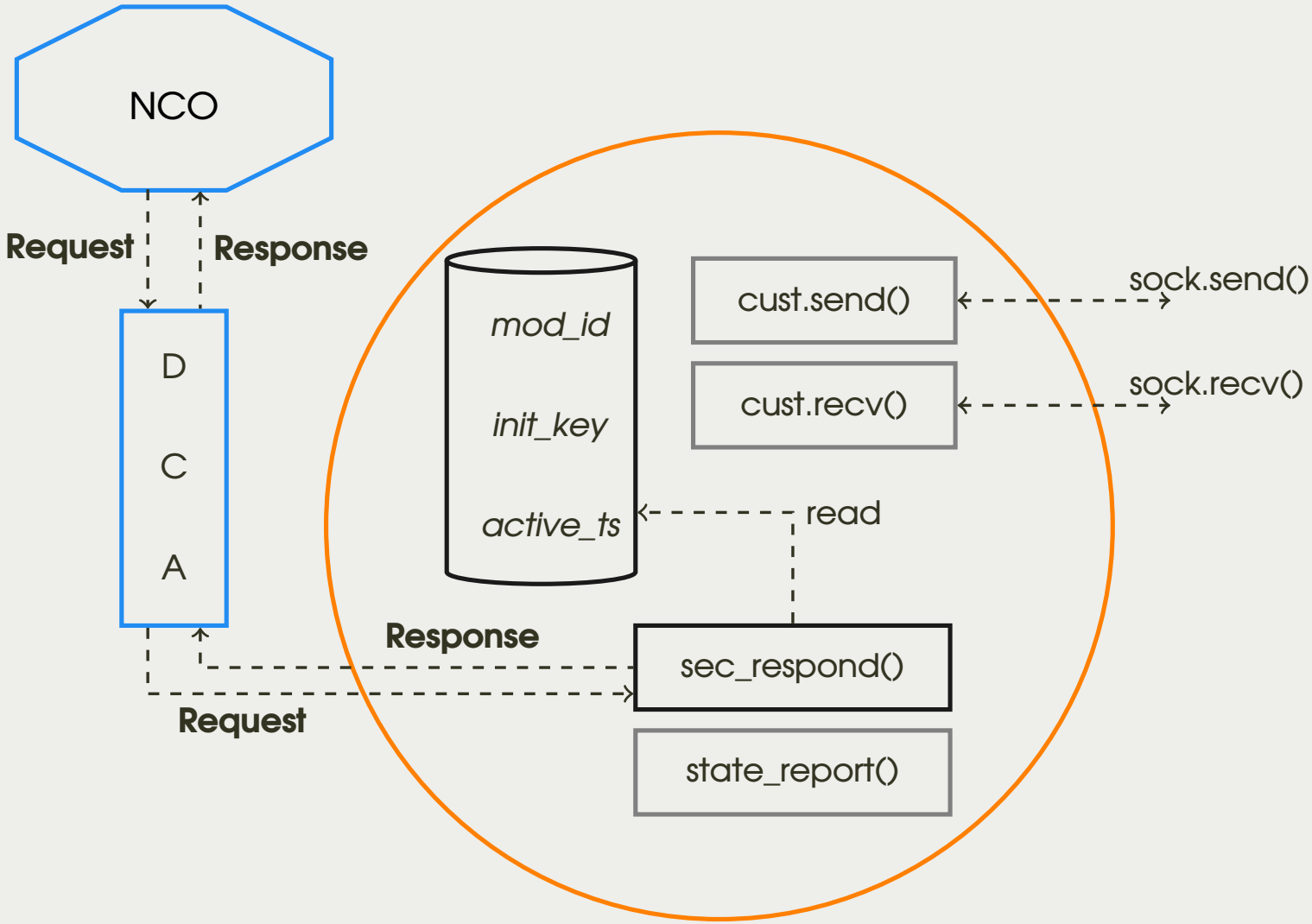
Processing Overhead: 1K Tag Insert/Delete Events



Processing Overhead: 3M Tag Insert/Delete Events



NCO/CM: Challenge/Response Security Check



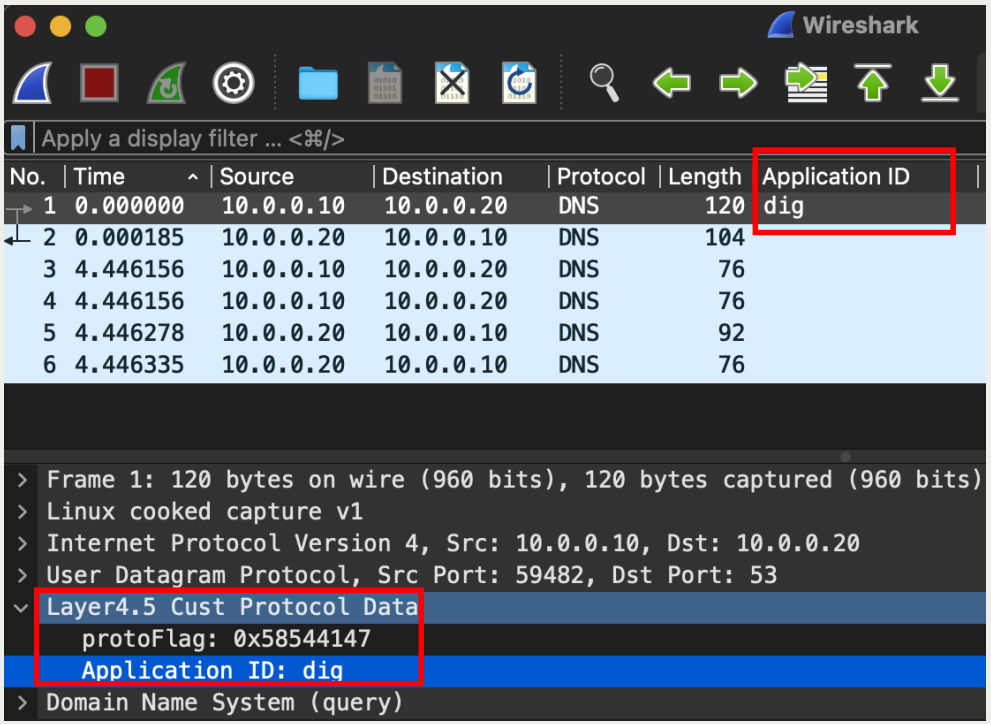
NCO/CM: Challenge/Response Security Check

```
NCO_security  Challenge before encrypt as hex is 2f5318c1cb2405d2
NCO_security  Sent challenge request to 2, challenge:
              {'cmd': 'challenge', 'id': 1,
               'iv': '6ead272b71116e01aa5834a3e4c5d284',
              'msg': '6da85e7d6f56944cb3f489ce12863bb2'}
NCO_security  The decrypted response is 2f5318c1cb2405d2000001
NCO_security  Challenge string matches
NCO_security  Module_id matches
```

NCO/CM: Deep Packet Inspection Middlebox

Problem: Unusual DNS client traffic

Goal: Determine ``non-standard'' applications performing DNS requests



Non-Layer 4.5 Middlebox:

- Inverse Customization Module

Client Customization:

- Insert application ID
- Target `dig` application

Server Customization:

- Remove application ID

Outline

Layer 4.5 Limitations of Initial Prototype

1. Event-driven:
   - Send/Recv calls only
   - L4 and below won't trigger (e.g., TCP ACK)
   - No control channel guarantee
2. One customization per socket:
   - Prevents chaining/order issues
   - Deconfliction at the NCO
3. Customization attachment:
   - Customizations apply to future sockets only
   - Implementation simplification

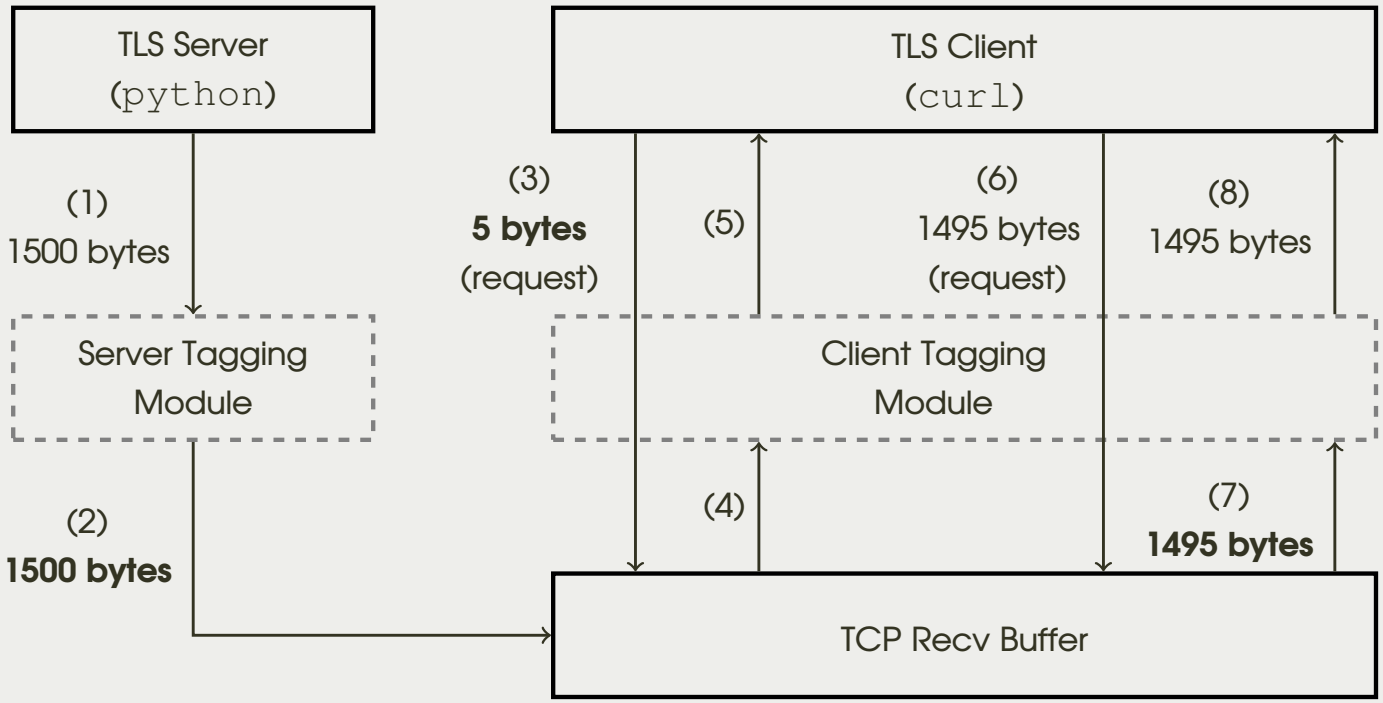
Layer 4.5 Limitations of Initial Prototype

4. Application behavior:
   - Send: 64KB or single packet
   - Recv: Multiple calls for single message
     - TLS: 5 byte record header specifies data length

5. Application encryption:
   - Flow encrypted before customization (send)
   - Flow decrypted after customization processing (recv)

Outline

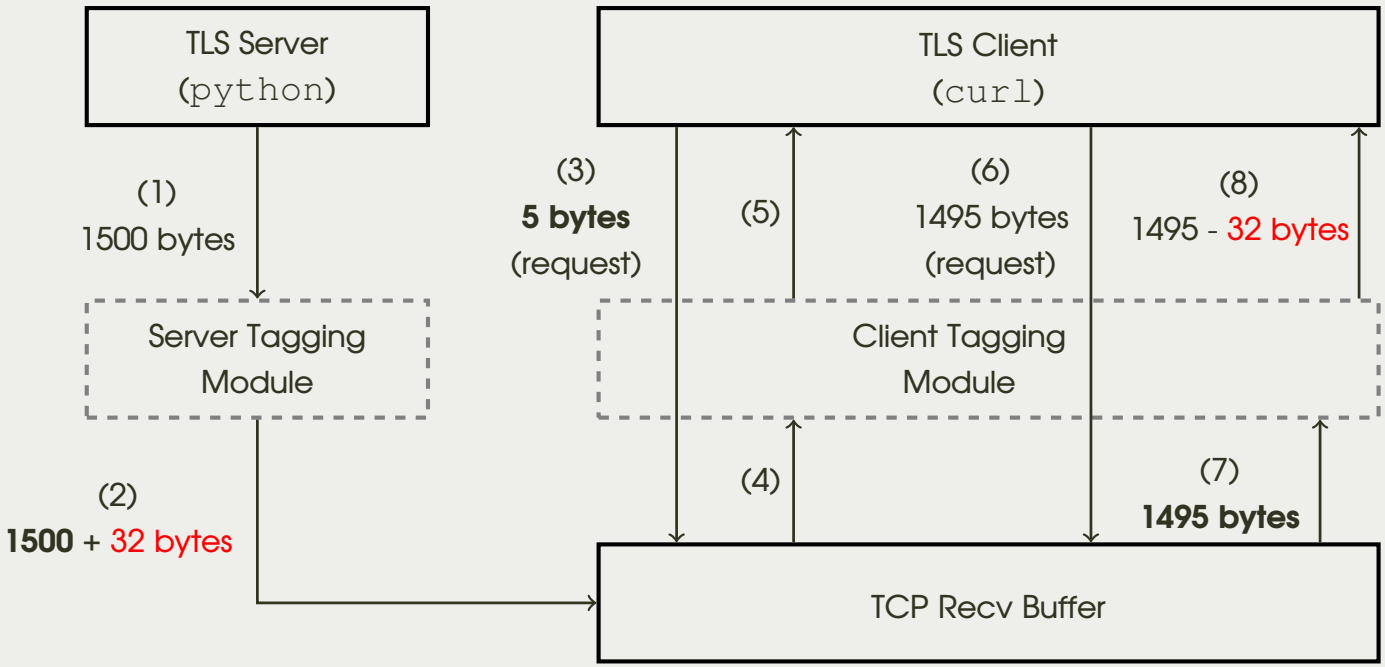
TLS without Customization



TLS with Customization

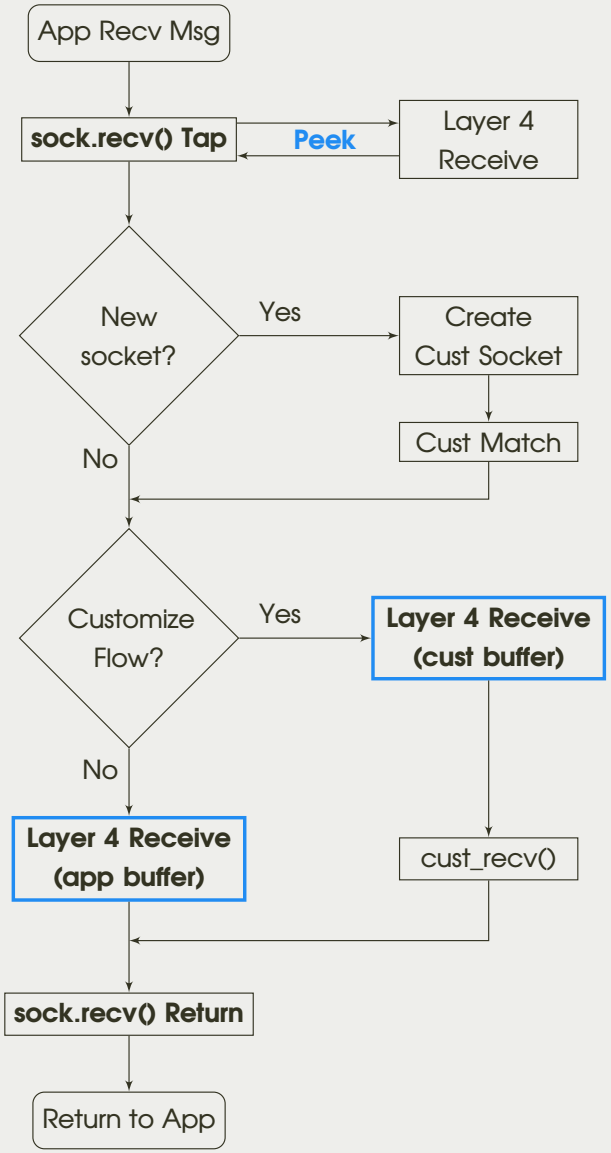
Goal: Add traffic classification tag every 1000 bytes

Problem:



Outline

Customization Module Buffering



Layer 4 Peek:

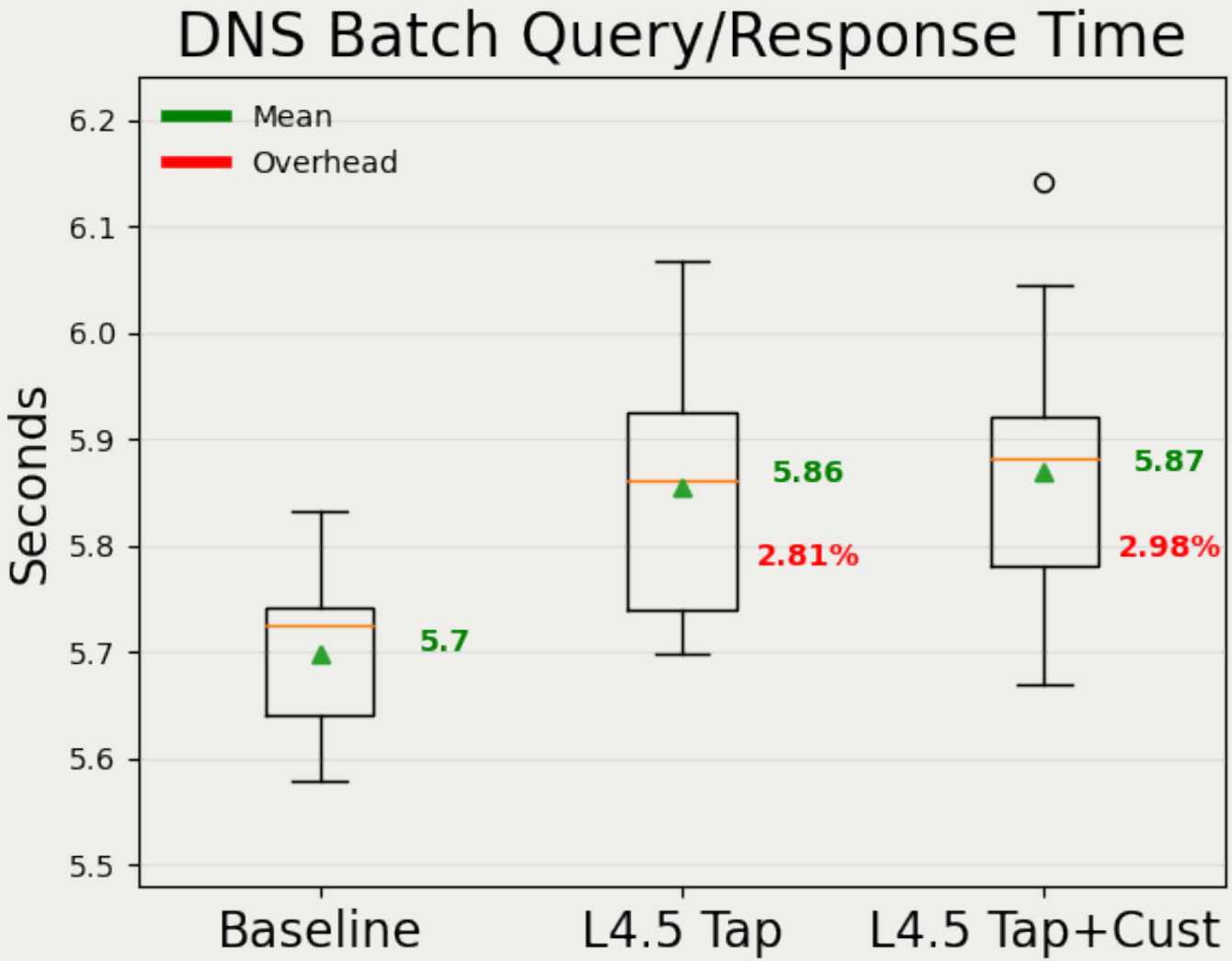
- Check if data present
- Assign socket values

Customization module receive buffer:

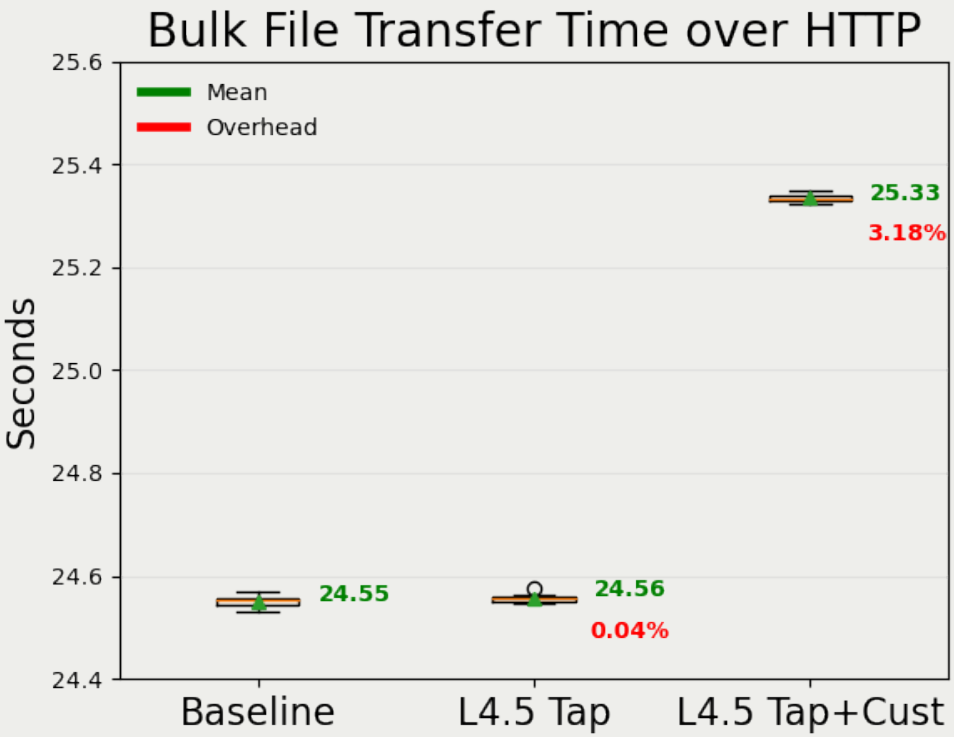
- Request more data than app
- Requires module to buffer data

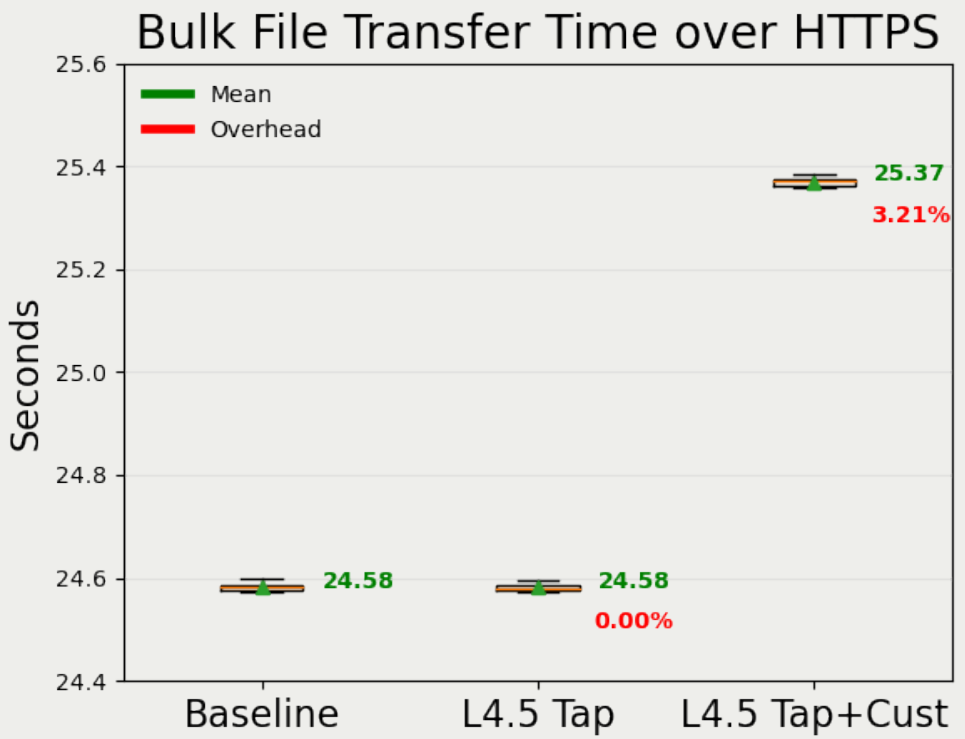
Outline

Processing Overhead: 1K Tag Insert/Delete Events



Processing Overhead: 3 M Tag Insert/Delete Events





Outline

Rotating Customizations on Active Flows

Rotation:

- Replacing a deployed customization module
- Automated on a normal interval (e.g., frequency hopping)

Active Flow:

- Application socket being used to send/recv traffic
- Socket is being customized

Purpose:

- Change customization without causing application errors
- Mission critical components remain available (i.e., no restart)

Challenges

Future Sockets:

- Only one customization module can match socket
- Revoking a module removes it from all sockets

Active Sockets:

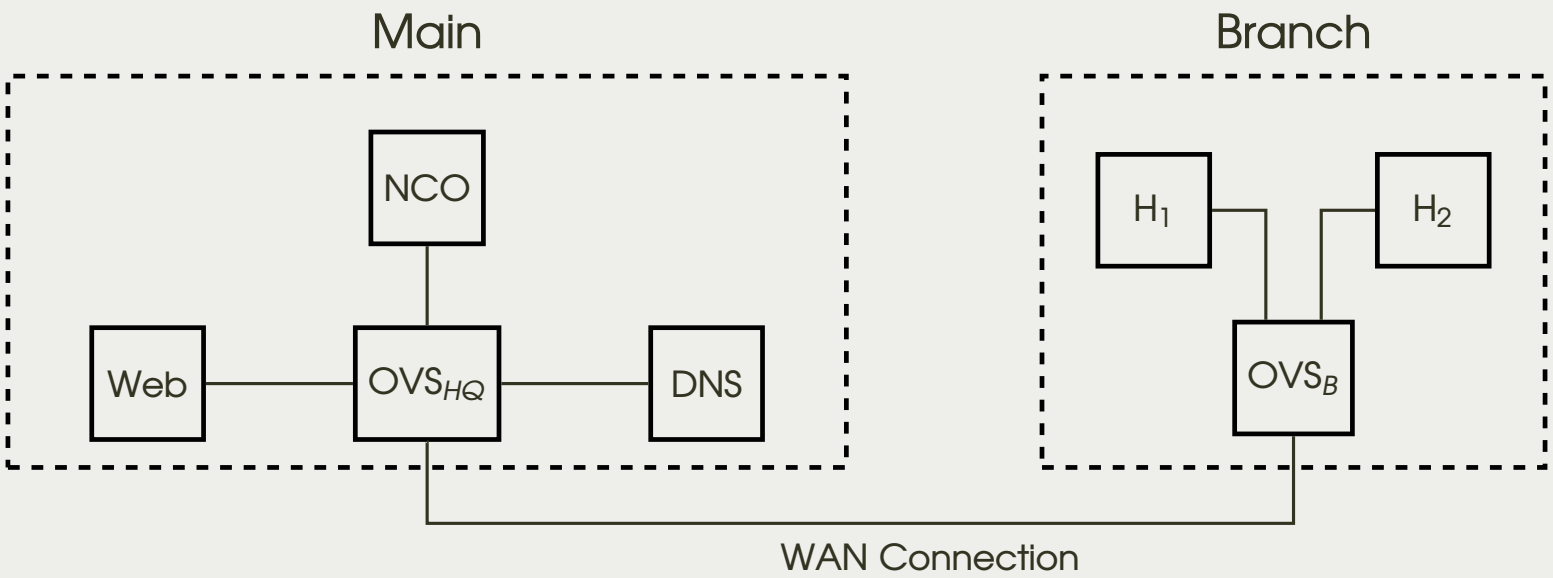
- Customization synchronization
- No method to attach to previously processed sockets

Outline

Architecture Extensions

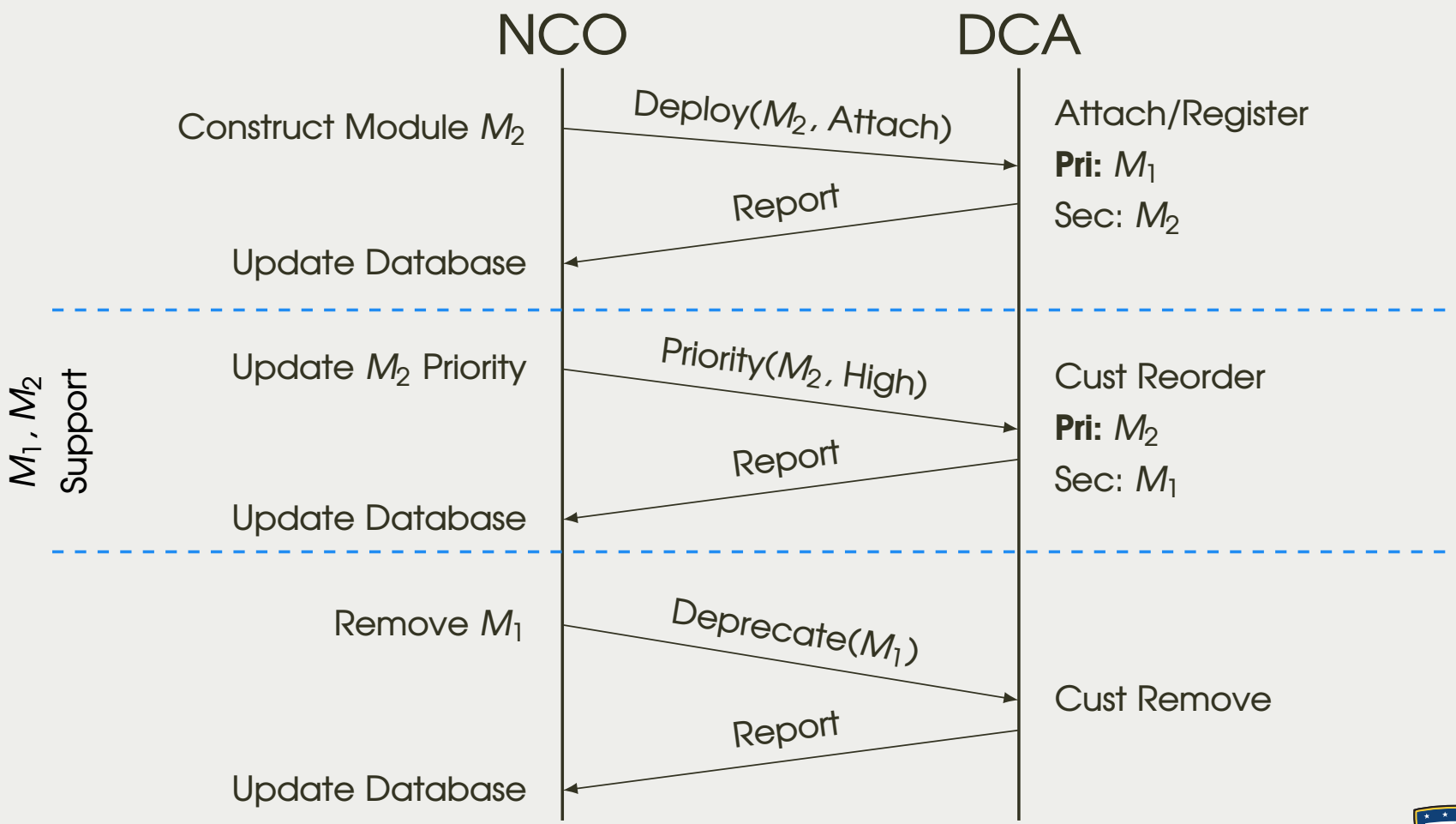
1. Customization Deprecation:
   - Module not usable for future socket matches
   - Remains on current sockets
2. Multiple Customization Attach:
   - Multiple modules can attach to a single socket
   - Only one module can actively customize flow
3. Priority:
   - Sort modules attached to a socket
4. Immediate Attach:
   - Re-check current sockets for customization match

Starting Assumptions



1. Customization module M_1 was previously deployed
   - High priority level
   - Non-deprecated
2. Customization module M_2 will be deployed with a lower priority

Rotation Process



Outline

Additional Operational Network Customizations

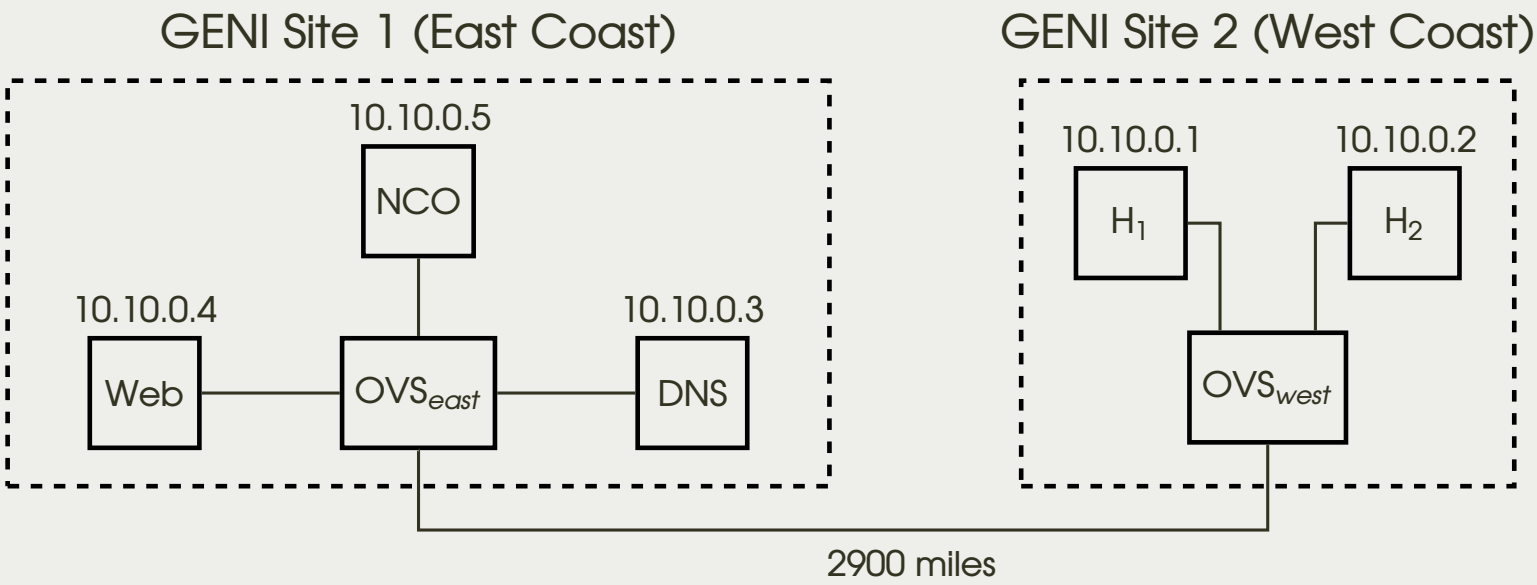
1. Rate-limiting: Add artificial delay
   - Brute force and DDoS attack mitigation
   - Web scraping

   Customization Module: Add 1000-msec delay to each send message

2. Link optimization: Reduce stress on network
   - WAN optimizers
   - MILCOM 2021

   Customization Module: DNS requests include ID and FQDN only

Customization Deprecation: Experiment Scenario



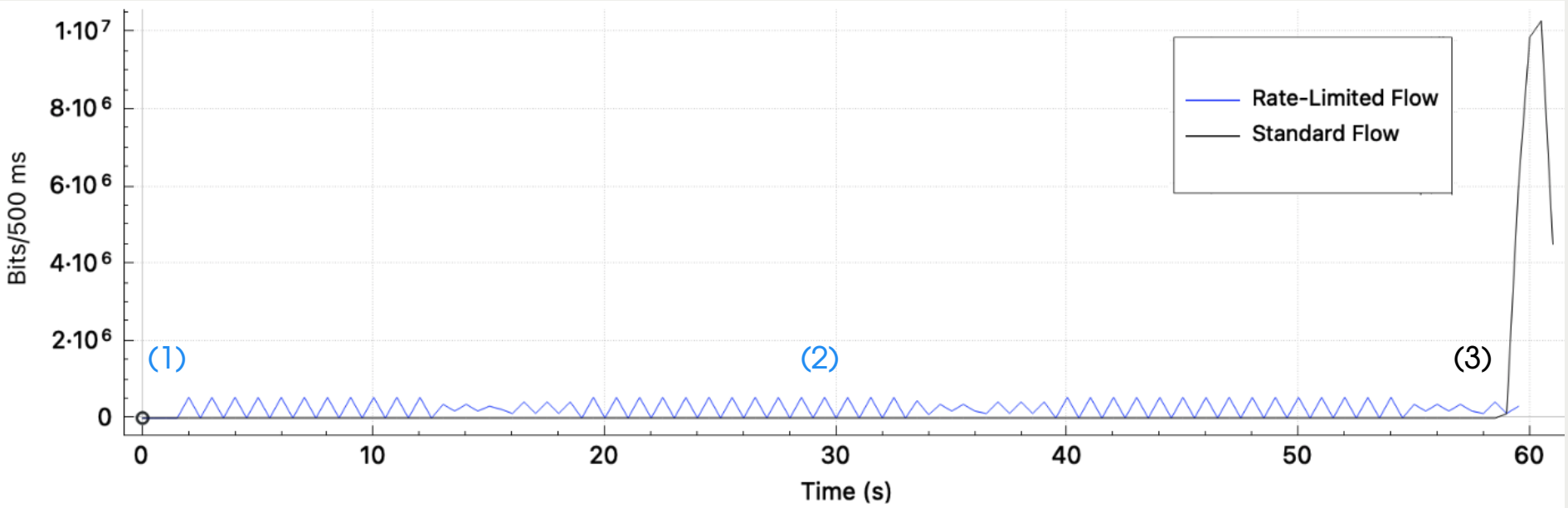
Starting condition:

- M_1 = rate limiting module
- Node = web server

Procedure:

- H_2 starts web transfer
- NCO deprecates module

Customization Deprecation: Throughput Correlation

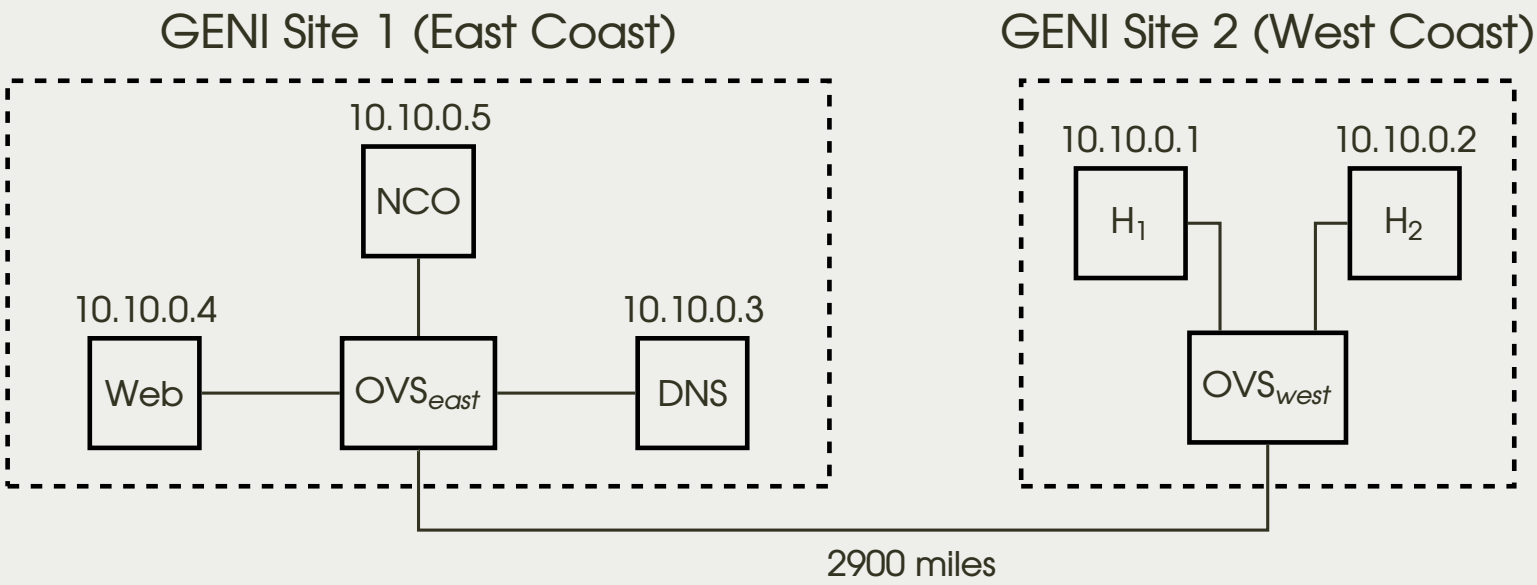


(1) : Customized flow start

(2) : Customization deprecated

(3) : Standard flow start

Customization Rotation: Experiment Scenario



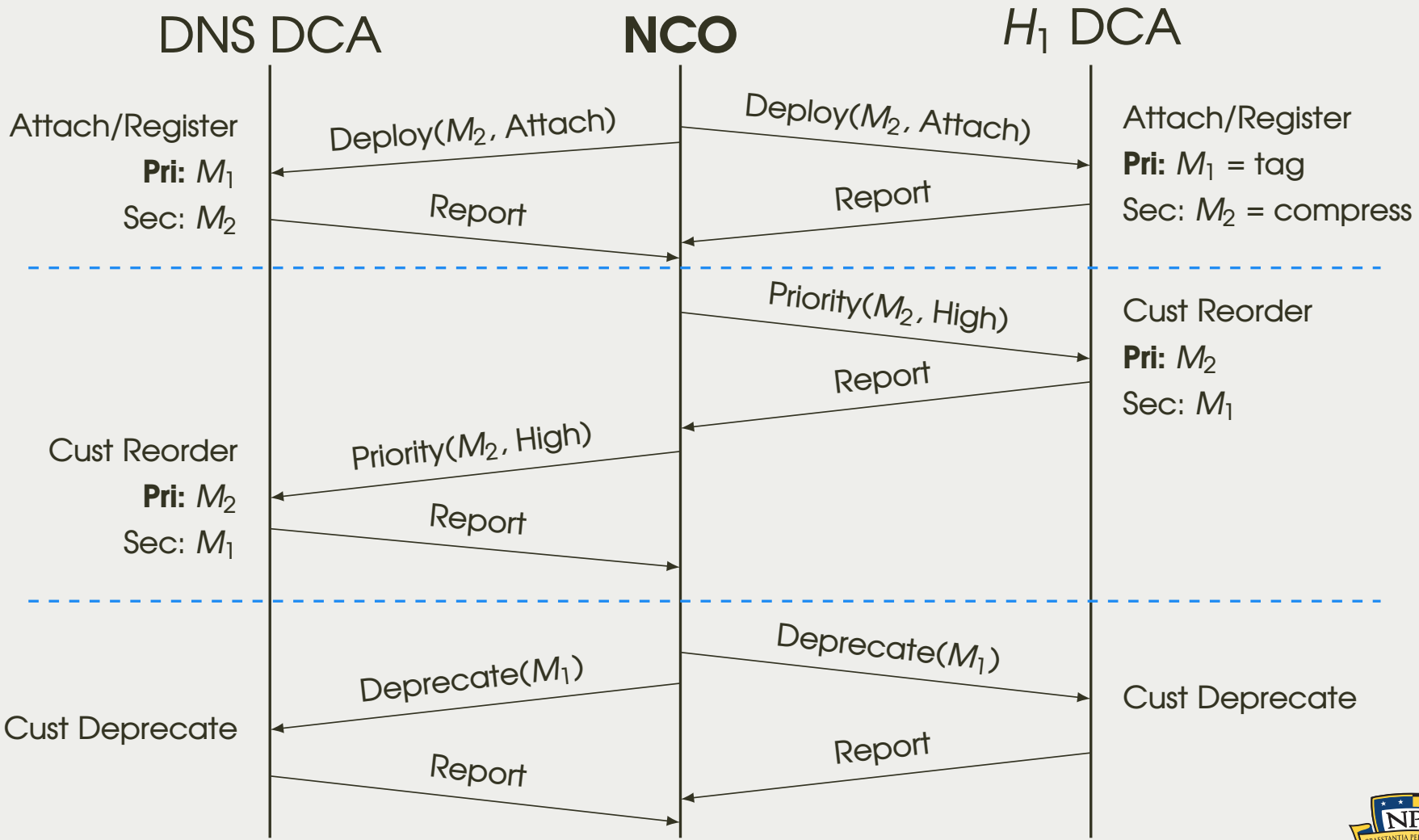
Starting condition:

- M_1 = DNS tagging
- Nodes = DNS server, H_1

Procedure:

- H_1 starts DNS request loop
- NCO conducts rotation
  - M_2 = DNS compression

Customization Rotation: Process



Rotation Process: DNS Server Log

```
   TASK-PID         TIMESTAMP  FUNCTION                 MESSAGE
      | |              |          |                       |
 insmod-22005   6915.385297:  register_cust:          L4.5: Registering module
 insmod-22005   6915.385299:  module_params:          Node protocol = 17
 insmod-22005   6915.385301:  module_params:          Node pid task = dnsmasq
 insmod-22005   6915.385301:  module_params:          Node tgid task = dnsmasq
 insmod-22005   6915.385302:  module_params:          Node id = 1
 insmod-22005   6915.385302:  module_params:          Node Activated = 1
 insmod-22005   6915.385303:  module_params:          Node Priority = 10
 insmod-22005   6915.385303:  module_params:          Node dest port = 0
 insmod-22005   6915.385304:  module_params:          Node source port = 53
 insmod-22005   6915.385306:  module_params:          Node dest_ip = 10.10.0.1
 insmod-22005   6915.385306:  module_params:          Node src_ip = 10.10.0.3
 insmod-22005   6915.385401:  cust_node:              L4.5: server front dns app tag module loaded, id=1


dnsmasq-6072    7014.943264:  update_cust_status:     L4.5: Assigning cust to socket, pid 6072
dnsmasq-6072    7014.943267:  assign_cust:           L4.5: assigning 1 nodes to socket
dnsmasq-6072    7014.943268:  assign_cust:           L4.5: socket cust[0] = Node id 1
```

Source	Destination	Protocol	App ID	XID	Request	Info
10.10.0.1	10.10.0.3	DNS	Xdig			Standard query 0x6afc A www.test_110.com OPT
10.10.0.3	10.10.0.1	DNS				Standard query response 0x6afc A www.test_110.com A 10.10.0.3
10.10.0.1	10.10.0.3	DNS	Xdig			Standard query 0x68dc A www.test_111.com OPT
10.10.0.3	10.10.0.1	DNS				Standard query response 0x68dc A www.test_111.com A 10.10.0.3
10.10.0.1	10.10.0.3	DNS	Xdig			Standard query 0xbef5 A www.test_112.com OPT
10.10.0.3	10.10.0.1	DNS				Standard query response 0xbef5 A www.test_112.com A 10.10.0.3
10.10.0.1	10.10.0.3	DNS	Xdig			Standard query 0x9ce6 A www.test_113.com OPT
10.10.0.3	10.10.0.1	DNS				Standard query response 0x9ce6 A www.test_113.com A 10.10.0.3

Rotation Process: DNS Server Log (Continued)

```
insmod-22015   7035.703489:  register_cust:          L4.5: Registering module
insmod-22015   7035.703492:  module_params:          Node protocol = 17
insmod-22015   7035.703493:  module_params:          Node pid task = dnsmasq
insmod-22015   7035.703493:  module_params:          Node tgid task = dnsmasq
insmod-22015   7035.703494:  module_params:          Node id = 3
insmod-22015   7035.703494:  module_params:          Node Activated = 1
insmod-22015   7035.703495:  module_params:          Node Priority = 20
insmod-22015   7035.703495:  module_params:          Node dest port = 0
insmod-22015   7035.703496:  module_params:          Node source port = 53
insmod-22015   7035.703498:  module_params:          Node dest_ip = 10.10.0.1
insmod-22015   7035.703499:  module_params:          Node src_ip = 10.10.0.3
insmod-22015   7035.703588:  set_update_cust_check:  L4.5 Cust Socket: Resetting pid 6072
insmod-22015   7035.703591:  cust_node:              L4.5: server dns compression module loaded, id=3
```

Source	Destination	Protocol	App ID	XID	Request	Info
10.10.0.1	10.10.0.3	L4.5_DNS		0x29de	test_114	45468 → 53 Len=24
10.10.0.3	10.10.0.1	DNS				Standard query response 0x29de A www.test_114.com A 10.10.0.3
10.10.0.1	10.10.0.3	L4.5_DNS		0xf809	test_115	51036 → 53 Len=24
10.10.0.3	10.10.0.1	DNS				Standard query response 0xf809 A www.test_115.com A 10.10.0.3

```
dnsmasq-6072   7081.699288:  cust_recv:              L4.5: DNS packet does not match custom pattern
dnsmasq-6072   7086.832761:  cust_recv:              L4.5: DNS packet does not match custom pattern
dnsmasq-6072   7091.966045:  cust_recv:              L4.5: DNS packet does not match custom pattern
dnsmasq-6072   7097.099618:  cust_recv:              L4.5: DNS packet does not match custom pattern
dnsmasq-6072   7102.232660:  cust_recv:              L4.5: DNS packet does not match custom pattern
dnsmasq-6072   7107.365791:  cust_recv:              L4.5: DNS packet does not match custom pattern
dnsmasq-6072   7112.498965:  cust_recv:              L4.5: DNS packet does not match custom pattern
dnsmasq-6072   7117.631986:  cust_recv:              L4.5: DNS packet does not match custom pattern
dnsmasq-6072   7122.765918:  cust_recv:              L4.5: DNS packet does not match custom pattern

python3-5966   7125.950551:  nl_recv_req:           L4.5: NLMSG_DATA = PRIORITY Node id 1 Level 30
```

Outline

Significant Contributions

1. First software defined customization architecture (NETSOFT 2022)
   - 1.1 per-process protocol customization
   - 1.2 per-network security controls
   - 1.3 aid middlebox traversal
2. Improved understanding and flexible support for application transparent customization (e.g., encrypted flows)
3. Using the new capabilities of our architecture, we are the first to demonstrate the previously unsupported capability for active flow customization rotation

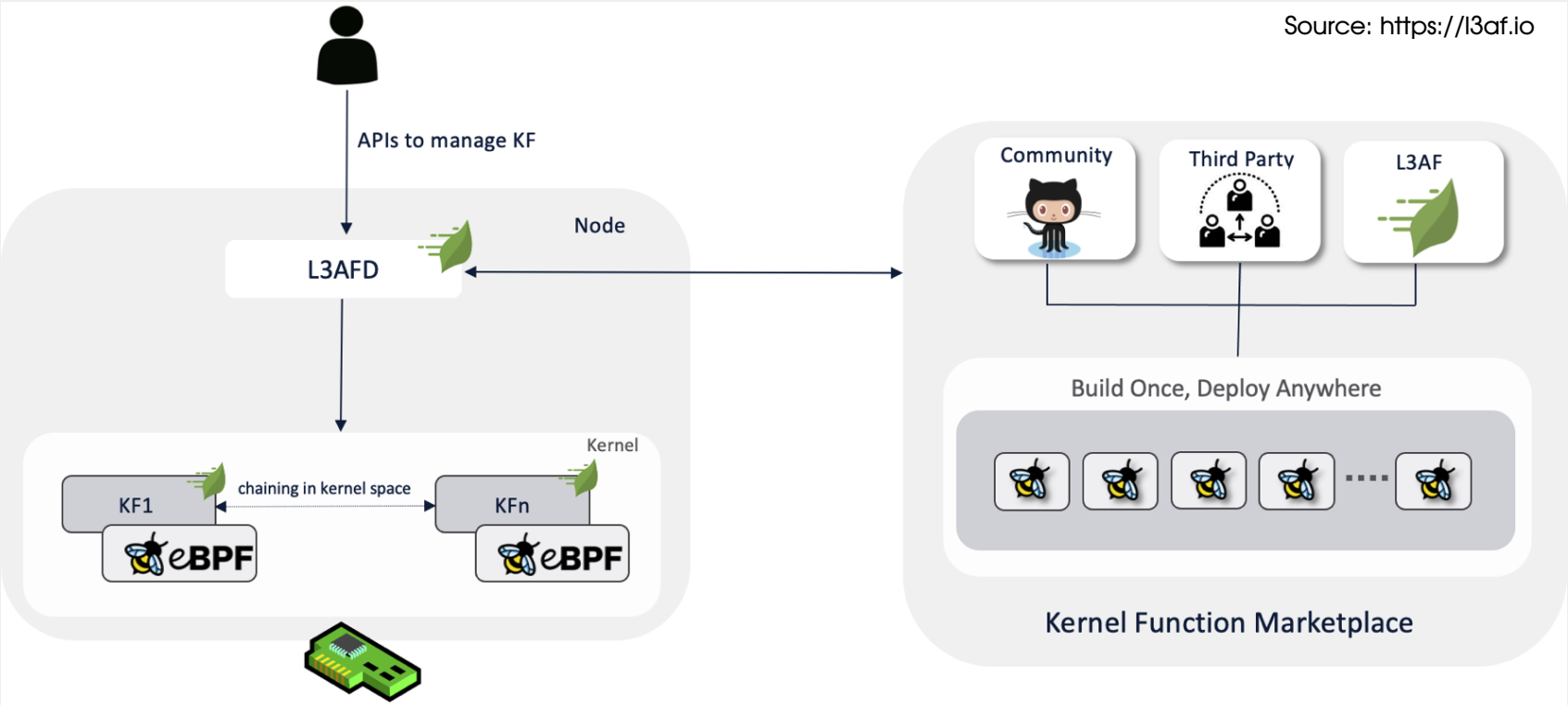
---

https://github.com/danluke2/software_defined_customization

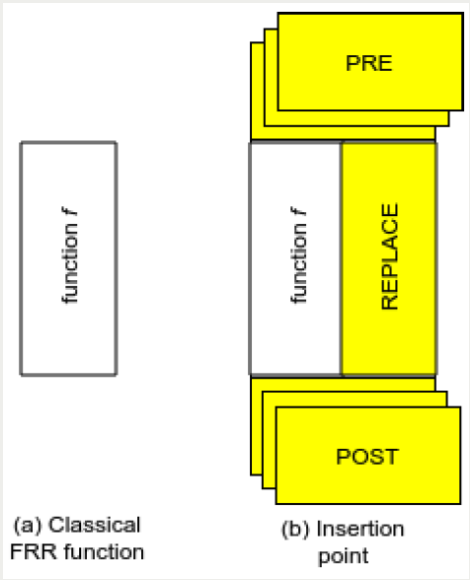
Thesis

A software defined Layer 4.5 protocol customization architecture is feasible and can provide continuous management capabilities, compatibility with modern encryption protocols, and rotating customizations on active application flows.

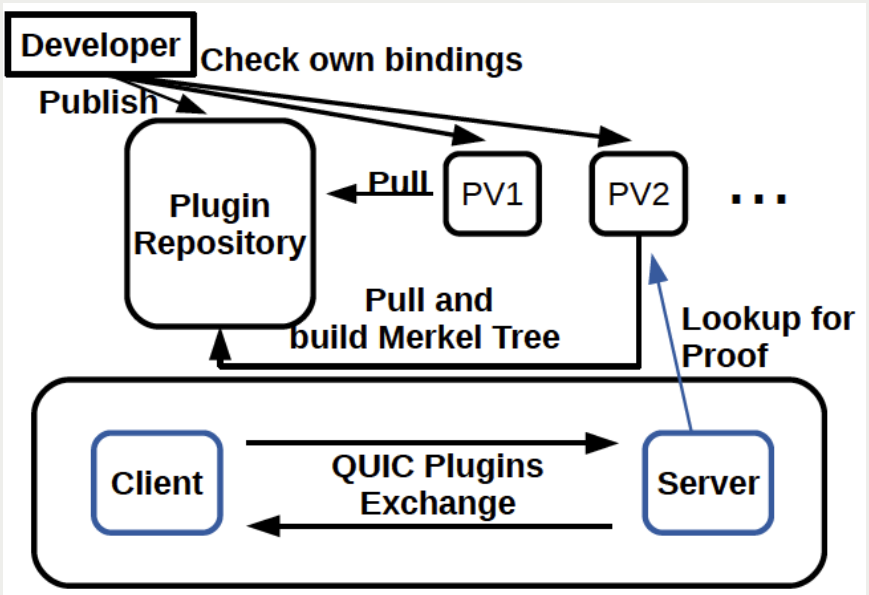
Customization Distribution: L3AF



Customization Distribution: Protocol Plugins



Source: Case for Plugin



Source: PQUIC

eBPF Complications

- Verifier
  - Must pass verification before execution permitted
  - Can't use kernel locks (multiprocessing)
  - Kernel sleep functions may also be limited
- Direct Packet Access
  - Limited to XDP and traffic control
  - Other hook points get copy of packet
- New helper functions or hook points require kernel modification
- Lower Layers:
  - Wasteful processing/Checksum recalculations
  - TCP Retransmits
  - TCP SYN/ACK modifications
  - IP fragmentation

Additional Insights

1. **PID Tracking:** The process (i.e., PID) that created the socket was not always the same process that closed the socket.
2. **Application Matching:** Attaching a customization module to a specific application may require using the name attached to the PID and/or the TGID.
3. **Middlebox Support:** Layer 4.5 must support different methods to perform inverse customizations because middlebox devices process customized flows at various layers.

Additional Insights

1. **TLS Dialect:** If a Man-in-the-Middle is present between two TLS devices, we may be able to detect the behavior using a customization module.

2. **Multiprocessing:** Applications that utilize multithreading and/or multiprocessing may further complicate customization module design.

3. **Blocking:** Application socket receive calls may wait until layer 4 has data. However, customization module may have all the remaining data buffered.

4. **Customization Complexity:** Even if the module does not require buffering capability, logic must still be present to handle it (e.g., DNS tagging).

Additional Insights

1. **Retransmits:** If we rotate a customization while TCP retransmits are occurring, then we could create customization mismatch condition.
2. **In-Order Arrival:** Packet ordering is not guaranteed for UDP traffic. Therefore, we need to support multiple customization modules for a set period of time during customization rotation.
3. **Customization Complexity:** Customization modules need to detect presence of customization (i.e., signature) and skip processing if not found

Future Work

- Prototype extensions
  - eBPF implementation
  - Different topologies
  - Various middleboxes
- Raise NCO abstraction
  - Control application on NCO
  - Baseline for other control applications with standard API
- TLS flow protection
  - Customization causes processing errors for normal hosts
  - Detect MITM attack
  - Customize TLS handshake

Open-Source Code Repository

- https://github.com/danluke2/software_defined_customization
  - NCO
  - DCA (user space, kernel space)
  - Middlebox DCA (user space)
  - Layer 4.5 modules
- Installation instructions
  - Vagrant pre-configured Virtualbox image
  - Sample modules to test installation success
- Experiment scripts for all experiments
- GENI testbed configuration files and startup scripts

Customization Deprecation: Layer 4.5 Log

```
    TASK-PID      TIMESTAMP  FUNCTION                 MESSAGE
       | |            |          |                      |
  insmod-22597   281.487102:  register_cust:          L4.5: Registering module
  insmod-22597   281.487105:  module_params:          Node protocol = 6
  insmod-22597   281.487106:  module_params:          Node pid task = python3
  insmod-22597   281.487106:  module_params:          Node tgid task = python3
  insmod-22597   281.487107:  module_params:          Node id = 1
  insmod-22597   281.487107:  module_params:          Node Activated = 1
  insmod-22597   281.487108:  module_params:          Node dest port = 0
  insmod-22597   281.487109:  module_params:          Node source port = 8080
  insmod-22597   281.487112:  module_params:          Node dest_ip = 0.0.0.0
  insmod-22597   281.487112:  module_params:          Node src_ip = 0.0.0.0
  insmod-22597   281.487113:  register_cust:          L4.5: Registration time: 1659032476
  insmod-22597   281.487114:  cust_node:              L4.5: server module loaded, id=1

                                                                                    (2)
python3-22581    313.454155:  create_cust_socket:    L4.5: Assigning cust to socket, pid 22581
python3-22581    313.454276:  cust_socket:           L4.5 tcp_accept: dest_ip:port=10.10.0.1:60924,
                                                     source_ip:port=10.10.0.4:8080, cust=true

                                                                                    (3)
python3-22576    341.681445:  nl_recv_req:           L4.5: NLMSG_DATA = DEPRECATE Node id 1
python3-22576    341.681451:  deprecate_cust:        L4.5: Removing module from use by new sockets
python3-22576    341.681463:  deprecate_cust:        L4.5: Deprecated time 1659032536

python3-22581    372.843220:  unassign_all:          L4.5: Cust removed, pid=22581, name=python3

                                                                                    (4)
python3-22581    372.843382:  new_inet_csk_accept:   L4.5: Customization skipped for pid 22581
python3-22581    372.843386:  cust_socket:           L4.5 tcp_accept: dest_ip:port=10.10.0.2:52404,
                                                     source_ip:port=10.10.0.4:8080, cust=false
```

(1)