FUNZIONAMENTO DETTAGLIATO DI SCASDK

1.) Principi di testing

I test SCAS presuppongono delle operazioni ad alto livello eseguite dal tester, raggruppabili in 3 categorie:

- Attori coinvolti: entità, controllate dal tester, che interagiscono con la funzione testata
- Tipi di azioni, che possono essere:
  - funzionali, cioè che scatenano interazioni tra i componenti
  - non-funzionali, che impattano sulla qualità della comunicazione, ad esempio aggiungendo ritardi
- Tipi di verifiche, divisi a loro volta in:
  - occurrence, verifica se un evento è accaduto
  - temporal, verifica se un più eventi si sono susseguiti nell'ordine corretto

2.) Architettura di ScasDK

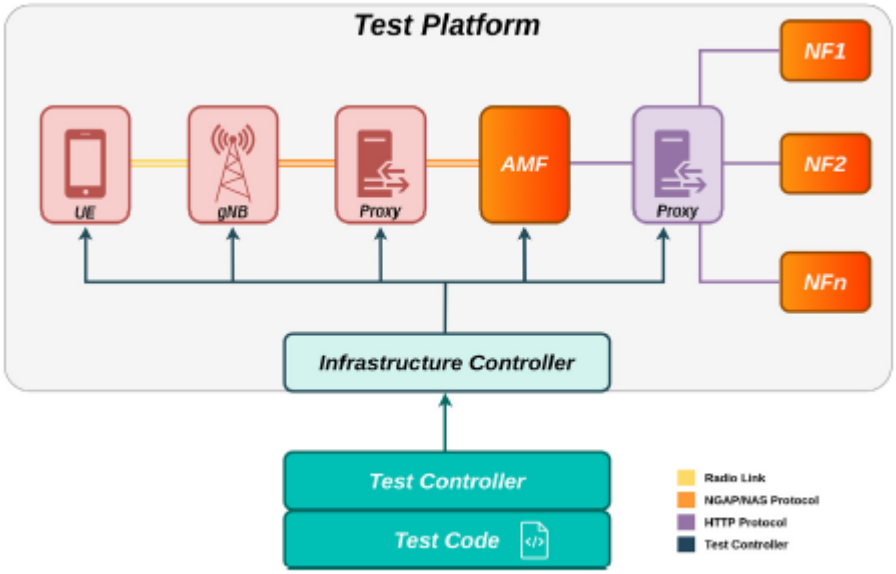


Fig. 5. SCAS Test Platform Architecture.

I componenti fondamentali sono:

- Proxy che modificano il traffico per simulare scenari limite
- Emulatori che generano comportamenti base della rete per simulare condizioni reali
- Managers che supervisionano le entità di test per garantire le corrette operazioni durante il test
- Il Controller che configura l'infrastruttura di test e ne coordina l'esecuzione

Il controller è diviso in due entità distinte:

- Infrastructure Controller, si occupa di gestire tutta l'architettura di test
- Test Controller, si occupa dell'esecuzione della logica di test, organizza le procedure, gli step e valida i risultati

Questa divisione di responsabilità permette di avere ruoli più chiari e una migliore manutenzione. Il Test Controller comunica con l'Infrastructure Controller tramite un'interfaccia dedicata.

2.1) Proxy

Il suo ruolo è quello di trasmettere messaggi tra componenti in modo trasparente e veloce. Inoltre, permette di registrare azioni in via di processazione, denominate *hooks*. Di hook ne esistono due tipi: write e read.
Quelli di tipo write includono azioni capaci di alterare la struttura del messaggio o il flusso della comunicazione, ad esempio aggiungendo ritardi o cancellando il messaggio.
Quelli di tipo read includono azioni che non impattano la struttura del messaggio o il flusso di comunicazione, ad esempio logging e analisi della comunicazione.
Quindi, alla ricezione di un messaggio, il proxy processa in modo sequenziale i write hooks, in contemporanea, dopo ogni modifica, manda a tutti i read hooks il messaggio.

2.2) Emulatori

Questi componenti permettono di generare eventi standard all'interno della rete, ad esempio registrazioni e autenticazioni.
Gli emulatori inizializzano un flusso standard che può poi essere modificato dai proxy.

2.3) Manager

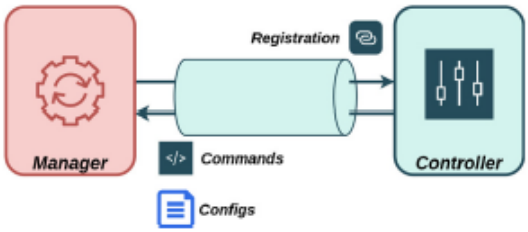
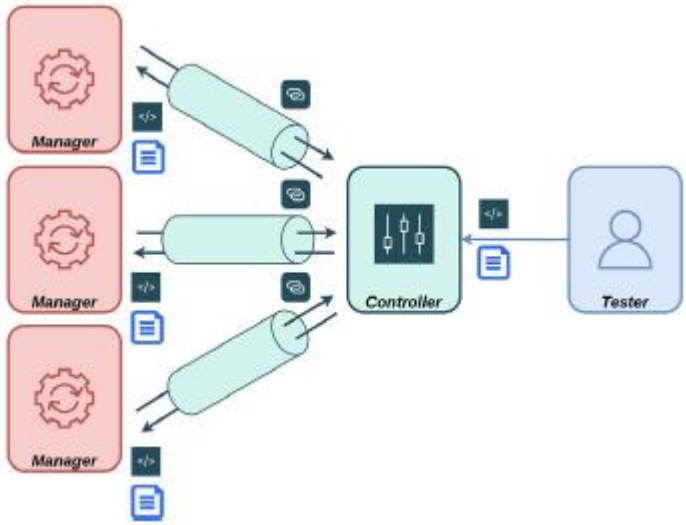


Fig. 6. ScasDK Manager managed by the remote Controller.

Questo componente aumenta le capacità di proxy ed emulatori permettendo il controllo da remoto.
Grazie al manager, il piano di controllo dell'infrastruttura è separato da quello dell'esecuzione del test.
Fornisce funzioni come configurazione, avvio, spegnimento e generazione di eventi, ed espone un'interfaccia unificata che astrae l'implementazione interna, consentendo la sostituzione o l'aggiornamento dei componenti senza impattare gli altri.
È compatibile con tecnologie e linguaggi diversi e mantiene una connessione persistente con un Controller centrale che invia direttamente configurazioni e comandi, eliminando la necessità di localizzare i singoli manager e migliorando l'efficienza di gestione.

2.4) Controller
Il controller ha un ruolo chiave di assicurare l'esecuzione dei test SCAS gestendo: la configurazione delle impostazioni, l'esecuzione del test e altri aspetti correlati. Per ogni test, questo componente configura le risorse necessarie, dopo aver creato uno scenario standard, inizia la fase di test, cattura il traffico tramite gli hooks ed eventualmente lo modifica. Alla fine del test, il controller fornisce il risultato e i dati relativi.

2.4.1) Infrastructure controller



Il suo obiettivo primario è rendere più intuitiva e semplice per il tester la configurazione del test. Tramite un'interfaccia, il tester può inserire i requisiti, senza specificare i metodi per ottenerli, e il controller si occupa della traduzione in operazioni. L'interazione segue un modello publish/subscribe: all'avvio i componenti contattano il controller, si iscrivono agli eventi rilevanti e si identificano con un ID univoco. Quando il controller riceve una richiesta, la invia al componente corretto. A differenza della versione precedente, il tester non deve conoscere gli indirizzi di rete dei componenti, poiché sono questi ultimi a stabilire la connessione. L'uso dell'ID garantisce l'identificazione anche se un componente cambia indirizzo IP, e rende semplice aggiungerne di nuovi, mantenendo un'unica interfaccia di gestione per tutti.

2.4.2) Test controller

Il test controller comprende tutte le logiche di test della piattaforma e il suo compito è di configurare la piattaforma di test assieme all'Infrastructure Controller. In particolare, imposta i parametri della rete e registra gli hooks richiesti dal test. Durante l'esecuzione del test, il controller genera eventi, monitora il traffico e lo modifica se necessario. Alla fine del test valida il risultato e fornisce i dati relativi.
Ogni istanza di test corrisponde ad un test controller separato, che si connette alla piattaforma tramite l'interfaccia dell'Infrastructure controller.

3.) Proof of concept implementation

La piattaforma di test utilizza container orchestrati da Kubernetes per distribuire funzioni 5G e componenti di controllo, seguendo l'approccio cloud-native ma rimanendo compatibile con bare metal e VM. Ingress entry points permettono di testare componenti esterni mantenendo la comunicazione con il sistema di controllo. L'intera piattaforma è confezionata come Helm chart, semplificando installazione, configurazione e gestione su qualunque cluster Kubernetes con coerenza e riproducibilità.
I proxy implementati sono NGAP/NAS e HTTP. Il proxy NGAP/NAS, la cui architettura è modulare, permette di cambiare facilmente il protocollo di trasporto tra SCTP e UDP. Qusto proxy è ottimizzato rispetto alla prima versione di ScasDK, supporta connessioni multiple gNB per scenari di handover. L'HTTP proxy utilizza Envoy, un server proxy spe con capacità di intercettazione e modifica dei messaggi in tempo reale tramite un external processor basato su gRPC, gestito dal controller per soddisfare i requisiti dell'architettura di test.
La piattaforma integra emulatori per funzioni di rete 5G Core, gNB e UE. Le funzioni di rete sono emulate da un client HTTP che supporta NRF e UDM, adatto a test semplici. Per gNB e UE viene utilizzato UERANSIM, uno strumento software creato per simulare le reti 5G New Radio, che implementa lo stack RAN secondo le specifiche 3GPP e usa UDP per sostituire l'hardware RF, offrendo flessibilità, monitoraggio e la possibilità di eseguire azioni di controllo sui componenti.
Nella piattaforma di test sono stati introdotti manager per proxy, gNB e UE (eccetto Envoy che ha già un sistema di gestione proprio). I manager operano in due modalità: standalone, in cui il manager funziona autonomamente e fornisce un'interfaccia gRPC per permettere alle entità esterne di comunicare con lui previa conoscenza del suo indirizzo, e remotely managed, dove, invece di un'interfaccia esterna, il manager stabilisce una connessione persistente con il controller che gli passa poi le configurazioni e le azioni.
L'infrastructure controller è strutturato in più moduli per migliorare la flessibilità e l'adattabilità.
Nell'implementazione del proof of concept sono stati istanziati due controller: uno per supervisionare il proxy HTTP e uno le altre entità. Per gestire il proxy HTTP viene utilizzato come controller Istio, un servizio atto al controllo e alla sicurezza di architetture basate su microservizi, il cui punto forte è la gestione del traffico. Il controller per la gestione degli altri componenti è stato sviluppato appositamente: presenta un'interfaccia dichiarativa per configurazioni (via Kubernetes API) e una imperativa per comandi (via gRPC).
Il test controller si basa sul Robot Framework, un framework per l'automatizzazione dei test, arricchito con librerie e task personalizzati per i test SCAS. Consente di configurare parametri, gestire proxy/UE/gNB, registrare hook, generare eventi, analizzare traffico e valutare i risultati. Grazie a librerie wrapper, i tester possono scrivere script senza conoscere i dettagli implementativi. I test seguono un modello in tre fasi (configurazione, azioni, verifica), applicato in modo iterativo. Il framework offre inoltre report dettagliati, garantendo una visione chiara delle prestazioni e dei problemi del sistema sotto test.
Nella fase di configurazione, vengono configurati i componenti di test, inclusi parametri di rete e hook. Una libreria dedicata semplifica la gestione degli hook, permettendo ai tester di scrivere solo la logica. Gli hook, implementati come classi Python, elaborano messaggi, definiscono condizioni e attivano handler su eventi specifici. La libreria gestisce automaticamente trigger ed eventi, registrandoli con timestamp per consentire una verifica accurata nei test.
Nella fase di azione vengono innescate alcune azioni per testare la rete, e il tester interagisce con gli emulatori per creare condizioni desiderate.
L'ultima fase è quella di controllo che gli eventi si siano effettivamente verificati e se lo abbiano fatto nell'ordine corretto, oltre a verificare lo stato dei componenti.

4.) Risultati

I risultati mostrano che la piattaforma consente di sviluppare ed eseguire test di sicurezza su implementazioni 5G Core open-source, con facilità anche per sviluppatori esterni inesperti. La sua adattabilità è stata dimostrata con l'integrazione in un caso DevSecOps e con l'applicazione in scenari di test online. L'analisi delle prestazioni evidenzia che l'impatto dei proxy è trascurabile o persino positivo, confermando la validità dell'approccio per test in tempo reale senza compromettere le operazioni di rete.
I test eseguiti sono stati 17:

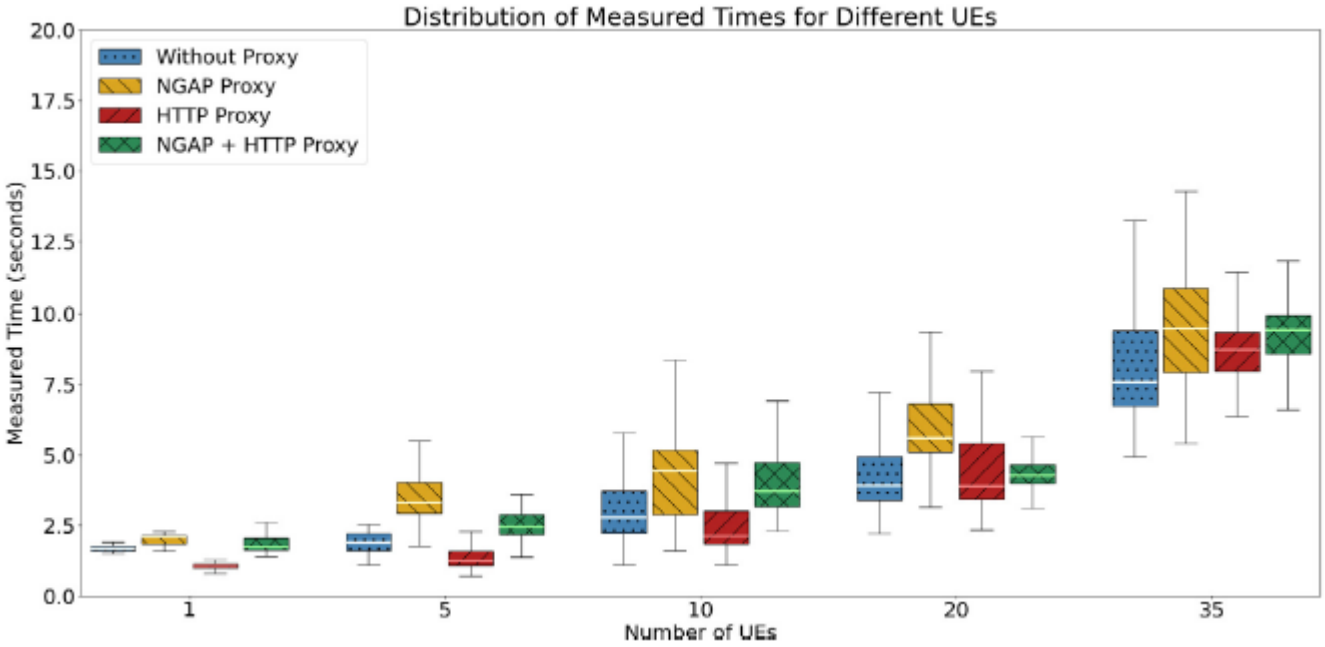
- Network Repository Function (NRF):
- NF Discovery Authorization for specific slice
- User Data Management Function (UDM):
- Synchronization Failure Handling
- De-concealment of SUPI from the SUCI based on the protec-tion scheme used to generate the SUCI.
- Storing of authentication status of UE by UDM
- Access and Mobility Management Function (AMF):
- RES* verification failure handling – Case A
- RES* verification failure handling – Case B
- RES* verification failure handling – Case C
- RES* verification failure handling – Case D
- Synchronization failure handling - Case A
- Synchronization failure handling - Case B
- NAS NULL integrity protection - Case A
- NAS NULL integrity protection - Case B
- NAS integrity algorithm selection and use
- Invalid or unacceptable UE security capabilities han-dling - Case 1
- Invalid or unacceptable UE security capabilities han-dling - Case 2
- Invalid or unacceptable UE security capabilities han-dling - Case 3
- Invalid or unacceptable UE security capabilities han-dling - Case 4

Per convalidare l'approccio, la piattaforma di testing è stata utilizzata impiegata su 3 Core-Networks: Open5GS v2.7.0, free5GC v3.3.0, and OAI v2.1.0.

Table 2
Test results.

	Open5GS v2.7.0	free5GC v3.3.0	OAI v2.1.0
NRF tests			
TC_DISC_AUTHORIZATION_SLICE_NRF	✗	✗	✗
UDM tests			
TC_SYNC_FAILURE_HANDLING_UDM	✓	✓	✗
TC_DE-CONCEAL_SUPI_from_SUCI_UDM	✓	✓	N/A
TC_AUTH_STATUS_STORE_UDM	✓	✓	✓
AMF tests			
TC_RES_STAR_VERIFICATION_FAILURE – Case A	✗	✗	✗
TC_RES_STAR_VERIFICATION_FAILURE – Case B	✗	✗	✗
TC_RES_STAR_VERIFICATION_FAILURE – Case C	✗	✓	✗
TC_RES_STAR_VERIFICATION_FAILURE – Case D	✗	✓	✗
TC_SYNC_FAIL_SEAF_AMF – Case A	✓	✓	✓
TC_SYNC_FAIL_SEAF_AMF – Case B	✓	✓	✗
TC_NAS_NULL_INT_AMF - Case A	N/A	N/A	N/A
TC_NAS_NULL_INT_AMF - Case B	✓	✓	✓
TC_NAS_INT_SELECTION_USE_AMF	✓	✓	✓
TC_UE_SEC_CAP_HANDLING_AMF - Case 1	✗	✓	✗
TC_UE_SEC_CAP_HANDLING_AMF - Case 2	✓	✓	✗
TC_UE_SEC_CAP_HANDLING_AMF - Case 3	✗	✓	✗
TC_UE_SEC_CAP_HANDLING_AMF - Case 4	✓	✓	✗

I test, come visibile nella tabella, mostrano che nessuna delle tre implementazioni 5G Core soddisfa gli standard di sicurezza: OAI supera solo 4 test su 17, Open5GS 9, free5GC 13. L'analisi dei tempi di esecuzione (da pochi secondi a 1,5 minuti) rivela che i test NRF/UDM sono rapidi grazie a semplici scambi HTTP, mentre quelli AMF risultano più lunghi e complessi, anche per via di timeout o ritardi introdotti intenzionalmente.



Nella fase finale del processo di valutazione sono state prese in considerazione le prestazioni di connessione degli UEs al Core Network con e senza l'introduzione di proxy, per verificare che non introducano rallentamenti significativi che comprometterebbero i test online. L'infrastruttura è stata ospitata su un server Dell PowerEdge con quattro VM: una dedicata al controller e tre al cluster Kubernetes (un master e due worker). Per superare i limiti di rete di Kubernetes, è stato usato il plug-in Multus, così da fornire più interfacce ai pod, requisito necessario per componenti come l'UPF. Tutti i componenti della rete 5G (free5GC) e della piattaforma ScasDK sono stati distribuiti in pod, ciascuno con un sidecar proxy Istio per la gestione trasparente della comunicazione.

Per i test, gli emulatori UE sono stati estesi a supportare la valutazione delle prestazioni. Tramite l'infrastructure controller, gli UEs hanno eseguito cicli completi di registrazione, attivazione di sessione PDU e deregistrazione, registrando i tempi e pubblicando le metriche su Prometheus. Sono stati considerati quattro scenari: (1) baseline senza proxy, (2) solo NGAP/NAS proxy, (3) solo HTTP proxy, (4) entrambi i proxy. Ogni scenario è stato ripetuto per almeno 50 cicli e con diverse quantità di UEs, sfruttando lo scaling orizzontale di Kubernetes.

I risultati mostrano che Istio offre prestazioni migliori del previsto, grazie al service mesh che stabilisce connessioni persistenti tra servizi, eliminando l'overhead dei continui handshake TCP. Questo riduce sensibilmente la latenza di procedure complesse come l'Authentication and Key Agreement (AKA), che comportano numerose richieste HTTP. Il proxy NGAP, invece, risulta meno performante: ciò era atteso, trattandosi di un prototipo privo di ottimizzazioni, ma le prestazioni restano comunque accettabili. Infine, lo scenario con entrambi i proxy attivi non presenta degradi rilevanti rispetto al baseline, confermando la fattibilità di scenari di test con proxy. L'analisi su differenti numeri di UEs evidenzia andamenti simili: con pochi UEs si osserva maggiore irregolarità, mentre con venti UEs i tempi medi di connessione diventano stabili dopo una breve fase iniziale.