

# Socket Programming for SYN Flood Mitigation: Simulating Attacks and SCM Proxy Defense

Wei-Cheng Chiu, M11315038

Pin-An Lin, M11315016

Reference: Software-Defined Networking Integrated with Cloud Native and Proxy Mechanism: Detection and Mitigation System for TCP SYN Flooding Attack, 2023 17th International Conference on Ubiquitous Information Management and Communication (IMCOM), IEEE Explore



# Contents Overview

1 Motivation

2 System Architecture

3 Environment



4 Implementation

5 Experiment

# Motivation

1

## Growing Threat

TCP SYN Flooding accounts for over 54% of online attacks (Cloudflare, 2021).

2

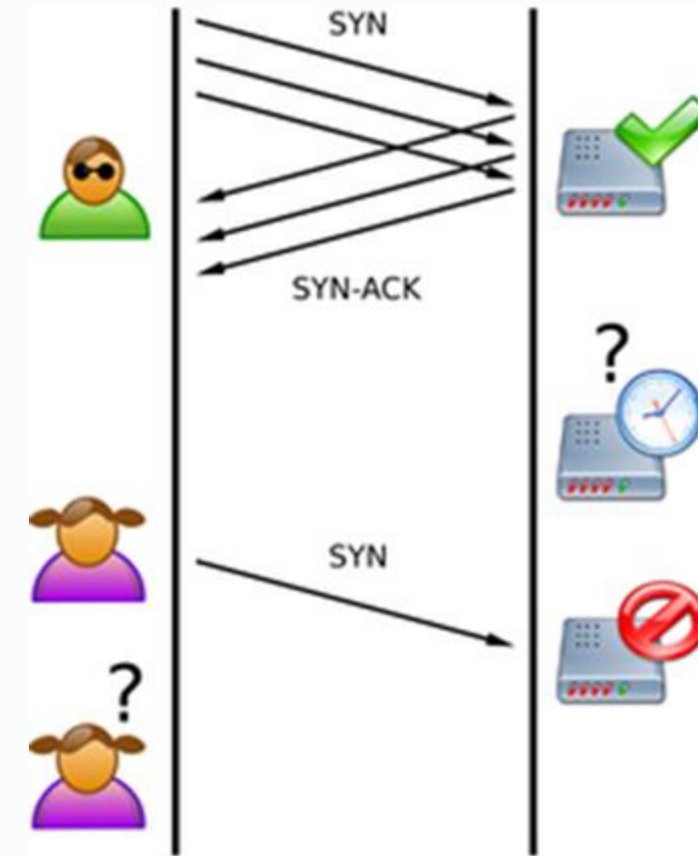
## Traditional Limitations

Conventional network architectures struggle to cope with large-scale DDoS attacks.

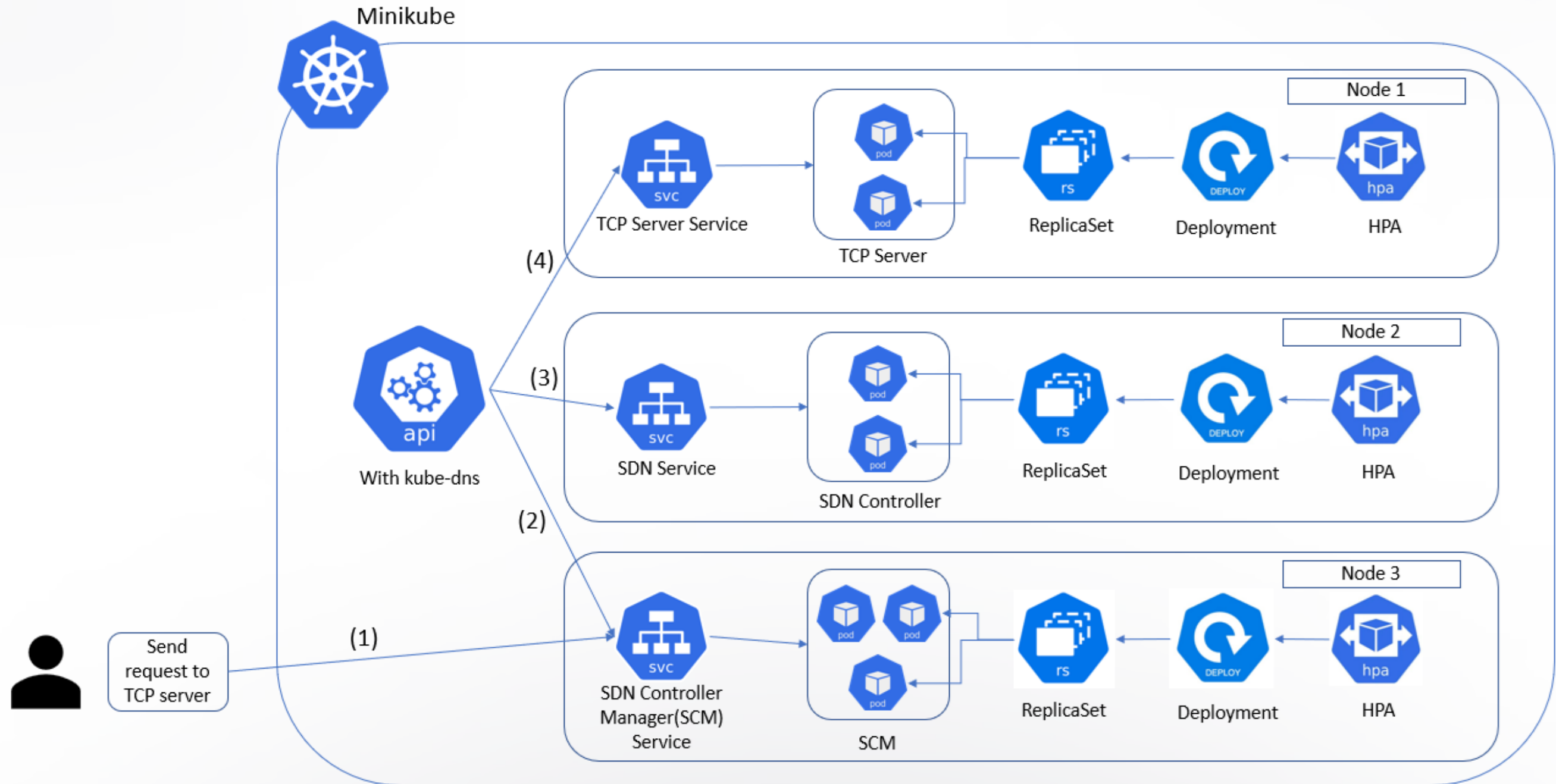
3

## Innovative Solution

Integrate SDN with Kubernetes for dynamic mitigation of SYN Flooding attacks.



# System Architecture



# Environment Setup Requirements



Docker Desktop with  
Kubernetes Enabled



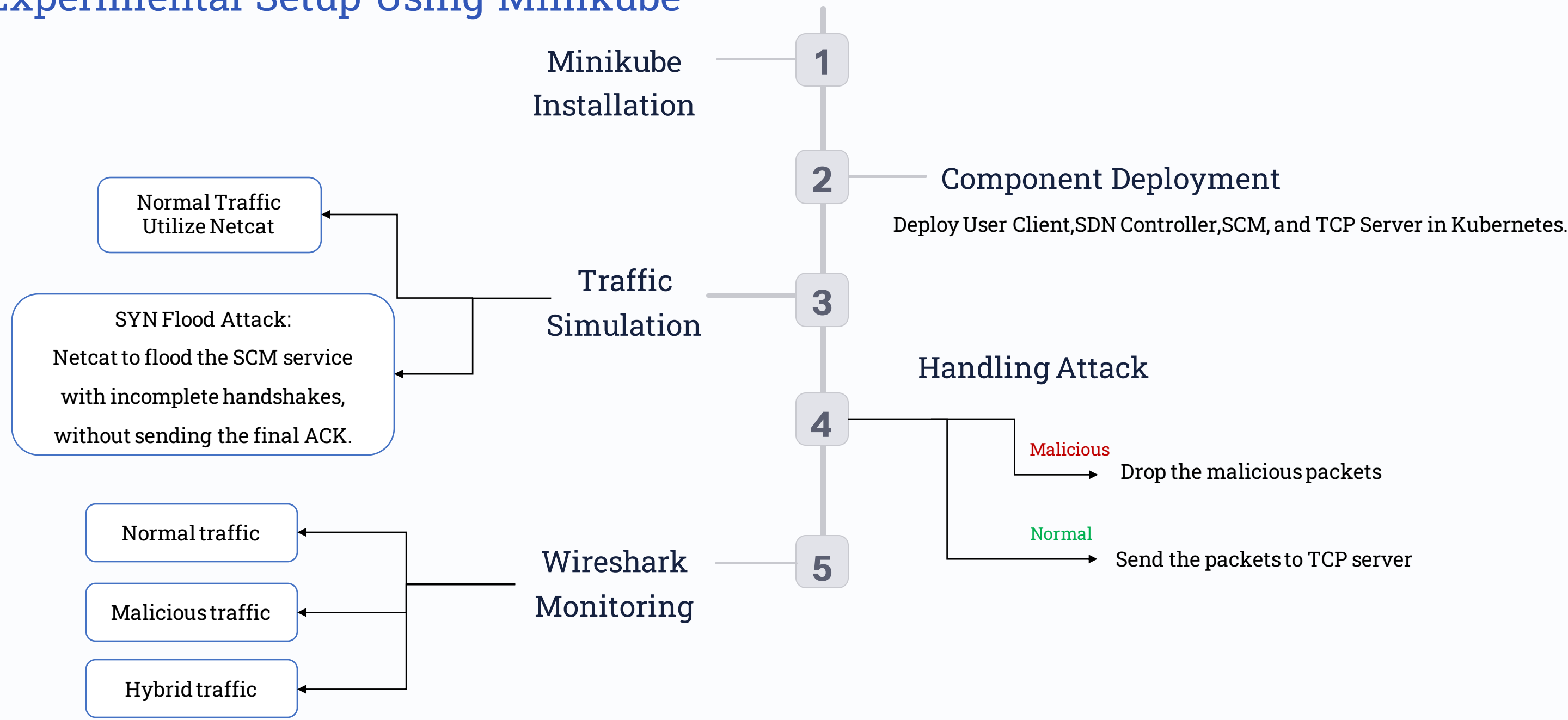
Minikube  
local k8s cluster



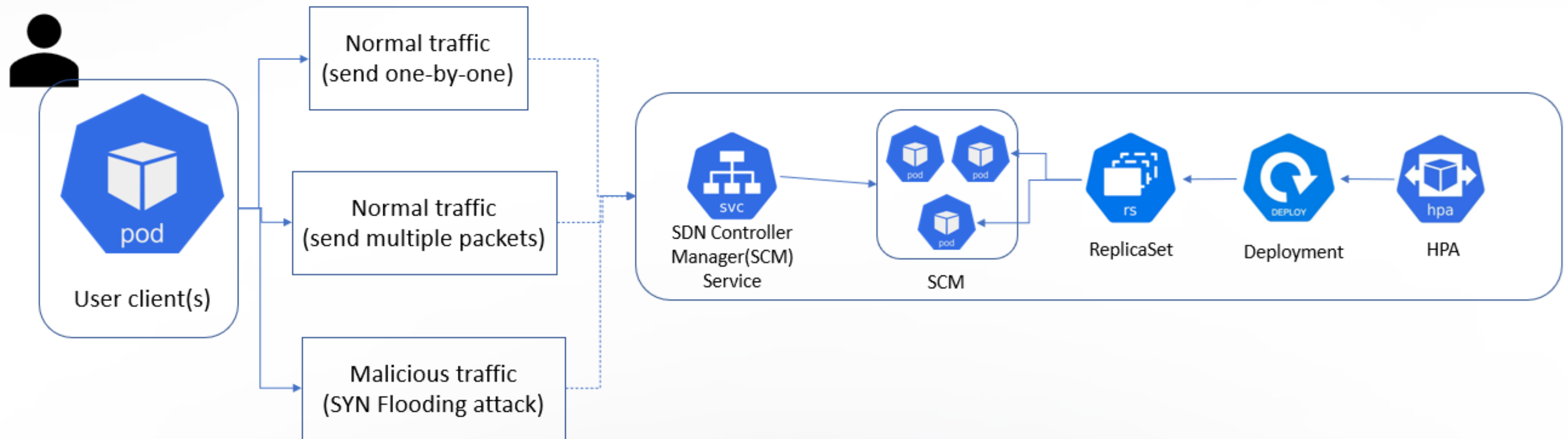
kubectl CLI Tool  
interface for k8s

# Implementation Methodology

## Experimental Setup Using Minikube



# Component Implementation: User-Client </>





# TCP-Server Implementation

```
1  apiVersion: apps/v1
2  kind: Deployment
3  metadata:
4    name: tcp-server
5  spec:
6    replicas: 1
7    selector:
8      matchLabels:
9        app: tcp-server
10   template:
11     metadata:
12       labels:
13         app: tcp-server
14     spec:
15       containers:
16       - name: tcp-server
17         image: allenlin316/tcp-server:v2
18         imagePullPolicy: Always
19         ports:
20         - containerPort: 9092
21         resources:
22           requests:
23             cpu: "100m"
24             memory: "128Mi"
25           limits:
26             cpu: "200m"
27             memory: "256Mi"
28         livenessProbe:
29           tcpSocket:
30             port: 9092
31           initialDelaySeconds: 15
32           periodSeconds: 20
33         readinessProbe:
34           tcpSocket:
35             port: 9092
36           initialDelaySeconds: 5
37           periodSeconds: 10
```

1

## Deployment Configuration

- The container exposes port 9092 for communication.
- Listens for SDN-controller connection

2

## Resource Allocation

- minimum:
  - 100m CPU
  - 128Mi memory
- maximum
  - 200m CPU
  - 256Mi memory.



# SCM-Proxy Implementation

**Three-way handshake  
validation**



Ensure secure communication between the user-client and the TCP-server.

**DDos Protection**



Monitors network traffic for suspicious patterns and identifies potential DDoS attacks.

**Traffic Filtering**



Filters incoming traffic based on predefined rules to block malicious requests and protect the TCP-server.

**Load Balancing**



Distributes incoming traffic across multiple TCP-server instances to improve performance and availability.

# SDN-Controller Integration

## 1 RYU SDN Controller

utilizes the RYU SDN controller for network management. It leverages the controller's capabilities for centralized control and programmability.

## 2 Custom Packet Routing

enables custom packet routing based on defined policies. This allows for flexible and dynamic routing of network traffic.

## 3 Network Policy Enforcement

enforces network policies to ensure secure and controlled communication within the network. It implements access control and traffic shaping rules.

```
1  apiVersion: apps/v1
2  kind: Deployment
3  metadata:
4    name: sdn-controller
5  spec:
6    replicas: 1
7    selector:
8      matchLabels:
9        app: sdn-controller
10   template:
11     metadata:
12       labels:
13         app: sdn-controller
14     spec:
15       containers:
16         - name: sdn-controller
17           image: allenlin316/sdn-controller:v2
18           imagePullPolicy: Always
19           ports:
20             - containerPort: 9091
21       resources:
22         requests:
23           cpu: "100m"
24           memory: "128Mi"
25         limits:
26           cpu: "200m"
27           memory: "256Mi"
28       livenessProbe:
29         tcpSocket:
30           port: 9091
31         initialDelaySeconds: 15
32         periodSeconds: 20
33       readinessProbe:
34         tcpSocket:
35           port: 9091
36         initialDelaySeconds: 5
37         periodSeconds: 10
38
```

# Auto-scaling Configuration

```
1  apiVersion: autoscaling/v2
2  kind: HorizontalPodAutoscaler
3  metadata:
4    name: scm-proxy-hpa
5  spec:
6    scaleTargetRef:
7      apiVersion: apps/v1
8      kind: Deployment
9      name: scm-proxy
10   minReplicas: 3
11   maxReplicas: 50
12   metrics:
13     - type: Resource
14       resource:
15         name: cpu
16         target:
17           type: Utilization
18           averageUtilization: 50
```

## minReplicas

- ❑ ensuring at least 3 replicas are always running

## maxReplicas

- ❑ preventing excessive scaling beyond this number

## averageUtilization

- ❑ targets 50% average CPU utilization for scaling, meaning the autoscaler will adjust replicas to maintain this average

```
SDN_simulation x SDN_simulation x User x + v
root@user-client:/app# python3 user-client.py
[Client] Connected to SCM Proxy.
[Client] Sending SYN...
[Client] Received: SYN-ACK
[Client] Sending ACK...
[Client] Enter a message (or 'exit' to quit): hello
[Client] Sending payload: hello
[Client] Received response: Response from TCP server
[Client] Enter a message (or 'exit' to quit): what up
[Client] Sending payload: what up
[Client] Received response: Response from TCP server
[Client] Enter a message (or 'exit' to quit): quit
[Client] Sending payload: quit
[Client] Server closed the connection.
root@user-client:/app# exit
exit
[User@Allen-TUF15: ~] D:\NTUST\SDN\final_project\SDN_simulation 59.897%
```

```
User x + v
Every 1.0s: kubectl.exe get all Allen-TUF15: Wed Nov 27 00:45:05 2024

NAME                                READY    STATUS    RESTARTS   AGE
pod/scm-proxy-866b487948-c58tq      1/1      Running   0           26m
pod/sdn-controller-5475b8559b-8gqkg 1/1      Running   0           8m10s
pod/tcp-server-6b9cb957d-4vf7j      1/1      Running   0           16m
pod/user-client                     1/1      Running   0           25m

NAME                                TYPE          CLUSTER-IP    EXTERNAL-IP  PORT(S)    AGE
service/kubernetes                  ClusterIP     10.96.0.1     <none>       443/TCP    44m
service/scm-proxy-service           ClusterIP     10.111.83.217 <none>       9090/TCP   30m
service/sdn-controller-service      ClusterIP     10.101.106.178 <none>       9091/TCP   8m10s
service/tcp-server-service          ClusterIP     10.106.106.208 <none>       9092/TCP   16m

NAME                                READY    UP-TO-DATE   AVAILABLE   AGE
deployment.apps/scm-proxy           1/1      1             1           26m
deployment.apps/sdn-controller      1/1      1             1           8m10s
deployment.apps/tcp-server          1/1      1             1           30m

NAME                                DESIRED    CURRENT    READY    AGE
replicaset.apps/scm-proxy-866b487948 1          1          1        26m
replicaset.apps/sdn-controller-5475b8559b 1          1          1        8m10s
replicaset.apps/tcp-server-5bdc7bb69b 0          0          0        17m
replicaset.apps/tcp-server-6686c4c8cd 0          0          0        30m
replicaset.apps/tcp-server-6b9cb957d 1          1          1        16m
```

```
Standard input [Default: scm-proxy-866b487948-c58tq-1000]
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

Apply a display filter: <Ctrl>F

No. Time Source Destination Protocol Length Info
1 0.000000 10.244.0.67 10.244.0.68 TCP 60 53914 → 9090 [WIN] Seq=642148 Len=0 RST=1406 SACK_PERM TSval=3433747469 TSecr=0 WS=128
2 0.000011 10.244.0.67 10.244.0.68 TCP 72 53914 → 9090 [ACK] Seq=642156 Len=0 TSval=3433747469 TSecr=382170393
4 0.000077 10.244.0.67 10.244.0.68 TCP 75 53914 → 9090 [PSH, ACK] Seq=642156 Len=3 TSval=3433747469 TSecr=382170393
5 0.000079 10.244.0.66 10.244.0.67 TCP 72 9090 → 53914 [ACK] Seq=642156 Len=0 TSval=3433747469 TSecr=3433747469
6 0.000123 10.244.0.66 10.244.0.67 TCP 79 9090 → 53914 [PSH, ACK] Seq=642156 Len=7 TSval=3433747469 TSecr=3433747469
7 0.000147 10.244.0.67 10.244.0.66 TCP 72 53914 → 9090 [ACK] Seq=642156 Len=0 TSval=3433747469 TSecr=382170393
8 0.000795 10.244.0.67 10.244.0.68 TCP 75 53914 → 9090 [PSH, ACK] Seq=642156 Len=3 TSval=3433747469 TSecr=382170393
9 0.000800 10.244.0.68 10.244.0.68 DNS 134 Standard query query 0x3872 A sdn-controller-service.default.vpc.cluster.local
10 0.000806 10.244.0.68 10.244.0.68 DNS 176 Standard query response 0x3872 A sdn-controller-service.default.vpc.cluster.local 10.101.106.178
11 0.000815 10.244.0.66 10.244.0.68 TCP 60 43390 → 9090 [WIN] Seq=642156 Len=0 RST=1406 SACK_PERM TSval=3440043901 TSecr=0 WS=128
12 0.001002 10.101.106.178 10.244.0.68 ARP 48 who has 10.244.0.67 Tell 10.244.0.1
13 0.001003 10.101.106.178 10.244.0.68 ARP 48 who has 10.244.0.67 Tell 10.244.0.1
14 0.001003 10.101.106.178 10.244.0.68 ARP 48 who has 10.244.0.67 Tell 10.244.0.1
15 0.001004 10.101.106.178 10.244.0.68 TCP 60 9090 → 43390 [WIN, ACK] Seq=642156 Len=0 RST=1406 SACK_PERM TSval=3440043901 TSecr=3440043901 WS=128
16 0.001005 10.244.0.66 10.101.106.178 TCP 72 43390 → 9090 [ACK] Seq=642156 Len=0 TSval=3440043901 TSecr=2340730549
17 0.001007 10.244.0.66 10.244.0.67 TCP 72 9090 → 53914 [ACK] Seq=642156 Len=0 TSval=3433747469 TSecr=3433747469
18 1.672033 10.244.0.67 10.244.0.68 TCP 77 53914 → 9090 [PSH, ACK] Seq=642156 Len=5 TSval=3433747469 TSecr=382170393
19 1.672040 10.244.0.67 10.244.0.67 TCP 72 9090 → 53914 [ACK] Seq=642156 Len=0 TSval=3433747469 TSecr=3433747469
20 1.672050 10.244.0.66 10.101.106.178 TCP 77 43390 → 9090 [PSH, ACK] Seq=642156 Len=5 TSval=3440043901 TSecr=2340730549
21 1.672051 10.101.106.178 10.244.0.68 ARP 48 who has 10.244.0.67 Tell 10.244.0.1
22 1.672059 10.101.106.178 10.244.0.68 ARP 48 who has 10.244.0.67 Tell 10.244.0.1
23 1.672069 10.101.106.178 10.244.0.68 ARP 48 who has 10.244.0.67 Tell 10.244.0.1
24 1.672074 10.101.106.178 10.244.0.68 ARP 48 who has 10.244.0.67 Tell 10.244.0.1
25 1.672122 10.101.106.178 10.244.0.68 TCP 76 9090 → 43390 [PSH, ACK] Seq=642156 Len=5 TSval=3440043901 TSecr=2340730549
26 1.672125 10.244.0.66 10.101.106.178 TCP 72 43390 → 9090 [ACK] Seq=642156 Len=0 TSval=3440043901 TSecr=2340730549
27 1.672128 10.244.0.66 10.244.0.68 TCP 60 9090 → 43390 [PSH, ACK] Seq=642156 Len=5 TSval=3440043901 TSecr=2340730549
```

# Network Traffic Monitoring



## Wireshark Integration

The **kubectl sniff** command allows for network traffic analysis using Wireshark. This provides a detailed view of network packets, enabling identification of potential issues or malicious activity.



## Resource Monitoring

Commands like **kubectl top pod** and **kubectl get hpa** provide insights into resource utilization and scaling behavior. This helps ensure optimal performance and identify potential bottlenecks.



## Log Analysis

The **kubectl logs** command enables examination of container logs, providing valuable information for troubleshooting and understanding application behavior. This helps identify errors, performance issues, or security threats.



# Demo Scenario 1: Normal Traffic

## Background Setting

- Pipeline for user sending to server:

user-client → SCM-proxy → SDN-controller → TCP-server

- Pipeline for server sending back to user:

TCP-server → SDN-controller → SCM-proxy → user-client

SDN-controller 10.244.0.201 forwarding network packets to TCP-server

NAME	READY	STATUS	RESTARTS	AGE	IP	NODE	NOMINATED NODE	READINESS GATES
scm-proxy-bd49cc696-2h8zj	1/1	Running	0	19m	10.244.0.203	minikube	<none>	<none>
scm-proxy-bd49cc696-7s252	1/1	Running	0	19m	10.244.0.202	minikube	<none>	<none>
scm-proxy-bd49cc696-8btcc	1/1	Running	0	19m	10.244.0.204	minikube	<none>	<none>
sdn-controller-7dd9c8d68f-z4r8g	1/1	Running	0	30m	10.244.0.201	minikube	<none>	<none>
tcp-server-6b766995d9-9tvhv	1/1	Running	0	36m	10.244.0.200	minikube	<none>	<none>
user-client	1/1	Running	1 (106m ago)	16h	10.244.0.189	minikube	<none>	<none>

No.	Time	Source	Destination	Protocol	Length	Info
80	0.056131	10.244.0.189	10.244.0.204	TCP	60	53258 → 9090 [SYN, ACK] Seq=64240 Win=0 MSS=1460 SACK_PERM TSval=572635957 TSecr=0 WS=128
81	0.056900	10.244.0.204	10.244.0.189	TCP	60	9090 → 53258 [SYN, ACK] Seq=64240 Win=0 MSS=1460 SACK_PERM TSval=572635957 TSecr=0 WS=128
82	0.056900	10.244.0.204	10.244.0.189	TCP	72	53258 → 9090 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=572635958 TSecr=1565819179
83	0.056900	10.244.0.204	10.244.0.189	TCP	75	53258 → 9090 [PSH, ACK] Seq=1 Ack=1 Win=64256 Len=3 TSval=572635958 TSecr=1565819179
84	0.056900	10.244.0.204	10.244.0.189	TCP	72	9090 → 53258 [ACK] Seq=1 Ack=4 Win=65280 Len=0 TSval=1565819179 TSecr=572635958
85	0.056900	10.244.0.204	10.244.0.189	TCP	79	9090 → 53258 [PSH, ACK] Seq=1 Ack=4 Win=65280 Len=7 TSval=1565819179 TSecr=572635958
86	0.056900	10.244.0.204	10.244.0.189	TCP	72	53258 → 9090 [ACK] Seq=4 Ack=8 Win=64256 Len=0 TSval=572635958 TSecr=1565819179
87	0.056900	10.244.0.204	10.244.0.189	TCP	75	53258 → 9090 [PSH, ACK] Seq=4 Ack=8 Win=64256 Len=3 TSval=572635958 TSecr=1565819179
88	0.056900	10.244.0.204	10.244.0.189	DNS	114	Standard query 0xecef A sdn-controller-service.default.svc.cluster.local
89	0.056900	10.244.0.204	10.244.0.189	DNS	178	Standard query response 0xecef A sdn-controller-service.default.svc.cluster.local A 10.96.100.54
90	0.056900	10.244.0.204	10.244.0.189	TCP	60	43892 → 9091 [SYN] Seq=64240 Win=0 MSS=1460 SACK_PERM TSval=2457694970 TSecr=0 WS=128
91	0.056900	10.244.0.204	10.244.0.189	TCP	80	9091 → 43892 [SYN, ACK] Seq=64240 Win=0 MSS=1460 SACK_PERM TSval=2457694970 TSecr=2457694970 WS=128
92	0.056900	10.244.0.204	10.244.0.189	TCP	72	43892 → 9091 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=2457694970 TSecr=2457694970
93	0.056900	10.244.0.204	10.244.0.189	TCP	72	9090 → 53258 [ACK] Seq=8 Ack=7 Win=65280 Len=0 TSval=1565819230 TSecr=572635958
94	0.056900	10.244.0.204	10.244.0.189	TCP	77	53258 → 9090 [PSH, ACK] Seq=7 Ack=8 Win=64256 Len=5 TSval=572638432 TSecr=1565819230
95	0.056900	10.244.0.204	10.244.0.189	TCP	72	9090 → 53258 [ACK] Seq=8 Ack=12 Win=65280 Len=0 TSval=1565821653 TSecr=572638432

```
root@user-client:/app# python3 user-client.py
[Client] Connected to SCM Proxy.
[Client] Sending SYN...
[Client] Received: SYN-ACK
[Client] Sending ACK...
[Client] Enter a message (or 'exit' to quit): hello
[Client] Sending payload: hello
[Client] Received response: Response from TCP server
[Client] Enter a message (or 'exit' to quit): what up
[Client] Sending payload: what up
[Client] Received response: Response from TCP server
[Client] Enter a message (or 'exit' to quit): |
```

- Showing packets being sent one by one from the **user-client pod**

- Showing user-client sending packets to TCP-server through SCM-proxy and SDN-controller with Wireshark

# Demo Scenario 2: Normal Traffic

- ❑ When sending a large volume of packets to the TCP-server, the traffic first passes through SCM-proxy and SDN-controller. Since each component has autoscaling enabled, the traffic will be distributed before reaching the TCP-server. This prevents the TCP-server from being overwhelmed (even if the TCP-server can't handle the load, Kubernetes will automatically generate more pods to handle it)
- ❑ When traffic is low, unnecessary pods will be automatically removed to maintain the initial setup

```
User@Allen-TUF15 D:\NTUST\SDN\final_project\SDN_simulation 131ms
$ kubectl get hpa sdn-controller-hpa --watch
```

NAME	REFERENCE	TARGETS	MINPODS	MAXPODS	REPLICAS	AGE
sdn-controller-hpa	Deployment/sdn-controller	cpu: 1%/20%	1	50	1	27s
sdn-controller-hpa	Deployment/sdn-controller	cpu: 14%/20%	1	50	1	48s
sdn-controller-hpa	Deployment/sdn-controller	cpu: 27%/20%	1	50	1	63s
sdn-controller-hpa	Deployment/sdn-controller	cpu: 27%/20%	1	50	2	78s
sdn-controller-hpa	Deployment/sdn-controller	cpu: 23%/20%	1	50	2	93s
sdn-controller-hpa	Deployment/sdn-controller	cpu: 23%/20%	1	50	3	108s
sdn-controller-hpa	Deployment/sdn-controller	cpu: 22%/20%	1	50	3	2m3s
sdn-controller-hpa	Deployment/sdn-controller	cpu: 21%/20%	1	50	4	2m18s
sdn-controller-hpa	Deployment/sdn-controller	cpu: 20%/20%	1	50	4	2m33s

- ❑ utilize HPA to keep track of usage of CPU and Memory

```
very 1.0s: kubectl.exe get all
```

NAME	READY	STATUS	RESTARTS	AGE
pod/scm-proxy-bd49cc696-2h8zj	1/1	Running	0	17m
pod/scm-proxy-bd49cc696-7s252	1/1	Running	0	17m
pod/scm-proxy-bd49cc696-8btcc	1/1	Running	0	17m
pod/sdn-controller-7dd9c8d68f-z4r8g	1/1	Running	0	23m
pod/tcp-server-6b766995d9-9tvhw	1/1	Running	0	33m
pod/user-client	1/1	Running	1 (103m ago)	16h

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
service/kubernetes	ClusterIP	10.96.0.1	<none>	443/TCP	17h
service/scm-proxy-service	ClusterIP	10.100.184.252	<none>	9090/TCP	48m
service/sdn-controller-service	ClusterIP	10.96.100.54	<none>	9091/TCP	23m
service/tcp-server-service	ClusterIP	10.100.16.157	<none>	9092/TCP	48m

NAME	READY	UP-TO-DATE	AVAILABLE	AGE
Deployment.apps/scm-proxy	3/3	3	3	48m
Deployment.apps/sdn-controller	1/1	1	1	28m
Deployment.apps/tcp-server	1/1	1	1	48m

NAME	DESIRED	CURRENT	READY	AGE
replicaset.apps/scm-proxy-bd49cc696	3	3	3	17m
replicaset.apps/scm-proxy-ccb94f76	0	0	0	48m
replicaset.apps/sdn-controller-7dd9c8d68f	1	1	1	28m
replicaset.apps/tcp-server-6b766995d9	1	1	1	33m
replicaset.apps/tcp-server-c6f99b776	0	0	0	48m

NAME	REFERENCE	TARGETS	MINPODS	MAXPODS	REPLICAS	AGE
horizontalpodautoscaler.autoscaling/scm-proxy-hpa	Deployment/scm-proxy	cpu: 1%/50%	3	50	3	16h
horizontalpodautoscaler.autoscaling/sdn-controller-hpa	Deployment/sdn-controller	cpu: 1%/20%	1	50	1	10m
horizontalpodautoscaler.autoscaling/tcp-server-hpa	Deployment/tcp-server	cpu: 1%/50%	1	50	1	48m

- ❑ pods are autoscaled because of large amount of packets



# Demo Scenario 3: SYN-Flooding Attack

Malicious SYN-flooding attack command: `hping3 scm-proxy-service -p 9090 --syn -i u5000000 -flood`

Shows the user-client performing a SYN-flooding attack against the SCM-proxy

No.	Time	Source	Destination	Protocol	Length	Info
3854...	819.553260	10.244.0.204	10.244.0.189	TCP	64	9090 → 59085 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
3854...	819.553263	10.244.0.189	10.244.0.204	TCP	60	59085 → 9090 [RST] Seq=1 Win=0 Len=0
3854...	819.553317	10.244.0.189	10.244.0.204	TCP	60	[TCP Port numbers reused] 59091 → 9090 [SYN] Seq=0 Win=512 Len=0
3854...	819.553318	10.244.0.204	10.244.0.189	TCP	64	9090 → 59091 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
3854...	819.553321	10.244.0.189	10.244.0.204	TCP	60	59091 → 9090 [RST] Seq=1 Win=0 Len=0
3854...	819.553355	10.244.0.189	10.244.0.204	TCP	60	[TCP Port numbers reused] 59095 → 9090 [SYN] Seq=0 Win=512 Len=0
3854...	819.553356	10.244.0.204	10.244.0.189	TCP	64	9090 → 59095 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
3854...	819.553360	10.244.0.189	10.244.0.204	TCP	60	59095 → 9090 [RST] Seq=1 Win=0 Len=0
3854...	819.553375	10.244.0.189	10.244.0.204	TCP	60	[TCP Port numbers reused] 59097 → 9090 [SYN] Seq=0 Win=512 Len=0
3854...	819.553376	10.244.0.204	10.244.0.189	TCP	64	9090 → 59097 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
3854...	819.553380	10.244.0.189	10.244.0.204	TCP	60	59097 → 9090 [RST] Seq=1 Win=0 Len=0
3854...	819.553414	10.244.0.189	10.244.0.204	TCP	60	[TCP Port numbers reused] 59101 → 9090 [SYN] Seq=0 Win=512 Len=0
3854...	819.553415	10.244.0.204	10.244.0.189	TCP	64	9090 → 59101 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460

replicaset.apps/tcp-server-c6f998776							0	0	0	68m
NAME	REFERENCE	TARGETS	MINPODS	MAXPODS	REPLICAS	AGE				
horizontalpodautoscaler.autoscaling/scm-proxy-hpa	Deployment/scm-proxy	cpu: 17%/50%	3	50	3	17h				
horizontalpodautoscaler.autoscaling/sdn-controller-hpa	Deployment/sdn-controller	cpu: 1%/20%	1	50	1	37m				
horizontalpodautoscaler.autoscaling/tcp-server-hpa	Deployment/tcp-server	cpu: 1%/50%	1	50	1	67m				

No.	Time	Source	Destination	Protocol	Length	Info
72	69.997889	10.244.0.1	10.244.0.200	TCP	72	55004 → 9092 [ACK] Seq=2 Ack=2 Win=64256 Len=0 TSval=1555940334 TSecr=364456
73	79.997566	10.244.0.1	10.244.0.200	TCP	80	59378 → 9092 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM TSval=1555950334 TSecr=364456
74	79.997580	10.244.0.200	10.244.0.1	TCP	80	9092 → 59378 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SACK_PERM TSval=1555960334 TSecr=364456
75	79.997592	10.244.0.1	10.244.0.200	TCP	72	59378 → 9092 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=1555950334 TSecr=364456
76	79.997720	10.244.0.1	10.244.0.200	TCP	72	59378 → 9092 [FIN, ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=1555950334 TSecr=364456
77	79.997751	10.244.0.200	10.244.0.1	TCP	72	9092 → 59378 [FIN, ACK] Seq=1 Ack=2 Win=65280 Len=0 TSval=3644511474 TSecr=364456
78	79.997781	10.244.0.1	10.244.0.200	TCP	72	59378 → 9092 [ACK] Seq=2 Ack=2 Win=64256 Len=0 TSval=1555950334 TSecr=364456
79	89.996478	10.244.0.1	10.244.0.200	TCP	80	51574 → 9092 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM TSval=1555960334 TSecr=364456
80	89.996478	10.244.0.1	10.244.0.200	TCP	80	51580 → 9092 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM TSval=1555960334 TSecr=364456
81	89.996492	10.244.0.200	10.244.0.1	TCP	80	9092 → 51574 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SACK_PERM TSval=1555960334 TSecr=364456
82	89.996492	10.244.0.200	10.244.0.1	TCP	80	9092 → 51580 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SACK_PERM TSval=1555960334 TSecr=364456
83	89.996504	10.244.0.1	10.244.0.200	TCP	72	51574 → 9092 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=1555960334 TSecr=364456
84	89.996504	10.244.0.1	10.244.0.200	TCP	72	51580 → 9092 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=1555960334 TSecr=364456
85	89.996608	10.244.0.1	10.244.0.200	TCP	72	51580 → 9092 [FIN, ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=1555960334 TSecr=364456
86	89.996615	10.244.0.1	10.244.0.200	TCP	72	51574 → 9092 [FIN, ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=1555960334 TSecr=364456
87	89.996624	10.244.0.200	10.244.0.1	TCP	72	9092 → 51580 [FIN, ACK] Seq=1 Ack=2 Win=65280 Len=0 TSval=3644521474 TSecr=364456
88	89.996633	10.244.0.1	10.244.0.200	TCP	72	51580 → 9092 [ACK] Seq=2 Ack=2 Win=64256 Len=0 TSval=1555960334 TSecr=364456

- ❑ Figure above shows that only the SCM-proxy blocked the attack (CPU usage increased significantly only for SCM-proxy)
- ❑ Figure on the left shows that the TCP-server did not receive any malicious attacks (blocked by SCM-proxy)



# Demo Scenario 4: Hybrid Traffic

Malicious Traffic

```
round-trip min/avg/max = 0.0/0.0/0.0 ms
root@user-client:/app# hping3 scm-proxy-service --syn -p 9090 -i u50000
flood
HPING scm-proxy-service (eth0 10.100.104.252): S set, 40 headers + 0 data
tes
hping in flood mode, no replies will be shown
^C
--- scm-proxy-service hping statistic ---
18493519 packets transmitted, 0 packets received, 100% packet loss
round-trip min/avg/max = 0.0/0.0/0.0 ms
```

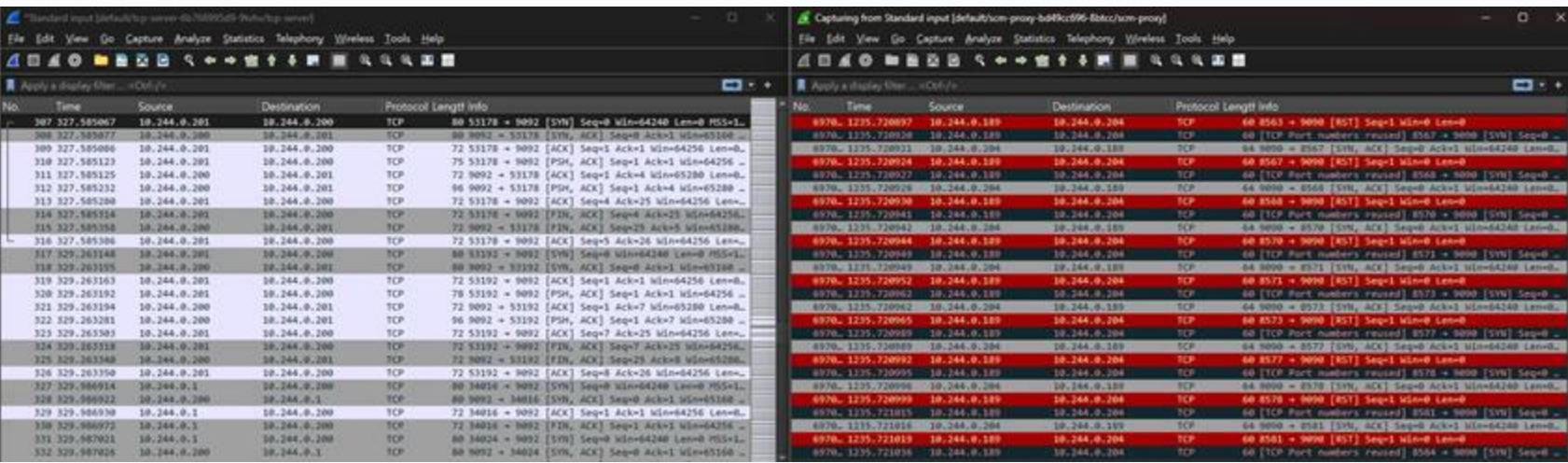
❑ SYN-flooding attack



Normal Traffic

```
$ kubectl exec -it user-client -- /bin/bash
root@user-client:/app# python3 user-client.py
[client] Connected to SCM Proxy.
[client] Sending SYN...
[client] Received: SYN-ACK
[client] Sending ACK...
[client] Enter a message (or 'exit' to quit): hello
[client] Sending payload: hello
[client] Received response: Response from TCP server
[client] Enter a message (or 'exit' to quit): ^[[A
[client] Received response: Response from TCP server
[client] Enter a message (or 'exit' to quit): dfd^[[A
[client] Received response: Response from TCP server
```

Can see that SCM-proxy receives a large volume of incoming packets, while SDN-controller and TCP-server receive fewer packets (because they only handle normal traffic)



NAME	REFERENCE	TARGETS	MINPODS	MAXPODS	REPLICAS	AGE
horizontalpodautoscaler.autoscaling/scm-proxy-hpa	Deployment/scm-proxy	cpu: 35%/50%	3	50	3	17h
horizontalpodautoscaler.autoscaling/sdn-controller-hpa	Deployment/sdn-controller	cpu: 28%/20%	1	50	1	44m
horizontalpodautoscaler.autoscaling/tcp-server-hpa	Deployment/tcp-server	cpu: 13%/50%	1	50	1	74m

❑ Sending normal request to TCP-server

# Advanced Features Overview

## DNS Integration

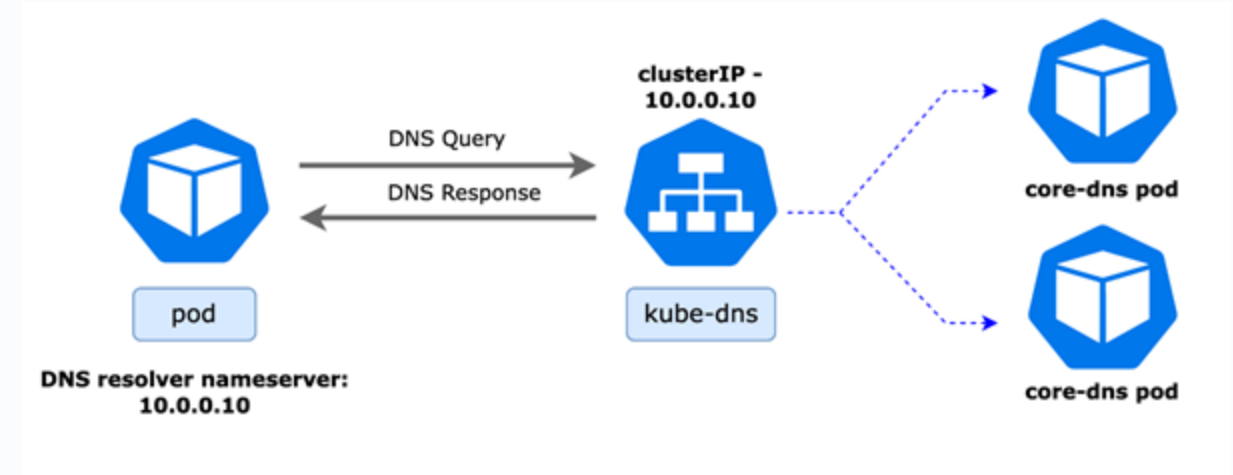
The system integrates with DNS for internal service discovery and dynamic service resolution. This eliminates the need for hardcoded IP addresses, enhancing flexibility and scalability.

## RYU SDN Controller

The RYU SDN controller enables custom routing policies, traffic engineering, and network management. This provides granular control over network traffic flow and optimizes resource utilization.

## Container Registry

The system integrates with Docker Hub for version control and easy deployment of containerized applications. This streamlines the development and deployment process, ensuring consistency and efficiency.



❑ kube-dns service pipeline

# Conclusion

## Challenges



- ❑ Setting up minikube local k8s cluster
- ❑ Handling network communication between services and pods

## Potential Enhancements



- ❑ Security: enhanced filtering algo. such as ML-based threat detection
- ❑ Scaling: vertical pod autoscaling, and cluster autoscaling
- ❑ Monitoring: using Grafana for visualization and alert management for potential issues

Thanks for your attention