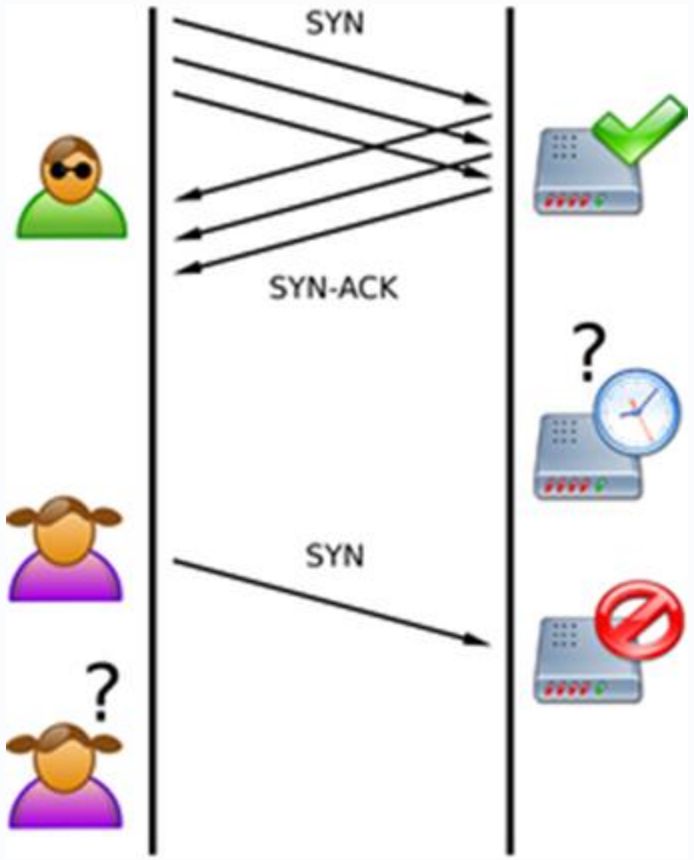
Socket Programming for SYN Flood Mitigation: Simulating Attacks and SCM Proxy Defense

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

Motivation



1

Growing Threat

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

2

Traditional Limitations

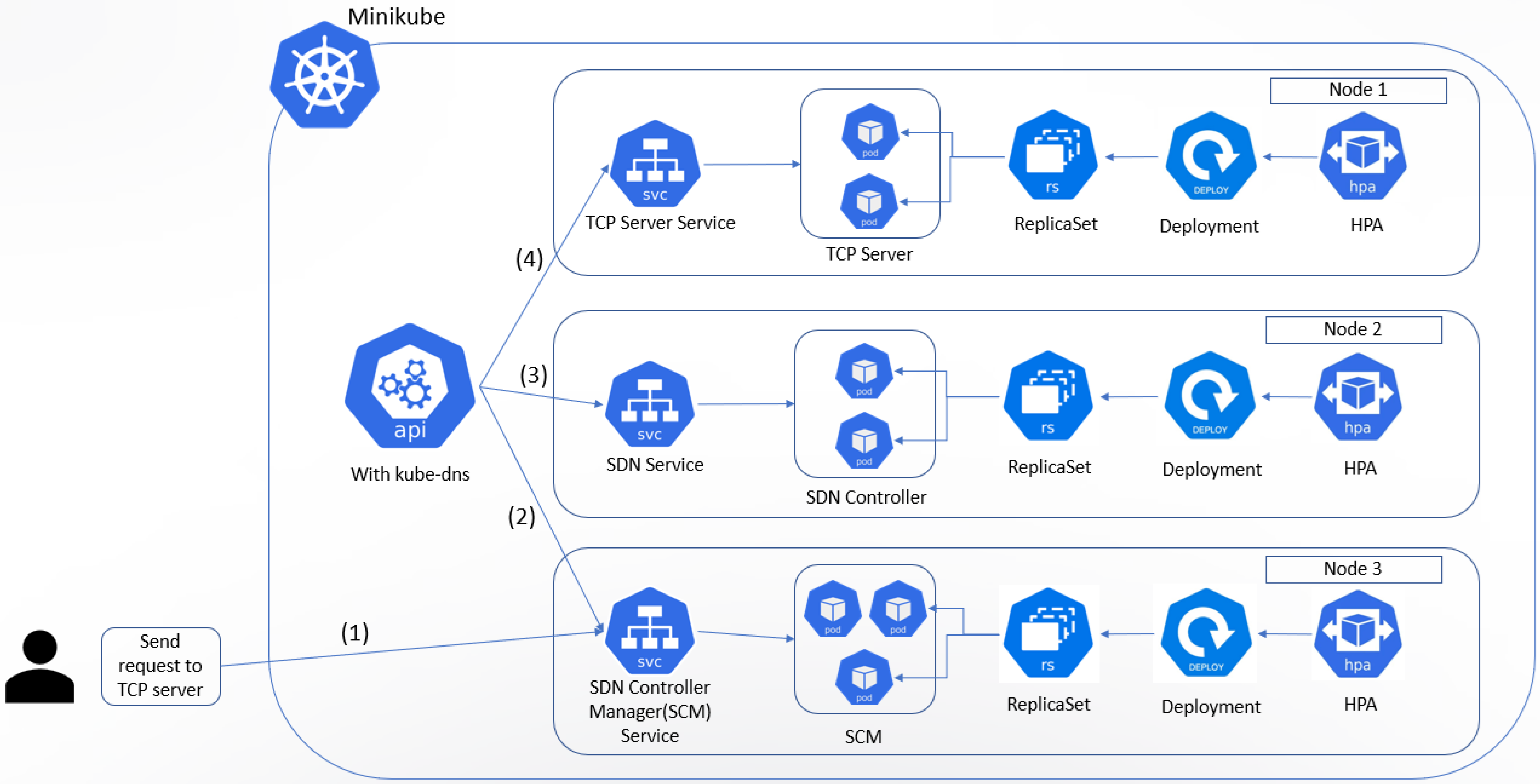
Conventional network architectures struggle to cope with large-scale DDoS attacks.

3

Innovative Solution

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

System Architecture



Environment Setup Requirements

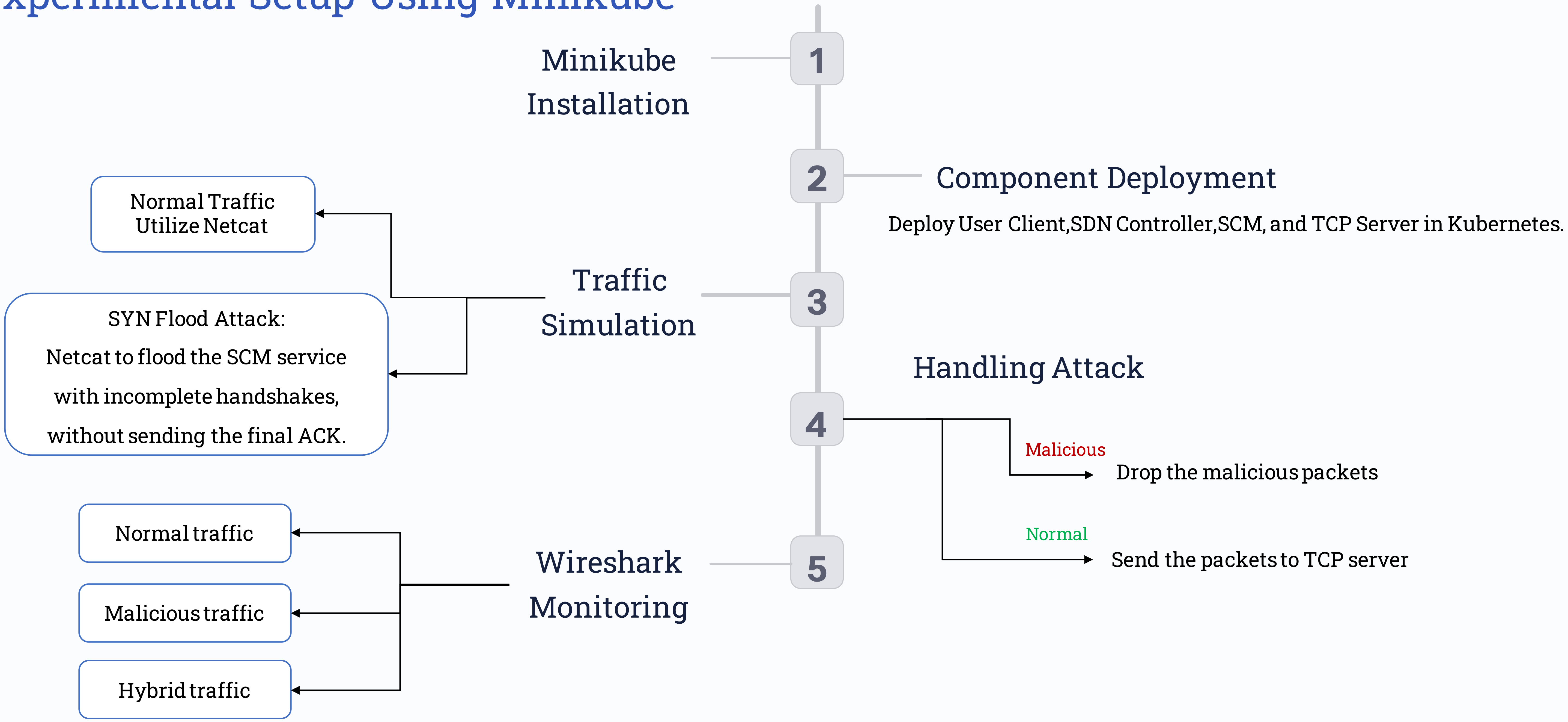
Docker Desktop with Kubernetes Enabled

Minikube
local k8s cluster

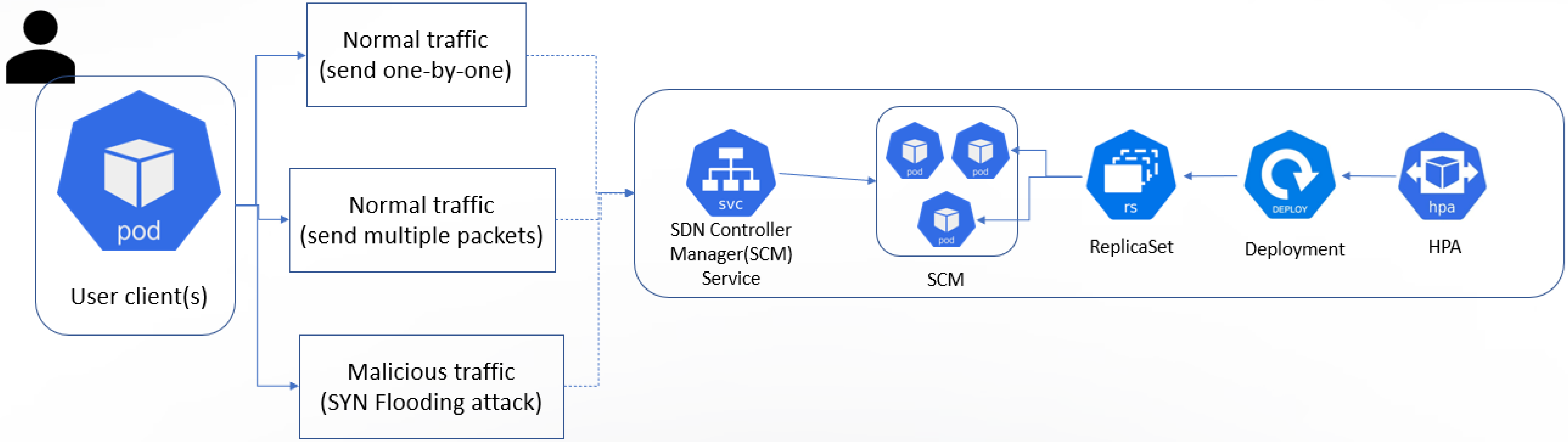
kubectl CLI Tool
interface for k8s

Implementation Methodology

Experimental Setup Using Minikube



Component Implementation: User-Client </>



TCP-Server Implementation

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: tcp-server
spec:
  replicas: 1
  selector:
    matchLabels:
      app: tcp-server
  template:
    metadata:
      labels:
        app: tcp-server
    spec:
      containers:
        - name: tcp-server
          image: allenlin316/tcp-server:v2
          imagePullPolicy: Always
          ports:
            - containerPort: 9092
          resources:
            requests:
              cpu: "100m"
              memory: "128Mi"
            limits:
              cpu: "200m"
              memory: "256Mi"
          livenessProbe:
            tcpSocket:
              port: 9092
            initialDelaySeconds: 15
            periodSeconds: 20
          readinessProbe:
            tcpSocket:
              port: 9092
            initialDelaySeconds: 5
            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

tilizes 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.

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: sdn-controller
spec:
  replicas: 1
  selector:
    matchLabels:
      app: sdn-controller
  template:
    metadata:
      labels:
        app: sdn-controller
    spec:
      containers:
        - name: sdn-controller
          image: allenlin316/sdn-controller:v2
          imagePullPolicy: Always
          ports:
            - containerPort: 9091
          resources:
            requests:
              cpu: "100m"
              memory: "128Mi"
            limits:
              cpu: "200m"
              memory: "256Mi"
          livenessProbe:
            tcpSocket:
              port: 9091
            initialDelaySeconds: 15
            periodSeconds: 20
          readinessProbe:
            tcpSocket:
              port: 9091
            initialDelaySeconds: 5
            periodSeconds: 10
```

Auto-scaling Configuration

```yaml
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: scm-proxy-hpa
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: scm-proxy
  minReplicas: 3
  maxReplicas: 50
  metrics:
  - type: Resource
    resource:
      name: cpu
      target:
        type: Utilization
        averageUtilization: 50
```

minReplicas

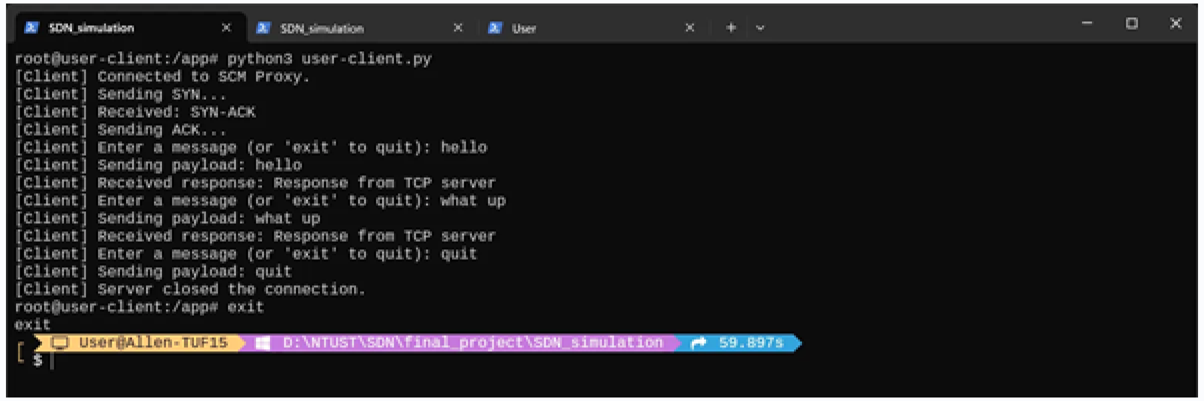
- ❑ ensuring at least 3 replicas are always running

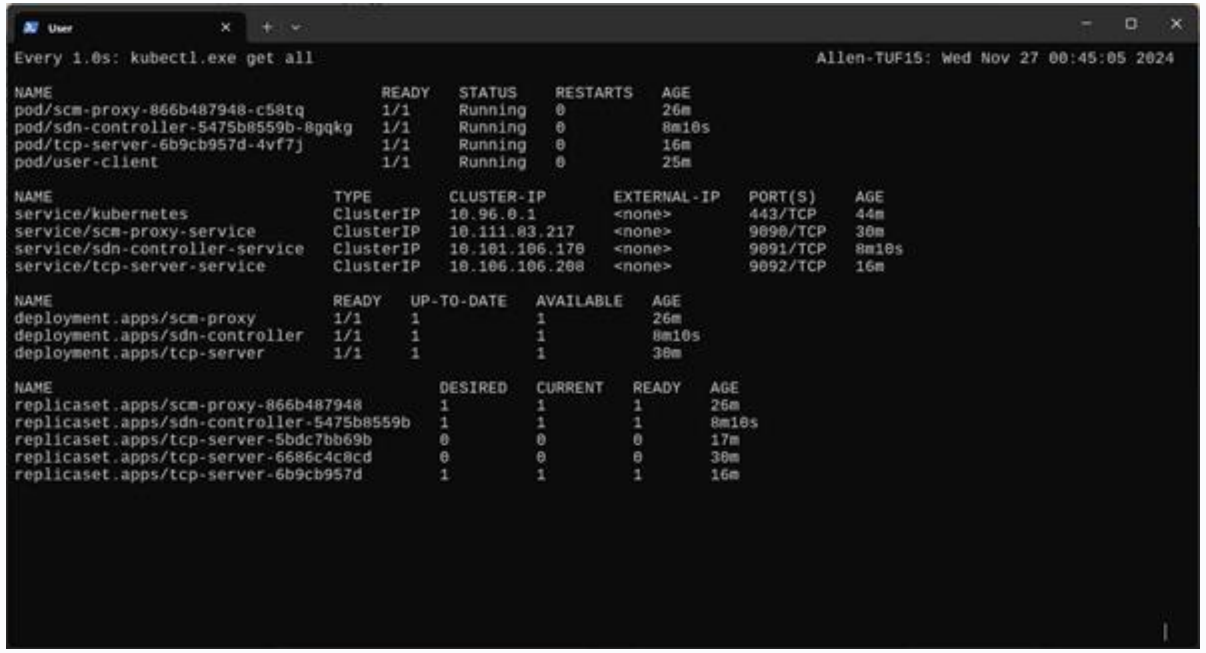
maxReplicas

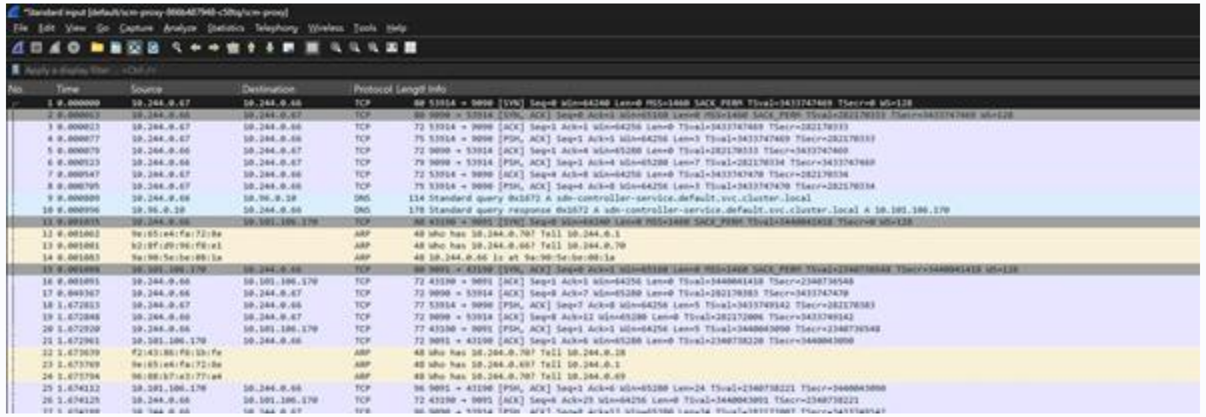
- ❑ preventing excessive scaling beyond this number

averageUtilization

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







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:

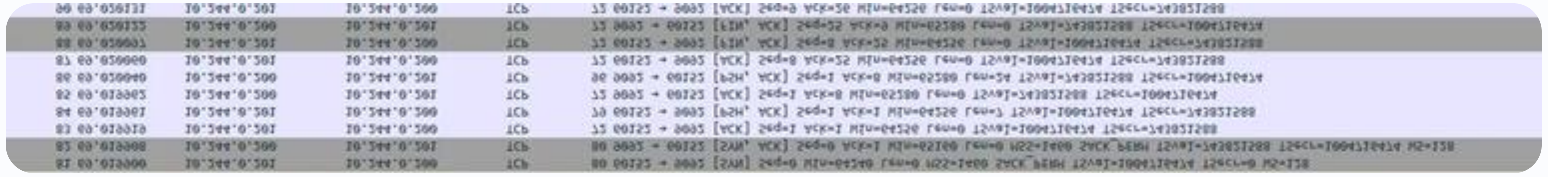
  user-client → SCM-proxy → SDN-controller → TCP-server

- Pipeline for server sending back to user:

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

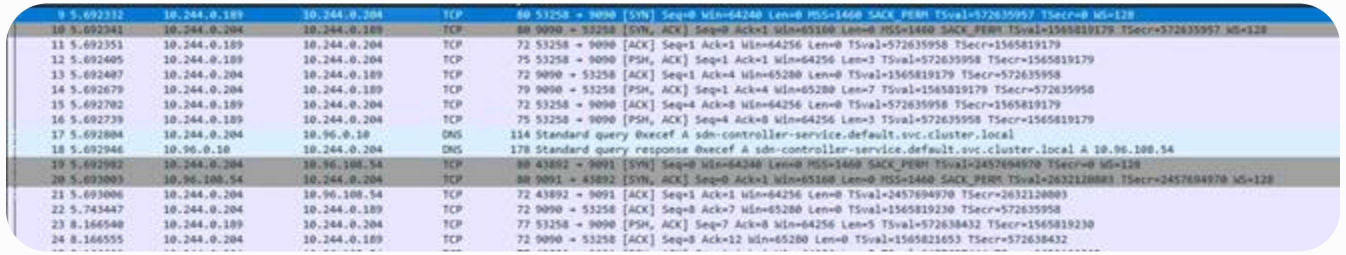
SDN-controller 10.244.0.201 forwarding network packets to TCP-server

```
User@Allen-TUF15  D:\NTUST\SDN\final_project\SDN_simulation  0ms  ! >>ERROR
$ kubectl get pod -o wide
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>
```



```
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): |
```

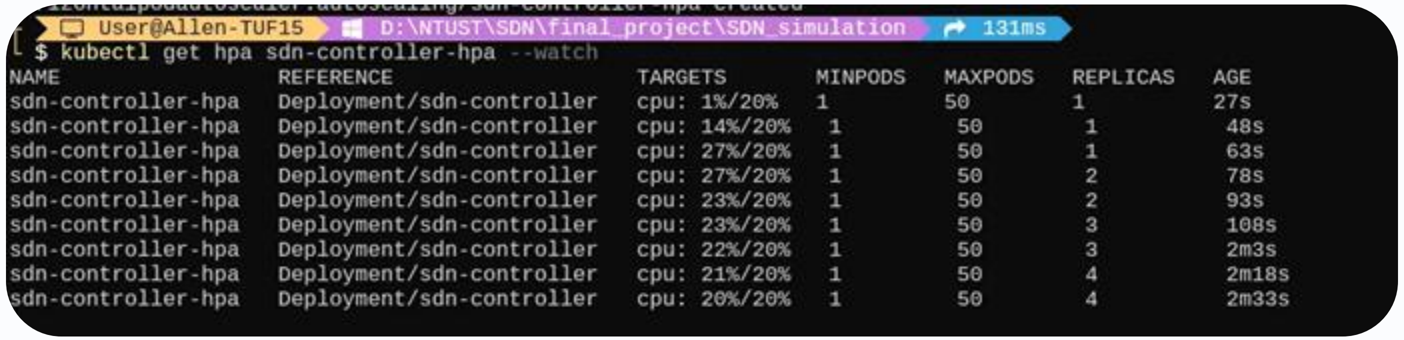
- ❑ Showing packets being sent one by one from the **user-client pod**



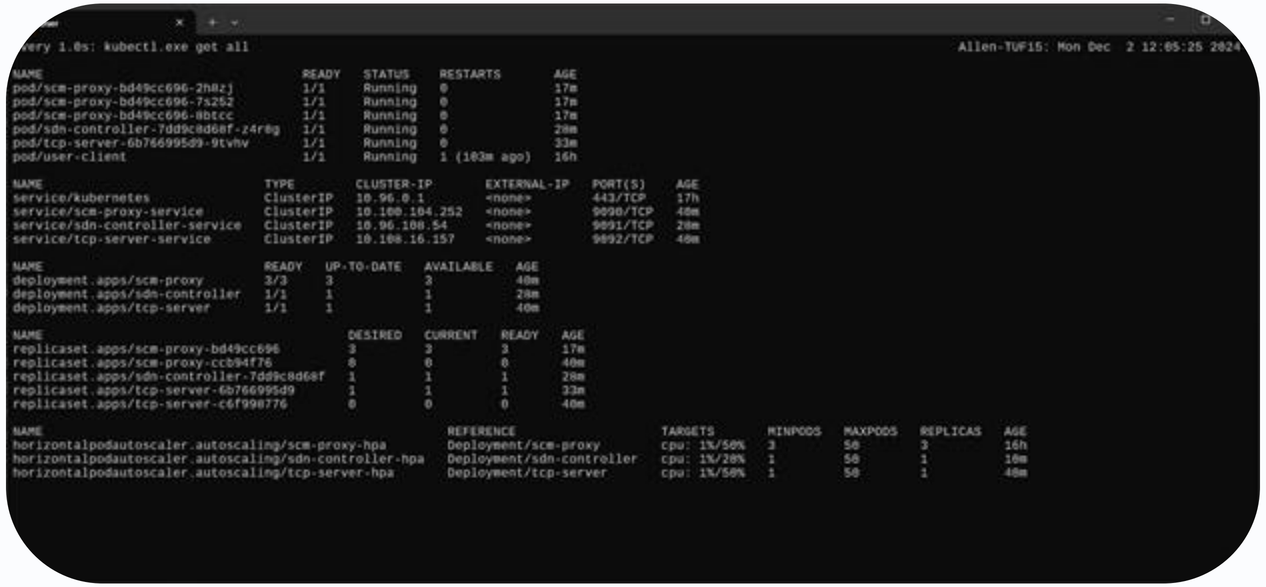
- ❑ Showing user-client sending packets to TCP-server through SCM-proxy and SDN-controller with Wireshark

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



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



- 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=155595033
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
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=364451
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=
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=
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=364451
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=155596033
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=155596033
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
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
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=364452
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=364452
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=
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=
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=
88	89.996653	10.244.0.1	10.244.0.200	TCP	72	51580 → 9092 [ACK] Seq=2 Ack=2 Win=64256 Len=0 TSval=1555960334 TSecr=364452

- ❑ 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 da
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
```

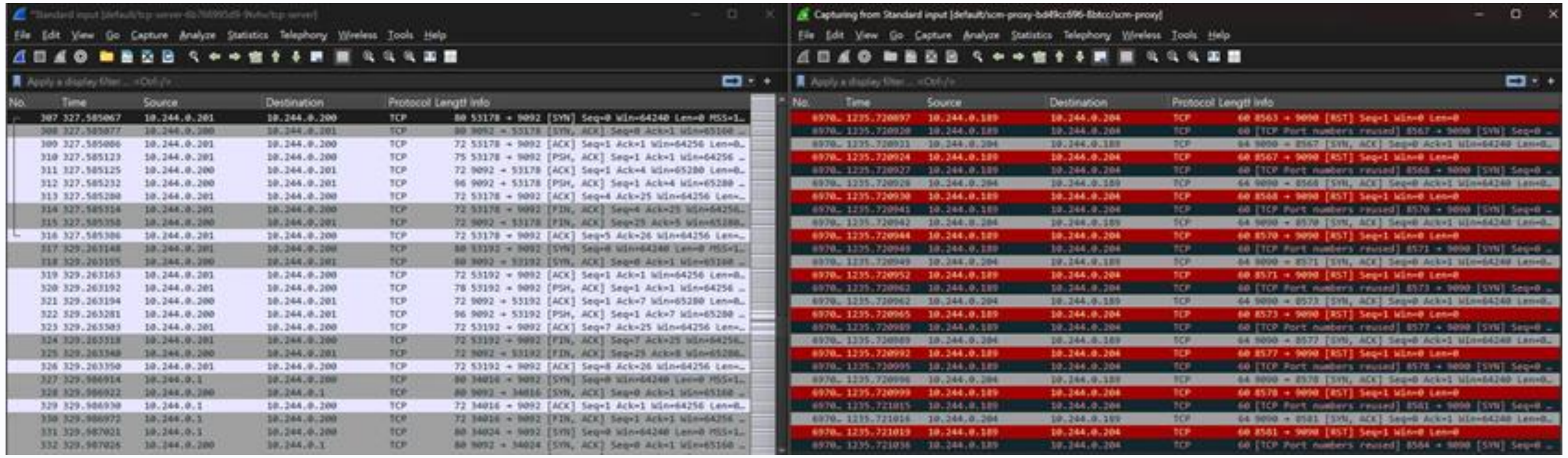
❑ SYN-flooding attack

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
```

❑ Sending normal request to 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
```

Advanced Features Overview

DNS Integration

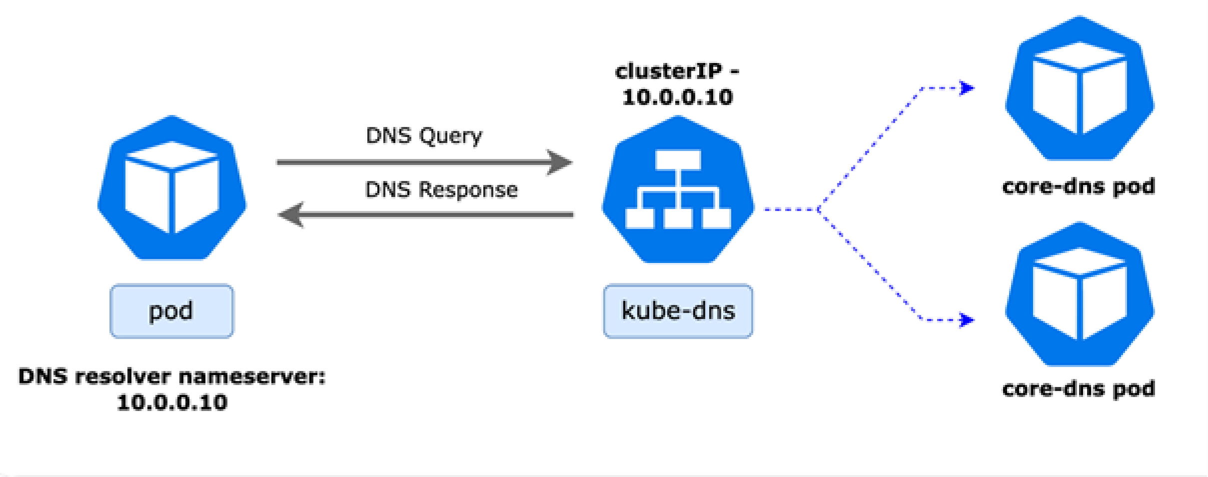
The system integrates with DNS for internal service discovery and dynamic service resolution. This eliminates the need for hardcoded IP addresses, enhancing flexibility and scalability.

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 Granfana for visualization and alert management for potential issues

Thanks for your attention