BASTION: A Security Enforcement Network Stack for Container Networks

Jaehyun Nam†, Seungsoo Lee†, Hyunmin Seo†, Phillip Porras*, Vinod Yegneswaran*, Seungwon Shin†
KAIST, Daejeon, Korea†, SRI International, CA, USA*

USENIX Annual Technical Conference 2020 (ATC '20)

2024.09.24.

GyeongHeon Jeong(ghjeong@mmlab.snu.ac.kr)

Index

- Introduction
- Background
- Motivation
- BASTION
- Evaluation
- Conclusion

Introduction

- As large-scale deployments of containerized applications increase, so do the number of security vulnerabilities
  - This is because they prioritized deployment speed over the risk of deploying insecure containers

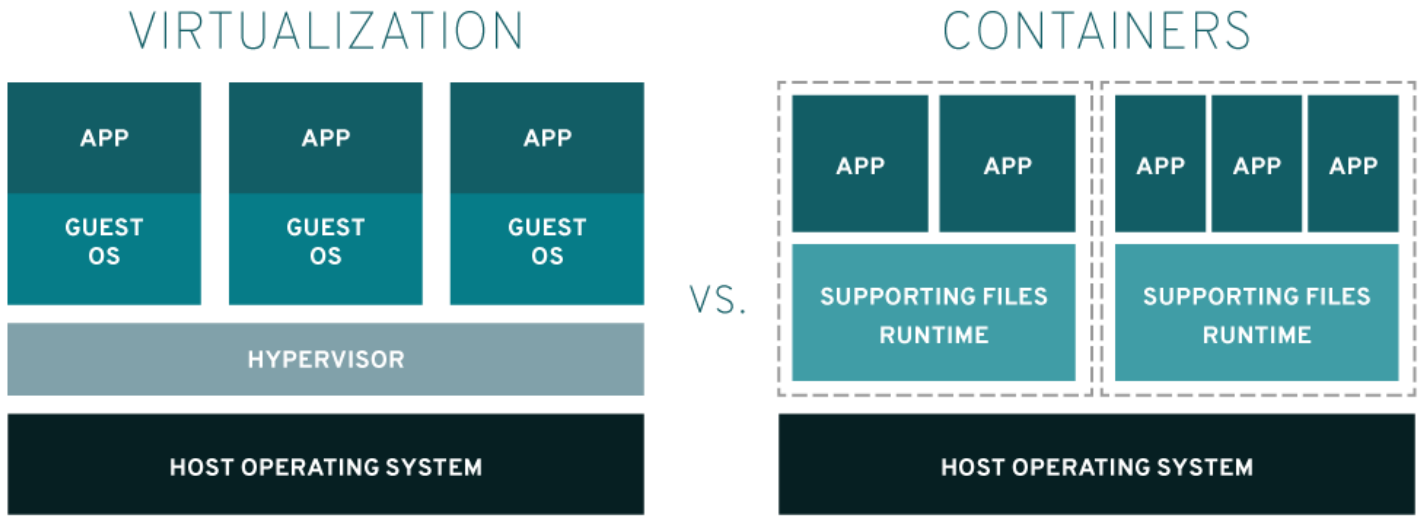
- Current container network interface plugins (e.g., Calico, Cilium, ...) is limited by a few security issues

- In response, authors introduced a new secure network stack called **BASTION**, which provides an isolated secure network stack for each container
  - **BASTION** can improve performance by up to 25.4% by controlling network communication between containers and enforcing security policies

Background - Container

- Virtualization
  - **Abstracts computer resources** to make what is physically one, such as an application, server, or storage device, appear to be many

- VM vs Container
  - Different way to virtualization
    - **Virtual machine**: Running on host operating system and has a guest operating system
    - **Container**: Sharing the Linux kernel and running independently with allocated system resources

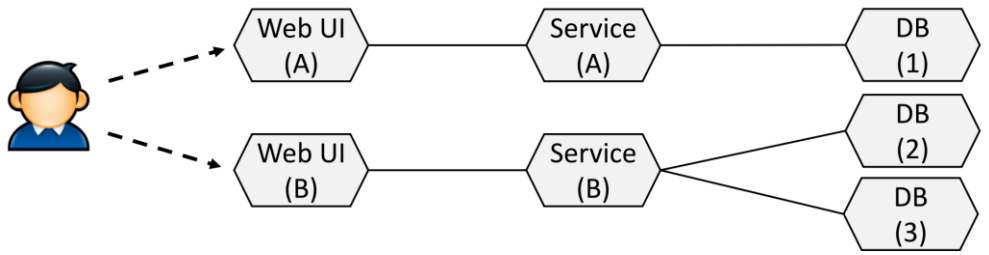


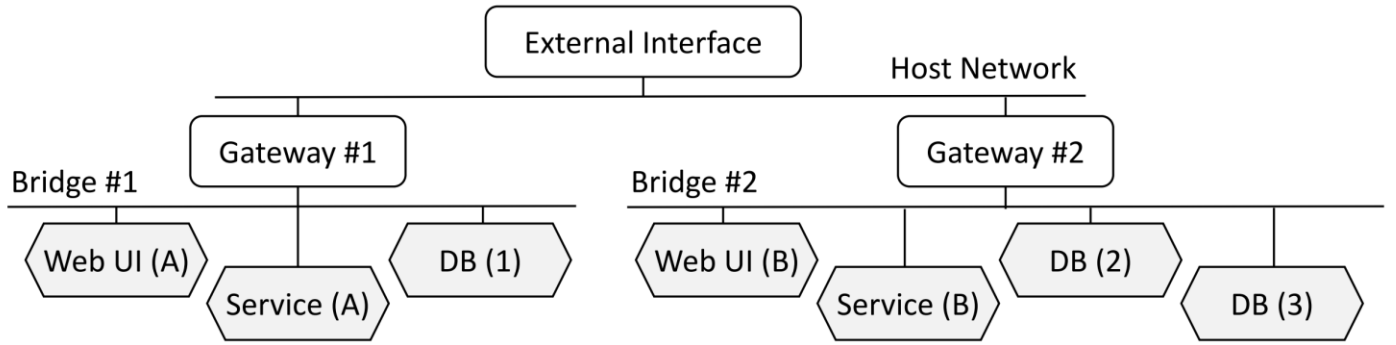
https://tech.ktcloud.com/

Background – Docker / Container Network

- Docker - Tool for using container
  - Open source project that automates the deployment of Linux applications into software **containers**
  - Image: Include all the files and configuration values needed to run the container

- Container network
  - Applies network and security policies into **bridge networks** using iptables



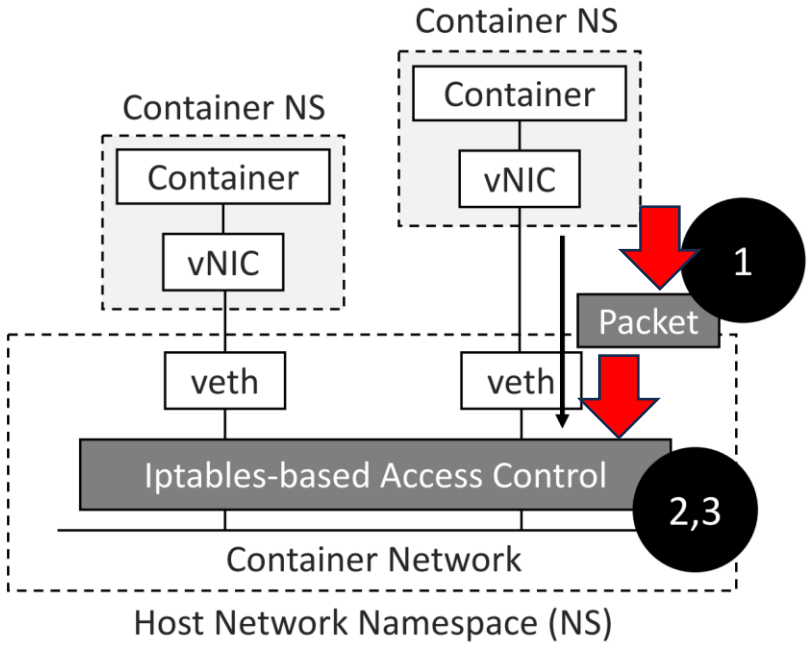


Security Challenges in Container Networks

1. Loss of container context (Authentication)
   - Do not know where packets actually come from in the host network namespace
   - Possible to forge packets on behalf of any other containers

2. Limitations of IP-based access controls (Performance)
   - Dynamically changed container IP addresses
   - Still vulnerable to L2 attacks due to limited scope

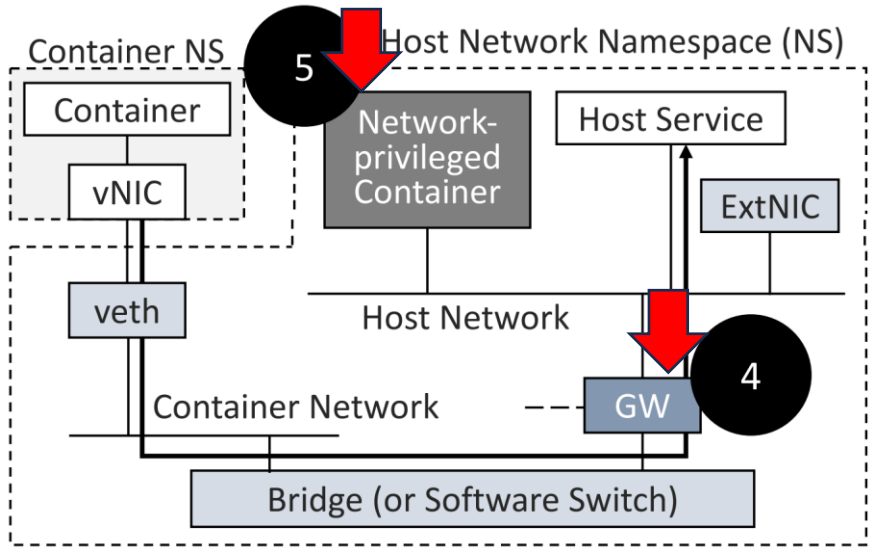
3. Network policy explosion (Performance)
   - Iptables: centralized mechanism for all network interfaces



Security Challenges in Container Networks (cont.)

4. Unrestricted host access (Authorization)
   - The gateway of a container network in the host network namespace
     - Necessary to access external networks
     - Allow accessing the services running at the host

5. Unrestricted network-privileged containers (Authorization)
   - Network-privileged containers
     - Share the host network namespace
   - All network interfaces under control
   - No solutions that consider security policies for them

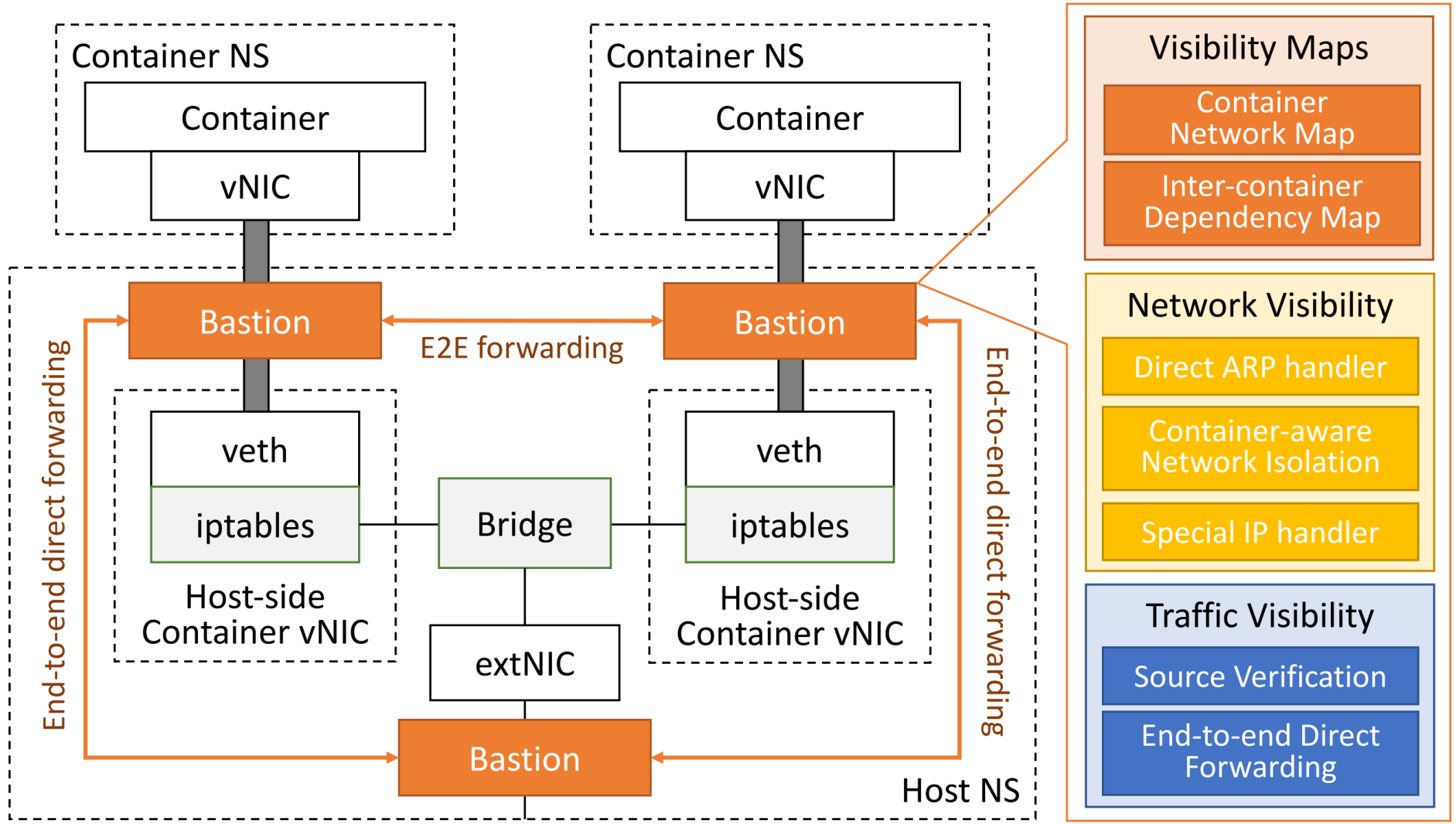


BASTION: Security Enforcement Network Stack

- Goal
  - Secure a container network through an intelligent container-aware communication sandbox
    - Protect network threats that abuse the security challenges of current container networks
    - Isolate inter-container communications according to their dependencies

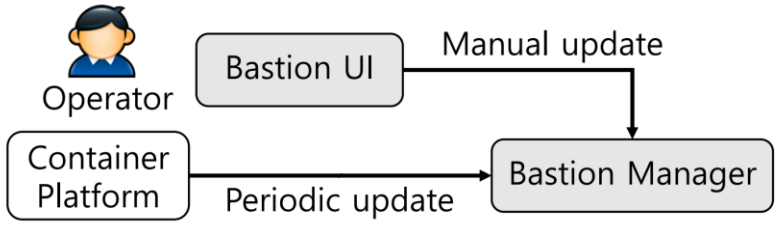
- Key Components
  - Bastion manager
    - Collect all network information (e.g., network configurations and policies) from container platforms
  - Network visibility service (Authorization, Performance)
    - Provide fine-grained control over different network topology visibility per container application
  - Traffic visibility service (Authentication, Performance)
    - Securely isolate inter-container communications in a point-to-point manner
    - Prevent the exposure of inter-container network traffic to other peer containers

Bastion Architecture



Bastion Manager

- Container network and dependency maps
  - Collect the network information of deployed containers from container platforms
  - Extract the inter-container dependencies using container configurations and network policies
- Security enforcement network stack
  - Install a security enforcement network stack at a newly deployed container
  - Update the maps of the security stack in run time



< Inter-container Dependency Map >

Source	Destination	Policy
WebApp	Service	Any
WebApp	Database	TCP:3306
Service	Database	TCP:3307

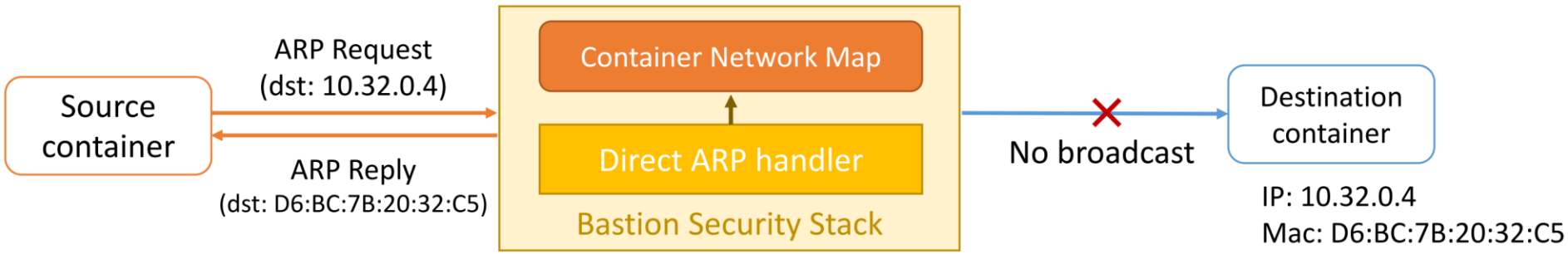
< Container Network Map >

Container	Network	Service	Interface	IP address	MAC address
WebApp-X1	WebService	WebApp	vethwepl6f964e8	10.32.0.2	96:0e:73:ef:86:fe
WebApp-X2	WebService	WebApp	vethweplb89dc35	10.32.0.3	6e:81:0f:a7:db:c7
Service-Y1	WebService	Service	vethweplb957e84	10.32.0.4	D6:bc:7b:20:32:c5
Database-Z1	WebService	Database	vethweplc5ee33c	10.32.0.5	42:a0:ae:b7:f5:97

Security Stack - Network Visibility Service

- Container Discovery
  - First step to identify other containers (targets for communication)
  - Possible to be exploited for **scanning all containers** by malicious containers
    - Current solutions: No prevention against non-IP-based communications

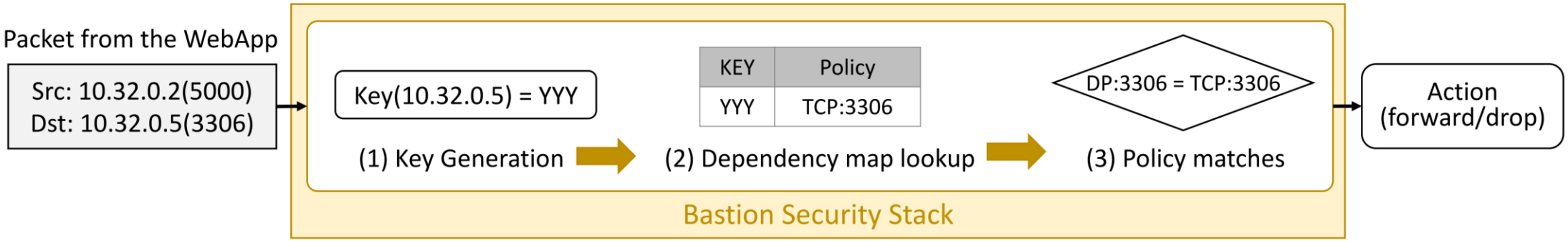
- Direct ARP handler
  - Directly response ARP requests at the security stack based on inter-container dependencies
    - Do not broadcast the requests to the network since this operation could be abused by an attacker



Security Stack - Network Visibility Service (cont.)

- Limitation of the direct ARP handler
  - Limited to container-level isolation (coverage issue)
  - Cannot address malicious network activities between **inter-dependent containers**

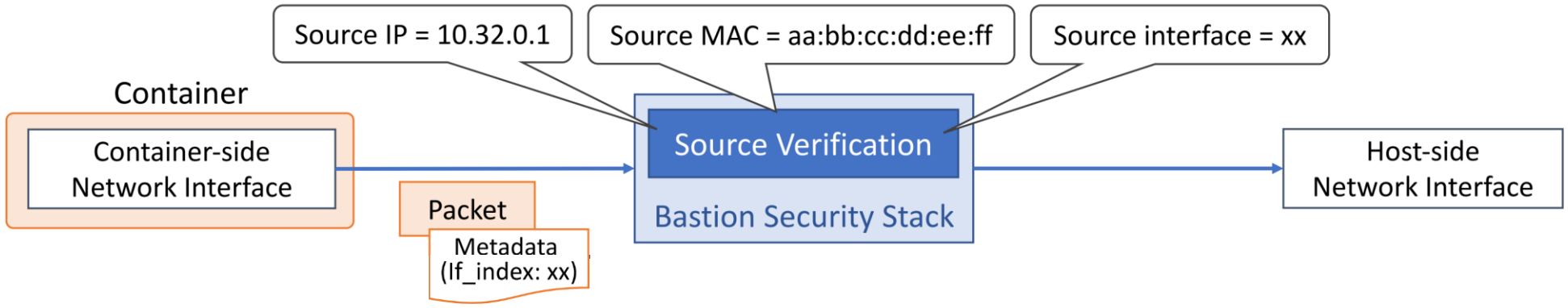
- Container-aware network isolation
  - Restrict container accesses according to finer-grained inter-container dependencies



Security Stack - Traffic Visibility Service

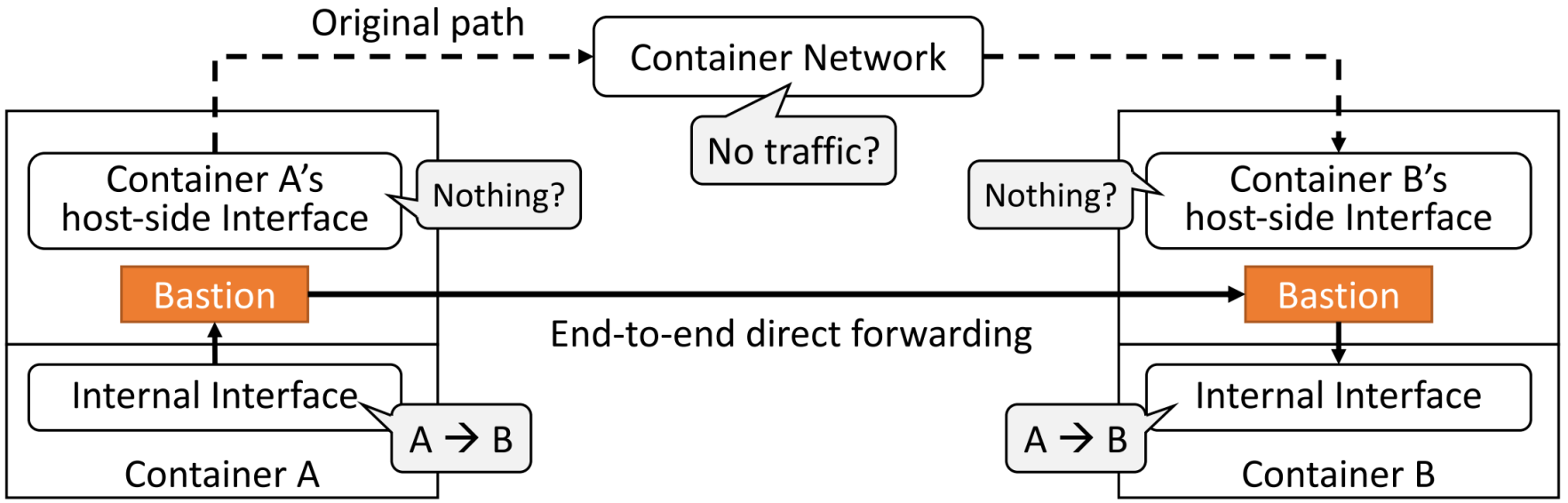
- How to verify sources in current solutions
  - Iptables: {source IP and MAC addresses} in packet headers
    - Vulnerable to Layer-2 attacks (e.g., ARP spoofing)

- Source verification in BASTION
  - {source IP and MAC addresses} in packet headers + kernel metadata at the container-side

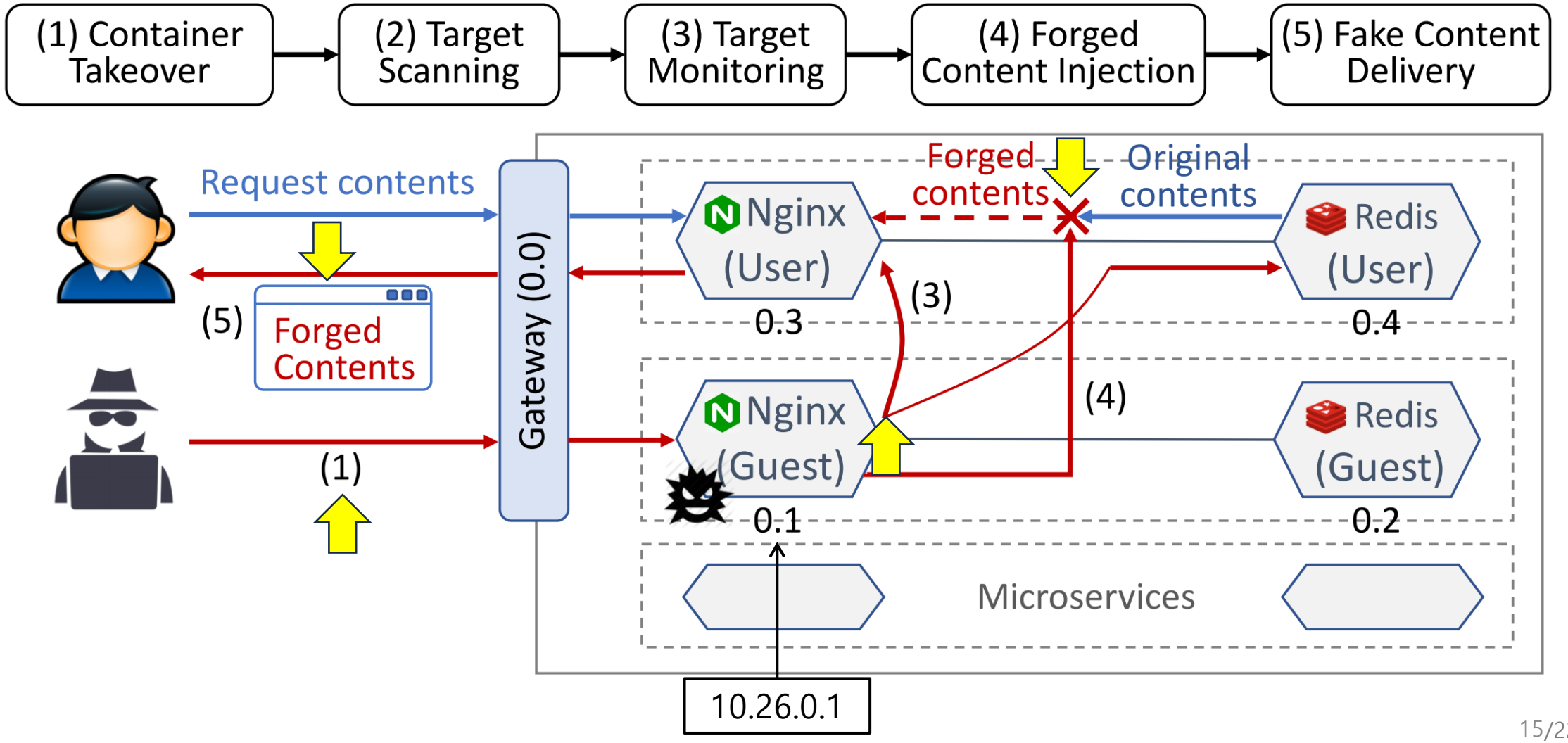


Security Stack - Traffic Visibility Service (cont.)

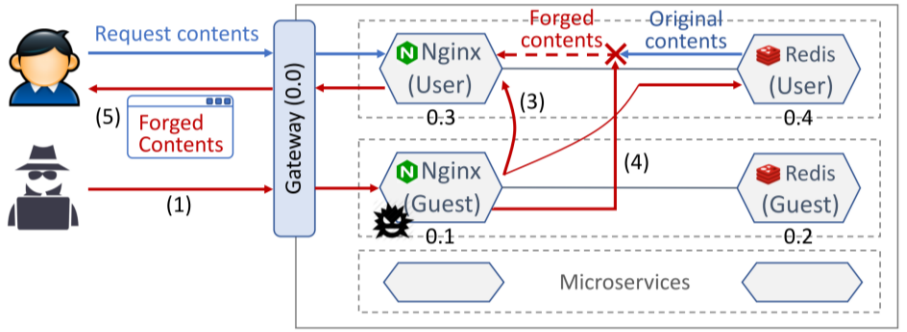
- End-to-end direct forwarding
  - Directly inject packets delivered from a source into a destination
  - Bypass the container network (the Linux network stack at the host-side)
  - Use the Linux networking features (XDP/eBPF)



Security Evaluation



Attack Scenario




```
root@attacker-64769f9f6d-gsmds:/tmp# ./arpping 10.46.0.0/24
Number of active containers : 55 → The number of all deployed containers
10.46.0.0, 96:0e:73:ef:86:fd      10.46.0.2, a2:80:27:eb:3d:c
10.46.0.3, 72:1d:6e:15:3b:1e      10.46.0.4, 9a:0e:fa:71:24:1
10.46.0.5, ce:a1:33:bf:e1:58      10.46.0.6, 9e:b2:69:ec:5d:e
10.46.0 Nginx-User 17:0b:f0:9a:20   The original MAC address of Redis-User
```

1. Scan neighbor containers in a network (Nginx-Guest view)

```
root@victim-1-fdd4f9f68-kwwtd:~# arp        The MAC address of Nginx-Guest
Address                  HWtype  HWaddress           Flags Ma
10.46.0.0                ether   96:0e:73:ef:86:fd   C
10.46.0.4 → Redis-User   ether   56:87:a7:15:17:69   C
10.46.0.1 → Nginx-Guest  ether   56:87:a7:15:17:69   C
kube-dns-86f4d74b45-zwp  ether   8e:1a:5d:bb:e8:c4   C
```

2. Spoof a target container (Nginx-User view)

```
50:11.375228 IP 10.46.0.3.40133 > 10.46.0.4.8000: Flags [S],
50:11.375279 IP 10.46.0.4.8000 > 10.46.0.3.40133: Flags [S.],
```

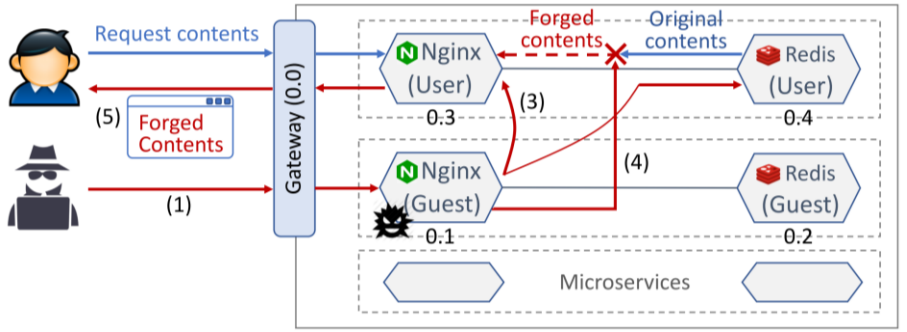
3. Eavesdrop the network traffic of the target (Nginx-Guest view)

Before injection

After injection

4. Inject a fake content (Client-side view)

Container Discovery



Before

```
root@attacker-64769f9f6d-gsmds:/tmp# ./arpping 10.46.0.0/24
Number of active containers : 55 → The number of all deployed containers
10.46.0.0, 96:0e:73:ef:86:fd      10.46.0.2, a2:80:27:eb:3d:c
10.46.0.3, 72:1d:6e:15:3b:1e      10.46.0.4, 9a:0e:fa:71:24:1
10.46.0.5, ce:a1:33:bf:e1:58      10.46.0.6, 9e:b2:69:ec:5d:e
10.46.0 Nginx-User 17:0b:f0:9a:20   The original MAC address of Redis-User
```

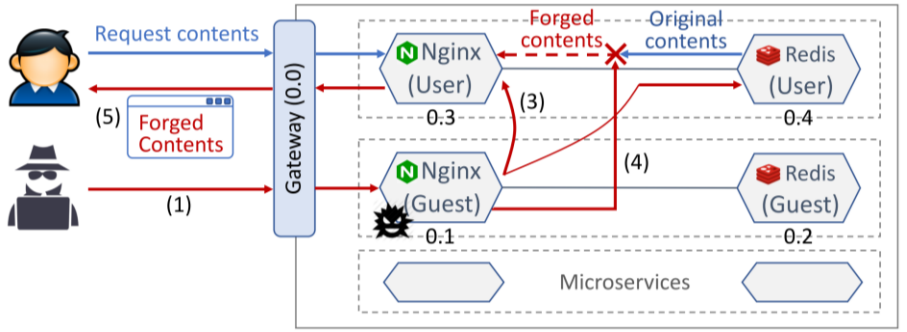
After

```
root@attacker-64769f9f6d-gsmds:/tmp# ./arpping 10.46.0.0/24
Number of active containers : 2 → The number of dependent containers
10.46.0.0, 96:0e:73:ef:86:fd      10.46.0.2, a2:80:27:eb:3d:c
root@attacker-64769f9f6d-gsmds:/tmp#
```

Gateway

Redis-Guest

Passive Packet Monitoring



Before

```
28:03.448899 IP 10.46.0.3.22167 > 10.46.0.4.8000: Flags [S],
28:03.448940 IP 10.46.0.3.22167 > 10.46.0.4.8000: Flags [S],
28:03.449027 IP 10.46.0.4.8000 > 10.46.0.3.22167: Flags [S.],
28:03.449047 IP 10.46.0.4.8000 > 10.46.0.3.22167: Flags [S.],
28:03.449155 IP 10.46.0.3.22167 > 10.46.0.4.8000: Flags [R],
28:03.449193 IP 10.46.0.3.22167 > 10.46.0.4.8000: Flags [R],
```

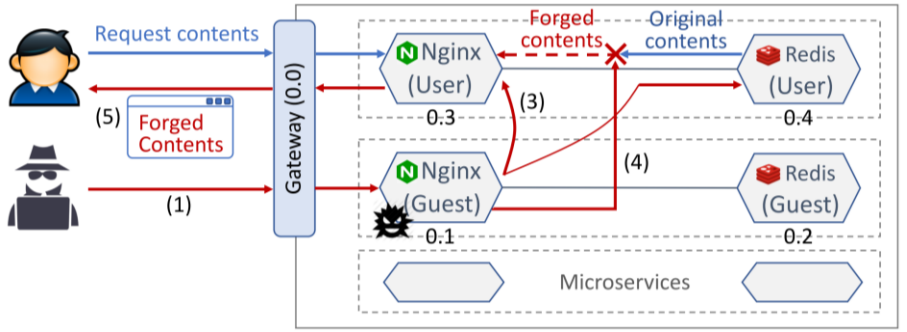
After

```
29:52.941051 IP 10.46.0.3.12119 > 10.46.0.4.8000: Flags [S],
29:52.941117 IP 10.46.0.3.12119 > 10.46.0.4.8000: Flags [S],
29:52.941338 IP 10.46.0.3.12119 > 10.46.0.4.8000: Flags [R],
29:52.941384 IP 10.46.0.3.12119 > 10.46.0.4.8000: Flags [R],
```

Disappear

With “full” bastion, all packet will disappear

Active Packet Injection



Before

```
20:21.353453 IP 10.46.0.4.8000 > 10.46.0.3.34104: Flags [R],
20:21.353456 IP 10.46.0.4.8000 > 10.46.0.3.34104: Flags [R],
```

< Attacker side >

```
20:21.353420 IP 10.46.0.3.34104 > 10.46.0.4.8000: Flags [.],
20:21.353460 IP 10.46.0.4.8000 > 10.46.0.3.34104: Flags [R],
```

< Victim side >

After

```
11:11.995745 IP 10.46.0.4.8000 > 10.46.0.3.12346: Flags [R],
11:11.995762 IP 10.46.0.4.8000 > 10.46.0.3.12346: Flags [R],
```

< Attacker side >

```
11:11.995614 IP 10.46.0.3.33452 > 10.46.0.4.8000: Flags [P.],
11:11.995655 IP 10.46.0.4.8000 > 10.46.0.3.33452: Flags [.],
11:11.995848 IP 10.46.0.4.8000 > 10.46.0.3.33452: Flags [.],
11:11.995866 IP 10.46.0.3.33452 > 10.46.0.4.8000: Flags [.],
```

< Victim side >

Dropped

Performance: Inter-container Throughputs

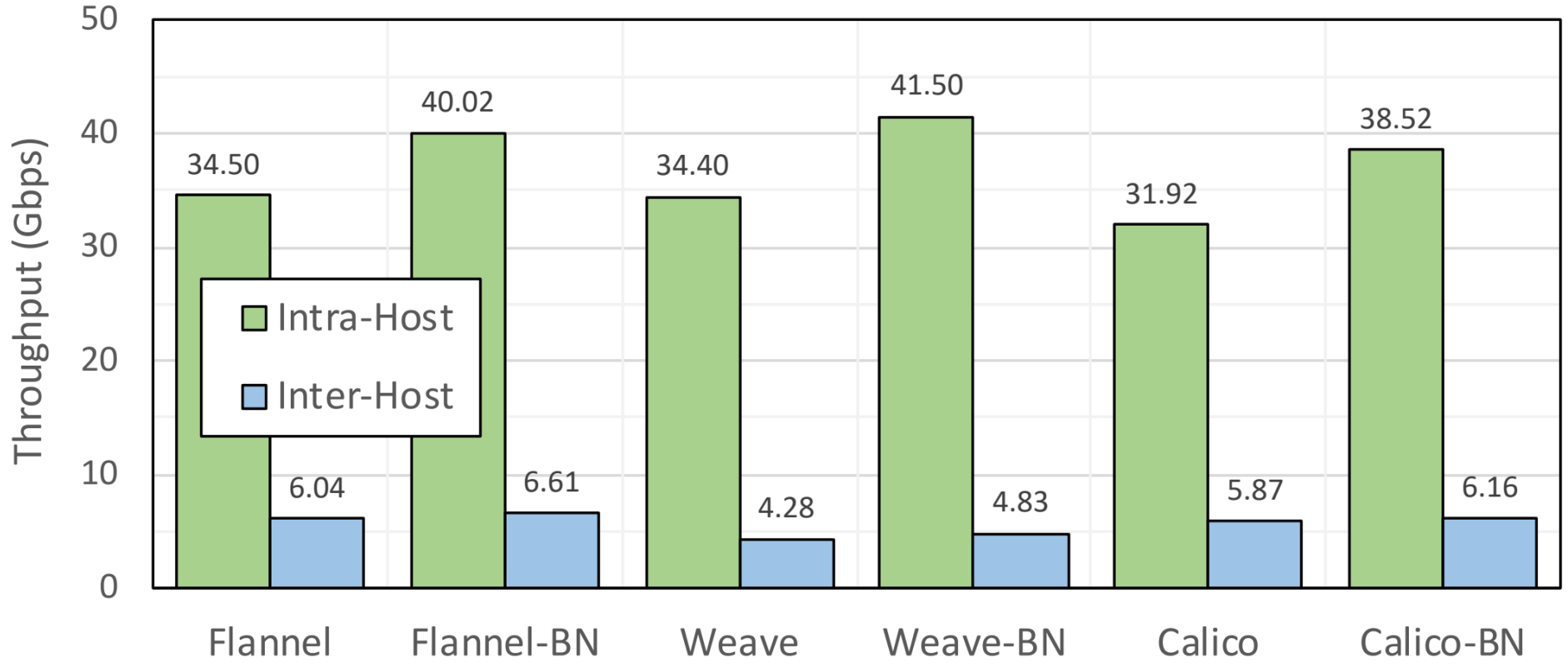
- Test environment
  - Xeon E5-2630v4 CPU with 64GB of RAM
  - Weave overlay networks, TCP traffic with iperf3

Throughput (Gbps)	Base (No Bastion)	Network Visibility only	Traffic Visibility only	Bastion (Fully deployed)
Within a host	34.4	33.7	41.8	41.5
Across hosts	4.28	4.23	4.91	4.83

  - Low inter-container throughputs across hosts?
    - Due to the heavy overheads in physical link traversal and tunneling between hosts

Performance: Bastion on Various Networks

- Test environment
  - Xeon E5-2630v4 CPU with 64GB of RAM
  - Flannel, Weave, and Calico overlay networks, TCP traffic with iperf3



Conclusion

- The state of current container security
  - Mostly focus on the security of containers themselves
  - Less concern the security issues in container networks

- Security assessment of container networks
  - Identified how the security challenges in current container networks impact containers

- Bastion: security enforcement network stack for container networks
  - Intelligently isolate inter-container communications
  - Effectively mitigate lateral attacks against peer containers

Thank you for listening