Fast PDU Session Cleanup for Resource Efficiency

CNDI Final Project Report | Group 9

Team Member

謝孟翰	陳冠霖	詹凱旭	林哲暐
314581047	314551140	314551070	314552006

Agenda

1. Introduction & Motivation
2. SMF Implementation
3. Performance Evaluation
4. Demo
5. Conclusion & Future Work

1. Introduction & Motivation

Why we need active resource management in 5G

The Scenario

In Smart Factory

Thousands of IoT devices connect to the free5GC.

They send sensor data periodically and then go silent.

The Problem: Passive Management

free5GC Current State

By default, free5GC relies on 3GPP standard timers to release PDU sessions.

These timers are often set long enough to **meet user need**

Result

Idle PDU sessions remain in the SMF and UPF, even when no data is being transmitted.

Consequence : Resource Waste

Occupied Resources

- **CPU and Memory Load:**

  Higher processing overhead for maintaining context.

- **System Instability:**

  Reduces the overall reliability of the core network.

Project Objectives

Objectives

Develop Fast Cleanup: Implement an active PDU Session release mechanism in SMF.

Subnet-Based Policy: Create dynamic timeout rules based on UE IP addresses (e.g., IoT vs. Default).

TODO

SMF: Actively monitor traffic(LastActiveTime) and enforce cleanup policies.

free-ran-ue: Supports network-initiated release processes for the fast clean up.

2. SMF Implementation

The free5GC SMF modifications

Implementation of Fast PDU Session Cleanup

Goal: Automate the release of inactive PDU sessions to optimize resource usage.

Core Components:

- **Subnet-Based Policy:** Defines when a session is considered idle.
- **Session Cleaner:** The engine that scans and detects idle sessions.
- **Resource Release:** The mechanism to execute the session termination.

Core Design Principles - Subnet-Based Policy

IP Assignment

When a PDU Session is established, the UE is assigned an IP address.

Policy Lookup

SMF checks the IP prefix against configured policies.

Timeout Set

IoT (10.60.x.x) → 30 secs

Default (10.61.x.x) → 60 secs

struct of CleanupPolicy & SMContext

```go
type CleanupPolicy struct {
    mu sync.RWMutex

    // subnet 到 IdleTimeout 的映射
    // key: 子網前綴 (如 "10.60.0", "10.61.0")
    // value: 閒置超時時間
    subnetPolicies map[string]time.Duration

    // 預設的閒置超時時間（當沒有匹配的子網策略時使用）
    defaultIdleTimeout time.Duration

    // 是否啟用 Fast Cleanup
    enabled bool
}
```

```go
type SMContext struct {
    // LastActiveTime 記錄最後一次資料活動時間（用於閒置檢測）
    LastActiveTime time.Time
    // IdleTimeout 閒置超時時間（根據子網策略設定）
    IdleTimeout time.Duration
    // ================================
```

smfcfg.yaml : set configuration for fast cleanup

```yaml
fastCleanup:
  enabled: true
  defaultIdleTimeout: 3240  # 預設 54 分鐘
  scanInterval: 5          # 每 5 秒掃描一次
  subnetPolicies:
    - subnet: "10.60.0.0/24"
      idleTimeout: 30      # 此子網  30 秒
    - subnet: "10.61.0.0/24"
      idleTimeout: 60      # 此子網 60 秒
```

```go
for _, policy := range config.SubnetPolicies {
    p.AddSubnetPolicy(policy.Subnet, time.Duration(policy.IdleTimeout)*time.Second)
}
```

cleanup_policy.go : AddSubnetPolicy()

```go
// AddSubnetPolicy 新增子網策略
// subnet: 子網前綴，如 "10.60.0" 或 "10.60.0.0/24"
// timeout: 該子網的閒置超時時間
func (p *CleanupPolicy) AddSubnetPolicy(subnet string, timeout time.Duration) {
    p.mu.Lock()
    defer p.mu.Unlock()

    // 解析 CIDR
    prefix := subnet
    if _, ipNet, err := net.ParseCIDR(subnet); err == nil {
        // 提取網路前綴
        prefix = getSubnetPrefix(ipNet.IP)
    }

    p.subnetPolicies[prefix] = timeout
    logger.CtxLog.Infof("Added cleanup policy: subnet=%s, idle_timeout=%v", prefix, timeout)
}
```

pdu_session.go : call SetIdleTimeoutByPolicy()

```
// ===== Fast Cleanup: 設定 IdleTimeout =====
// 在 IP 分配後，根據子網策略設定該 Session 的閒置超時時間
smContext.SetIdleTimeoutByPolicy()


// 在 PDU Session 建立時呼叫
func (smContext *SMContext) SetIdleTimeoutByPolicy() {
    policy := GetCleanupPolicy()
    if !policy.IsEnabled() {
        // Fast Cleanup 未啟用，不設定超時
        smContext.IdleTimeout = 0
        return
    }

    smContext.IdleTimeout = policy.GetIdleTimeoutForIP(smContext.PDUAddress)
    smContext.LastActiveTime = time.Now()
```

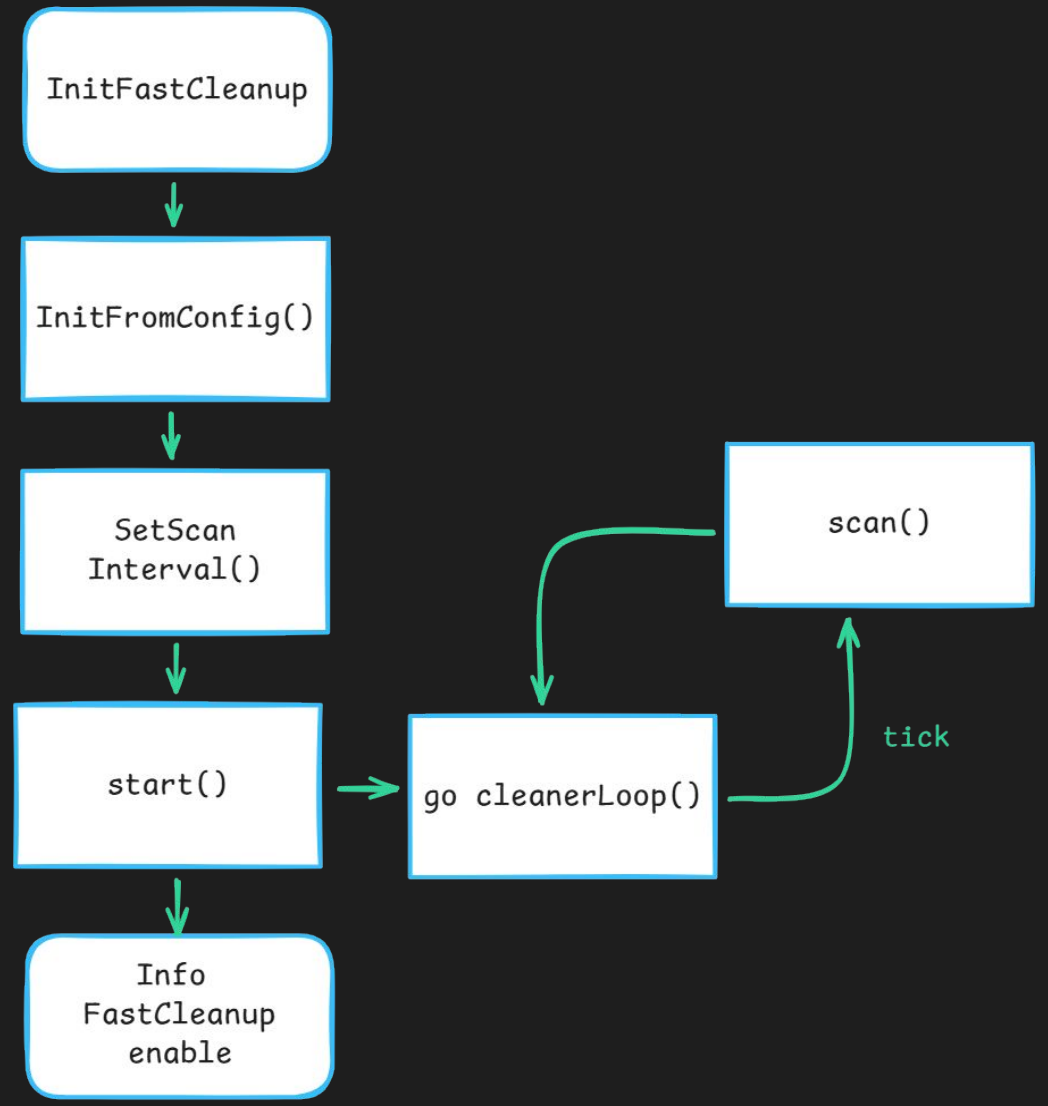
Core Design Principles - Session Cleaner

🧹 Active Cleanup

How to remove devices that have been idle for longer periods of time?

The Solution

A dedicated Goroutine runs periodically.
It iterates through all active SM Contexts.



resource_release.go : InitFastCleanup()

```go
func (p *Processor) InitFastCleanup(config *factory.FastCleanup) {
    if config == nil {
        logger.MainLog.Info("Fast Cleanup config not found, skipping initialization")
        return
    }

    // 初始化策略
    policy := smf_context.GetCleanupPolicy()
    policy.InitFromConfig(config)

    // 初始化清理器
    cleaner := smf_context.GetSessionCleaner()

    // 設定掃描間隔
    if config.ScanInterval > 0 {
        cleaner.SetScanInterval(time.Duration(config.ScanInterval) * time.Second)
    }

    // 設定釋放回調 - 使用網路發起的釋放
    cleaner.SetReleaseCallback(func(smContext *smf_context.SMContext) error {
        return p.NetworkInitiatedReleaseSession(smContext, nasMessage.Cause5GSMInsufficientResourcesForSpecificSliceAndDNN)
    })

    // 啟動清理器
    if config.Enabled {
        cleaner.Start()
        logger.MainLog.Infof("Fast Cleanup enabled - scan_interval=%ds, default_idle_timeout=%ds",
            config.ScanInterval, config.DefaultIdleTimeout)
    }
}
```

session_cleaner.go : Scan()

```go
// 遍歷所有 SM Context，找出閒置的 Session
smContextPool.Range(func(key, value interface{}) bool {
    if value == nil {
        return true
    }
    smContext, ok := value.(*SMContext)
    if !ok || smContext == nil {
        return true
    }
```

```go
    // 檢查是否超過閒置時間
    // 條件：CurrentTime - LastActiveTime > IdleTimeout
    if smContext.IdleTimeout > 0 && !smContext.LastActiveTime.IsZero() {
        idleDuration := now.Sub(smContext.LastActiveTime)
        if idleDuration > smContext.IdleTimeout {
            idleSessions = append(idleSessions, smContext)
        }
    }
    return true
```

```go
    // 釋放閒置的 Session
    cleanedCount := 0
    for _, smContext := range idleSessions {
        if err := callback(smContext); err != nil {
            logger.CtxLog.Warnf("Failed to clean session for UE[%s] PDUSessionID[%d]:
        } else {
            cleanedCount++
        }
    }
```

resource_release.go : NetworkInitiatedReleaseSession()

```go
func (p *Processor) NetworkInitiatedReleaseSession(smContext *smf_context.SMContext, cause uint8)
    smContext.SMLock.Lock()
    defer smContext.SMLock.Unlock()
    logger.PduSessLog.Infof("Network initiated release for UE[%s] PDUSessionID[%d], cause: %d",
        smContext.Supi, smContext.PDUSessionID, cause)
    // 檢查狀態是否允許釋放
    state := smContext.State()
    if state != smf_context.Active && state != smf_context.ModificationPending {…
    }
    // 設定狀態為釋放中
    smContext.SetState(smf_context.InActivePending)
    // 呼叫已有的釋放邏輯
    needNotify, removeContext := p.requestAMFToReleasePDUResources(smContext)
    if needNotify {
        p.SendReleaseNotification(smContext)
    }
    if removeContext {
        p.RemoveSMContextFromAllNF(smContext, false)
    }
```

Core Design Principles - Traffic Detection

Traffic Detection

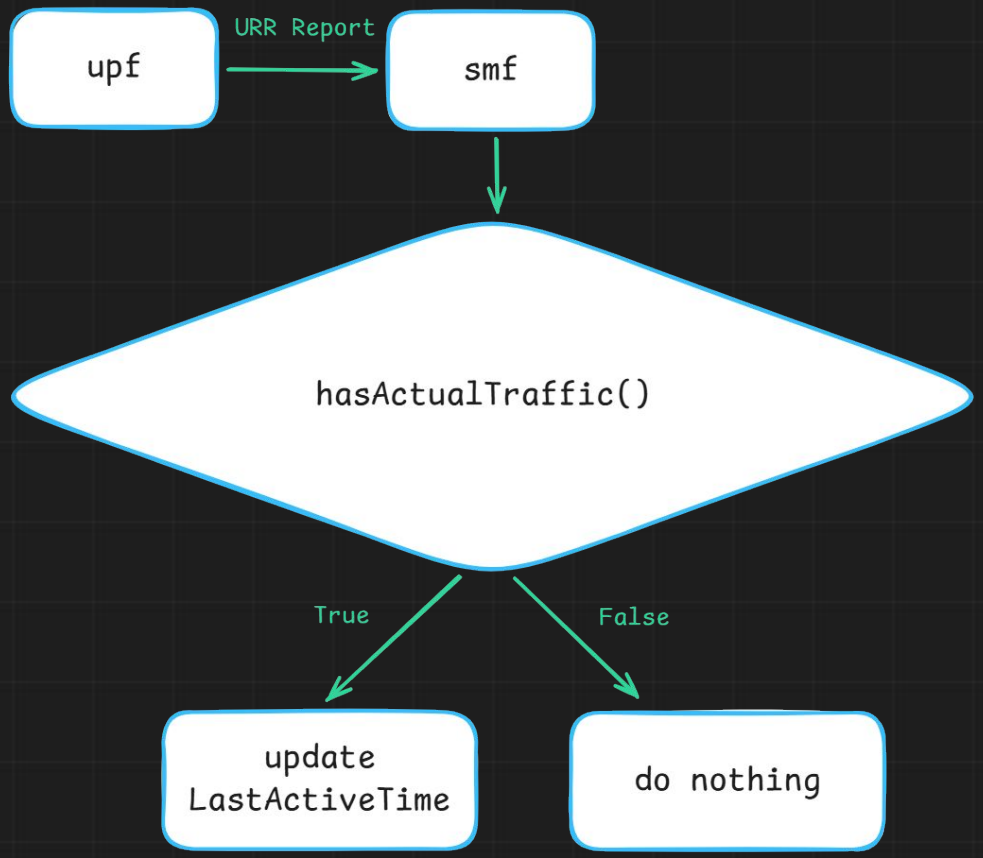
How to determine the traffic of a device?

The Solution

We intercept PFCP Usage Reports to

detect actual packets.

- Check VolumeMeasurement.TotalPktNum > 0
- Check VolumeMeasurement.TotalVolume > 0
- If true, update LastActiveTime = Now()



pfcp_reports.go : hasActualTraffic()

```go
func hasActualTraffic(
    usageReportRequest []*pfcp.UsageReportPFCPSessionReportRequest,
    usageReportModification []*pfcp.UsageReportPFCPSessionModificationResponse,
) bool {
    for _, report := range usageReportRequest {
        if report.VolumeMeasurement != nil {
            if report.VolumeMeasurement.TotalVolume > 0 ||
                report.VolumeMeasurement.TotalPktNum > 0 {
                return true
            }
        }
    }
    for _, report := range usageReportModification {
        if report.VolumeMeasurement != nil {
            if report.VolumeMeasurement.TotalVolume > 0 ||
                report.VolumeMeasurement.TotalPktNum > 0 {
                return true
            }
        }
    }
    return false
```

```go
// Fast Cleanup: 檢查是否有實際資料傳輸，若有則更新最後活動時間
hasTraffic := hasActualTraffic(usageReportRequest, usageReportModification)
logger.PduSessLog.Debugf("[FastCleanup] hasActualTraffic=%v for SM Context %s",
    hasTraffic, smContext.Ref)
```

3. Performance Evaluation

Proving the Solution Works

Test Environment

Core Network

SMF: fastCleanUp mode

UPF: URR report

RAN / UE

Insert Subscriber script for different slice

Closed-Loop stress testing script

SMF Configuration

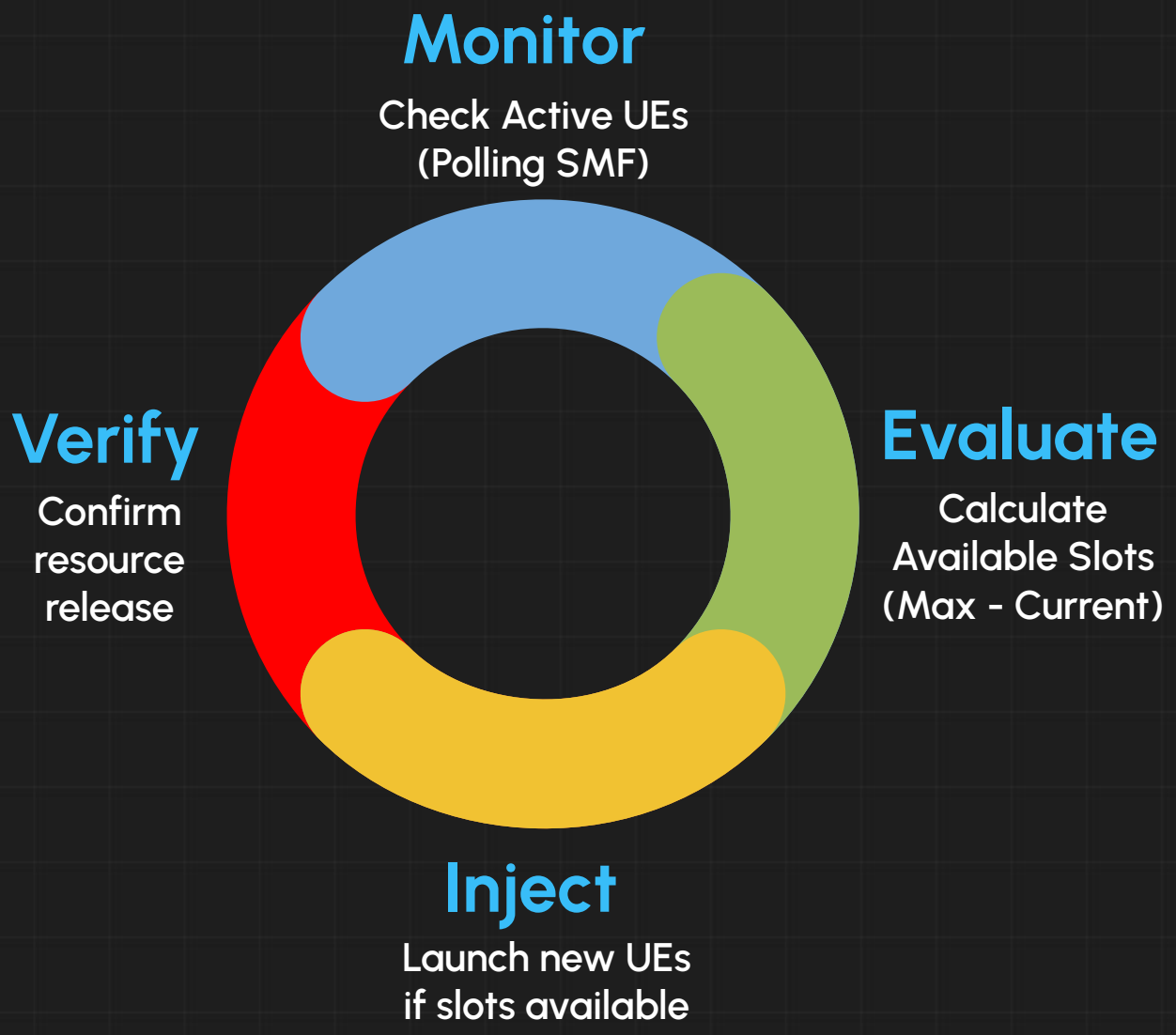
Scan_Interval: 5s

free5GC Default Idle Timeout: 54mins

Subnet Policy: 10.60.0.0/24: 30s, 10.61.0.0/24:60s

```
fastCleanup:
  enabled: true
  defaultIdleTimeout: 3240  # 預設 54 分鐘
  scanInterval: 5           # 每 5 秒掃描一次
  subnetPolicies:
    - subnet: "10.60.0.0/24"
      idleTimeout: 30       # 此子網  30 秒
    - subnet: "10.61.0.0/24"
      idleTimeout: 60       # 此子網 60 秒
```

Closed-Loop Stress Testing

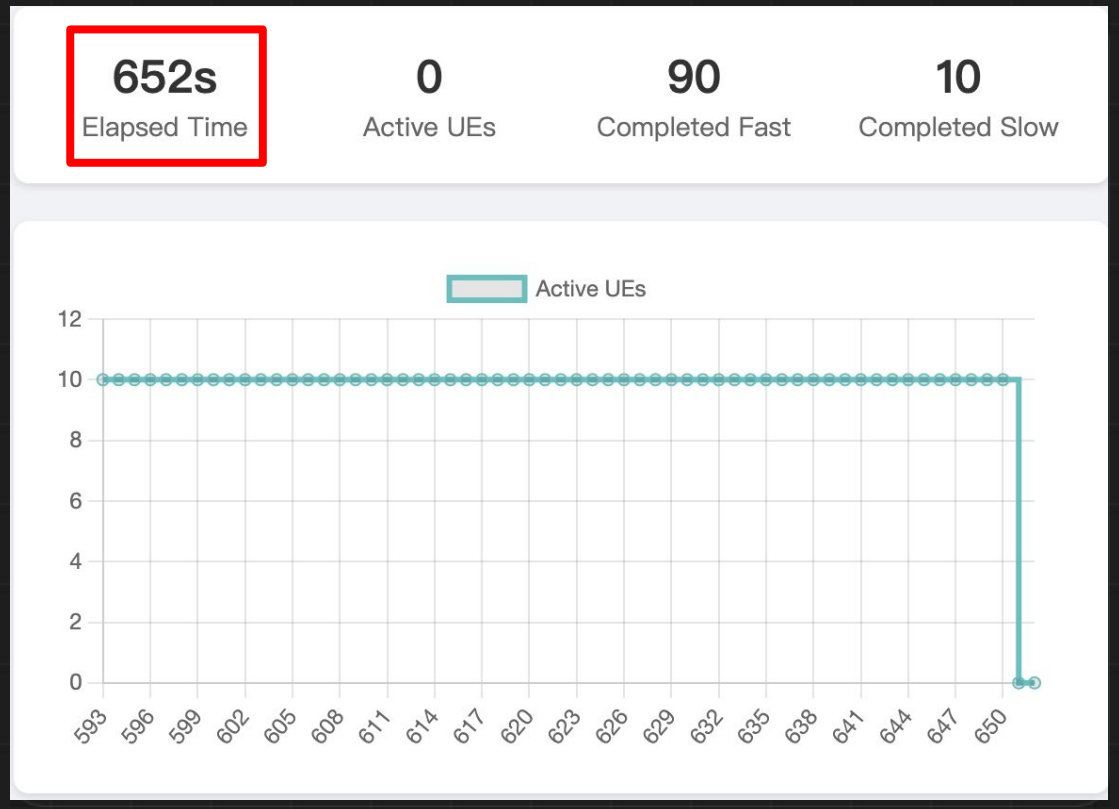


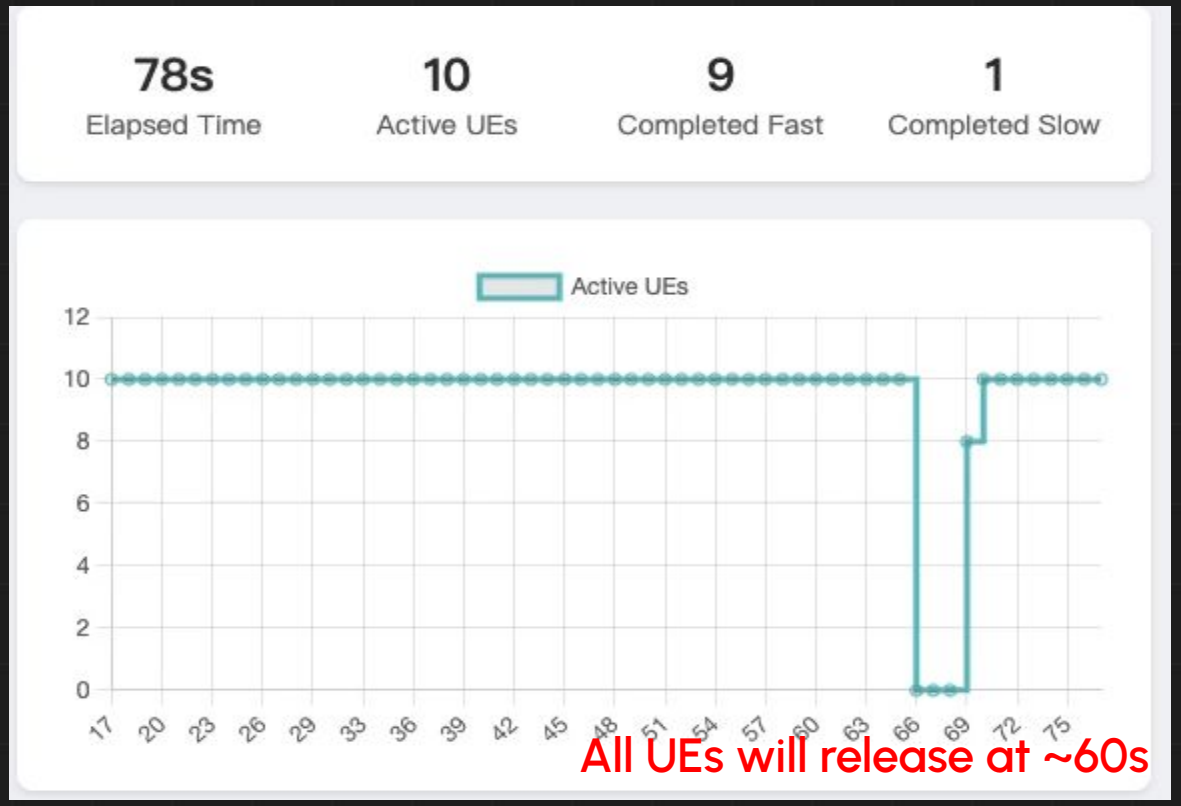
Test Steps

- Step 1: Run free5GC
- Step 2: Run web console
- Step 3: Insert Subscribers.sh 90 10 -> 90 UEs for slice 1(10.60.0.0/24) + 10 UEs for slice 2(10.61.0.0/24)
- Step 4: Run gNB
- Step 5: Start Demo Dashboard: provide real-time charts and listen SMF log
- Step 6: Run Closed-Loop Test Script: build (9 fast UEs + 1 slow UEs) * 10batch to simulate factory

Result: default 60s idleTimeout

Fast_UE and Slow_UE will timeout after 60s idle.

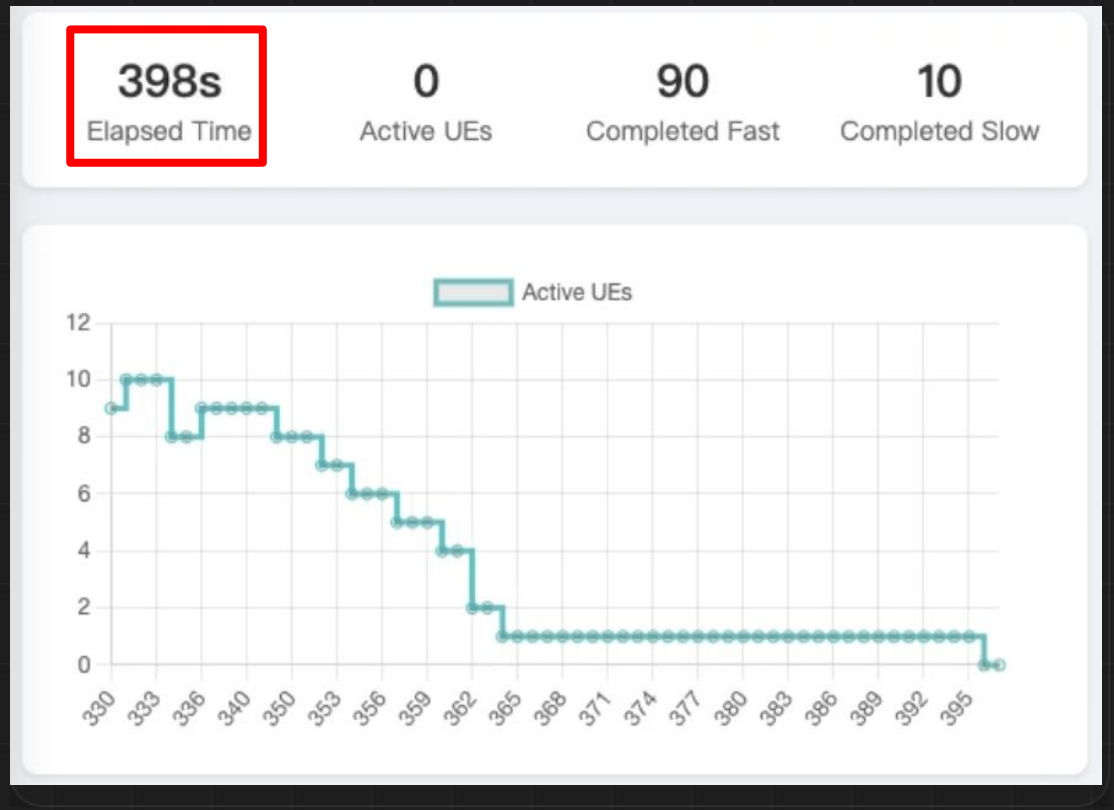


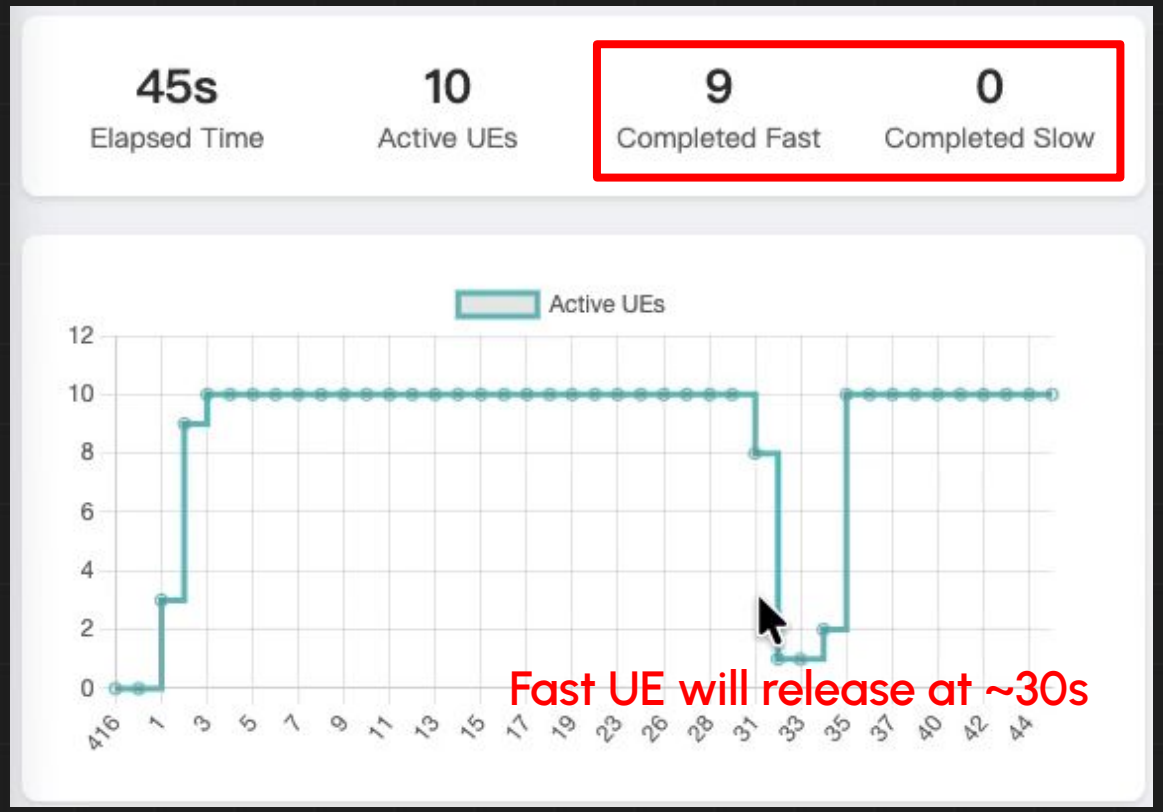


Result: 30s idleTimeout - Fast Cleanup

Fast_UE will timeout after 30s idle

Slow_UE will timeout after 60s idle.





4. Demo

Demo Video

1. Experiment - Default

2. Experiment - Fast CleanUp

5. Conclusion & Future work

Conclusion

Problem: Smart Factory Scenario (IoT Idle State)

Passive Management (Standard Timer)

Resource Waste

Solution: SMF Modification: Implementation Session_Cleaner & Traffic Detection

Subnet-Based Policy: Dynamic timeout based on IP(e.g. IoT v.s Default)

Result: Active resource release

Reduce CPU/Memory overhead

Future Works

Optimization: Polling (Current) -> Event-Driven (Future) Eliminate ScanInterval

Value Proposition: TEID Exhaustion (Public Net) vs. Resource Efficiency (Private Net)

Dynamic Policy: Static Config (YAML) -> AI/Learning-based Adaptive Timeout

Work Assignment List

Name	Job
陳冠霖	SMF logic, free-ran-ue modification
詹凱旭	SMF logic, Last Active Time logic
謝孟翰	Experiment conduction, Result evaluation
林哲暐	Experiment UE setup, Frontend visualization

Thanks!