MSCS-631: Design and Simulation of a Scalable and Secure Network Architecture for IoT Applications – Phase 2

Oishani Ganguly

GitHub Repository: https://github.com/Oishani/MSCS631_Project

Security Framework for a Smart-City IoT Architecture
- Executive Summary
- Configuration
  - Topology & Components
  - Security Mechanisms Implemented
- Test Methodology
  - Scalability Tests
  - Security Tests
- Theoretical Results
  - Scalability Summary (Theoretical)
  - Per-Sensor Breakdown Example
  - Security Test Outcomes (Theoretical)
- Analysis
  - Scalability
  - Security
  - Limitations
- Extension Toward a Smart-City IoT Security Framework
  - Identified Gaps and Required Components
  - Adaptation & Extension Recommendations
- Proposed Improvements
- Conclusion

Security Framework for a Smart-City IoT Architecture

Executive Summary

This report details the configuration, tests performed, theoretical results, analysis, and suggestions for future enhancements to the architecture for a secure smart-city IoT architecture. The stack includes simulated sensors, a gateway enforcing policy and anomaly detection, a cloud policy/ingest service, and optional firewall segmentation. This is a proof of concept only and therefore live data capture was omitted. Thus, the results presented are synthetic/theoretical, generated to reflect expected behaviors from the actual simulation scripts. Accompanying artifacts are provided to justify the analysis.

Configuration

Topology & Components

- **Sensors:** Multiple simulated IoT sensors (`sensor-1...sensor-N`) sending telemetry (temperature readings) to the gateway.
- **Gateway:** Receives encrypted telemetry, fetches policy from cloud, enforces allowed sensor list and performs rate-based anomaly detection, forwards valid telemetry.
- **Cloud Service:** Hosts `/policy` endpoint and secure ingestion endpoint; rejects unauthorized sensors.
- **Firewall:** Linux `iptables` restricting traffic to TLS port 8443 between gateway and cloud.
- **PKI:** Self-signed CA issues certificates; TLS used for sensor→gateway and gateway→cloud.

Security Mechanisms Implemented

- TLS (server-side; optional plaintext mode for isolated unit testing).
- Policy enforcement via cloud-provided allowlists.
- Rate anomaly detection using sliding-window thresholds.
- Input validation for malformed JSON and missing fields.
- Unauthorized sensor rejection.
- Firewall-enforced segmentation.

Test Methodology

Scalability Tests

Simulated concurrent sensors, each producing telemetry to gauge latency growth and success rate under load. Parameters used for theoretical data: 1, 5, 10, and 20 concurrent sensors, each sending 30 messages at 0.2s intervals. Gateway thresholding (20 messages/60s) produces throttling under high concurrency.

Security Tests

Three attack categories were simulated theoretically via the attack simulator logic:

1. **Unauthorized Access** - sensor ID not in policy.
2. **Malformed Payloads** - non-JSON and missing `sensor_id`.
3. **Rate Flood** - legitimate sensor exceeding normal message rate to trigger throttling.

Theoretical Results

Scalability Summary (Theoretical)

The following table summarizes expected behavior as the number of concurrent sensors increases under the workload. Each sensor sends 30 telemetry messages at 0.2 second intervals. The gateway enforces a threshold of roughly 20 messages per 60 seconds to prevent runaway load; beyond certain concurrency this results in observable throttling and higher latency due to queuing, TLS overhead, and internal processing delays.

# Sensors	Avg Latency (s)	Success Rate (%)	Throttling Behavior	Typical Variability (±)
1	0.11	~100.0	None	±0.02
5	0.13	~99.3	Rare	±0.03

10	0.18	~97.0	Occasional	±0.05
20	0.30	~92.5	Frequent	±0.08

Interpretation

Latency increases with concurrency primarily because of TLS handshake costs for new connections, context switching while servicing many simultaneous sockets, and queuing when bursts arrive faster than the gateway can process and forward telemetry. The success rate degrades as throttling becomes active; this trade-off protects the cloud backend from sustained overload. Variability captures jitter introduced by scheduling, simulated throttling, and transient policy enforcement delays, with heavier load producing wider spreads in observed latencies.

Additional theoretical confidence

Under stable conditions the system would maintain over 95% success for up to 10 sensors; beyond that, adaptive backoff or connection pooling would help avoid oscillation and further latency spikes.

Per-Sensor Breakdown Example

sensor_id	total	success	failures	success_rate_percent	avg_latency_secs	median_latency_secs	p90_latency_secs	p99_latency_secs
sensor-1	30	30	0	100.0	0.1099	0.1072	0.1345	0.1478

Security Test Outcomes (Theoretical)

```
[*] Unauthorized sensor test (should be rejected by policy)
```

```
Sent payload: {'sensor_id': 'sensor-bad', 'temperature': 30,
'timestamp': '2025-08-03T11:05:00Z'}


Gateway response: {"status":"rejected","reason":"not allowed"}


Result: PASS (unauthorized sensor was rejected)
```

Malformed payloads

```
[*] Malformed payload test

-- Non-JSON test --

Sent: this-is-not-json

Response: {"status":"error","reason":"bad format"}

-- Missing sensor_id test --

Sent: {'temperature': 25}

Response: {"status":"error","reason":"missing sensor_id"}

Result: PASS (malformed inputs handled/rejected)
```

Rate flood scenario

```
[*] Rate flood test (attempt to trigger throttling/anomaly
detection)


Flood duration: 15s at 25 msg/sec


Initial responses: {"status":"accepted"} x N
```

```
Threshold breach logged: WARNING Rate anomaly detected;
throttling message
```

```
Subsequent responses: {"status":"throttled"} interleaved with
occasional accepted ones
```

```
Result: PASS (rate anomaly detection engaged)
```

Edge case notes

- Replayed unauthorized requests are still rejected since sensor identity is validated per message; without proper replay protection, repeated attempts would be visible in logs but not accepted.
- If malformed payloads are carefully crafted to resemble valid structure (e.g., JSON with unexpected fields), the system currently lacks schema validation beyond presence of `sensor_id` and may accept harmless extraneous data, which should be considered in future extensions.
- During rate floods, the gateway's sliding window mechanism may temporarily allow short bursts before throttling kicks in, so detection latency is small but nonzero; this behavior is visible in the interleaving of accepted and throttled responses.

Analysis

Scalability

Latency grows with concurrent load; the gateway's throttling mechanism prevents overload by dropping or delaying excess messages, trading off completeness (success rate) for system stability. At moderate scale (≤10 sensors) latency remains acceptable (<200ms) with high success. Beyond that, throttling becomes visible, reducing throughput but protecting downstream cloud services.

Security

Policy enforcement effectively blocks unauthorized identities; every telemetry message is checked against a dynamically fetched allowlist, minimizing the risk of rogue devices injecting data. Malformed payloads are caught early - non-JSON or missing critical fields cause rejection, preventing corruption of downstream logic. High-rate bursts trigger rate anomaly detection, with synthetic logs indicating throttling engagement, which curtails volumetric abuse. However, there are nuanced security trade-offs: stale policy data (if the gateway fails to refresh due to temporary cloud unavailability) could temporarily allow previously revoked sensors unless additional revocation signaling is added. The current mechanism also risks false positives during legitimate short-lived bursty traffic, and without cryptographic replay protection, attackers could replay previously captured valid telemetry (though TLS mitigates this to some degree if session parameters are not reused). Comprehensive logging is essential for post-incident analysis; the system logs key events such as policy updates, rejections, and throttling, which can be correlated (with consistent timestamps) to detect blended attack patterns.

Limitations

- Static thresholds for anomaly detection may misclassify benign bursts or fail on stealthy slow attacks.
- Simplified trust model (client certs optional) reduces authentication guarantees in MVP.
- No full IDS integration; deeper protocol inspection is absent.
- Lack of centralized log correlation hinders cross-component forensic reconstruction.

Extension Toward a Smart-City IoT Security Framework

Identified Gaps and Required Components

While the MVP architecture captures many core technical controls (encryption, policy enforcement, anomaly detection, segmentation), a full Smart-City IoT security framework requires additional layers to ensure governance, risk management, scale, and resilience. The key gaps that must be addressed are:

1. **Explicit Threat Modeling:** Clearly define stakeholders (city operators, service providers, citizens), critical assets (sensors, telemetry streams, decision engines), adversary capabilities (physical compromise, supply chain manipulation, replay attacks, insider misuse), attack surfaces, and prioritized threat scenarios. This enables alignment of controls to real-world risks and supports structured mitigation planning.
2. **Risk Assessment & Prioritization:** Quantify threat likelihood and impact, map existing controls (and deficiencies) to those threats, and prioritize remediation. For example, create a risk matrix addressing unauthorized injection, distributed

denial-of-service (DDoS), firmware tampering, or data exfiltration, guiding investment in hardening and detection.

3. **Security Policies & Procedures:** Formalize operational policies for device onboarding/offboarding, certificate issuance and revocation, acceptable use, patching cadence, incident response, and audit reviews. These workflows ensure consistency, reduce human error, and provide an audit trail for compliance.
4. **Device Lifecycle Management:** Securely manage sensors and edge devices from provisioning through decommissioning. This includes secure bootstrap, signed and versioned firmware updates, tamper detection, and revocation mechanisms for lost or compromised units.
5. **Supply Chain & Physical Security:** Establish governance around trusted manufacturers, component provenance, anti-counterfeiting measures, and physical access controls for distributed infrastructure (e.g., hardened enclosures, tamper-evident seals, and location verification).
6. **Privacy / Data Governance:** Define data minimization, anonymization, retention, and access policies appropriate for smart-city telemetry, especially when data could be sensitive (e.g., location, usage patterns). Incorporate regulatory requirements and citizen privacy expectations into data handling rules.
7. **Incident Response & Recovery:** Develop playbooks for detection, containment, eradication, and recovery across likely incident types (e.g., compromised gateway, volumetric flooding, unauthorized sensor spoofing). Include escalation paths, rollback procedures, and post-incident reviews.
8. **Audit & Compliance:** Implement tamper-evident, centralized audit trails with periodic review processes. Align controls to relevant standards or municipal regulations, enabling demonstrable compliance and accountability.
9. **Scalability & Federation:** Provide mechanisms for handling a heterogeneous and growing set of domains and devices—supporting federated trust across city departments, hierarchical policy propagation, and domain isolation to limit blast radius.
10. **Performance & SLA Metrics:** Define security-related service-level objectives, such as maximum acceptable telemetry latency under load, anomaly detection latency, policy refresh reliability, and system availability guarantees to inform capacity planning and operational monitoring.

Adaptation & Extension Recommendations

To evolve the MVP into a practical, municipal-grade Smart-City IoT security framework, the following concrete adaptations and extensions are recommended:

- **Threat Model Table:** Create a structured table enumerating assets, threats, adversary capabilities, existing defenses, and residual risk. Example entry:

Asset: telemetry integrity; *Threat*: spoofed sensor data; *Control*: policy-based allowlist + future mutual TLS; *Residual Risk*: key compromise.

- **Control Catalog Mapping:** Map each implemented and proposed control to a standard framework (e.g., NIST CSF functions: Identify, Protect, Detect, Respond, Recover), fostering interoperability with broader city-wide security governance.
- **Risk Register Template:** Introduce a living risk register with example entries (e.g., "Compromised sensor → false telemetry injection → mitigation: policy enforcement + planned mutual TLS; priority: high"). Include risk owner, likelihood, impact, and mitigation status.
- **Policy Lifecycle Formalization:** Extend the policy distribution mechanism to support signed policy blobs, versioning, rollback on misconfiguration, and efficiently disseminate differential updates to gateways while maintaining integrity and traceability.
- **Phased Roadmap for Improvements:** Organize the suggested enhancements into phases:
  Immediate: mutual TLS, centralized logging, consistent timestamps.
  Mid-term: adaptive anomaly detection, IDS/IPS integration, backpressure signaling.
  Long-term: federated identity, automated secret vaults, DDoS scrubbing proxies.
- **Metrics & Dashboard Specification:** Define what to observe (e.g., telemetry latency distributions, throttle event frequency, policy refresh success rate, anomaly detection rate) and specify alert thresholds. Provide design for a real-time dashboard to surface trends and deviations.
- **Incident Response Scenarios:** Codify typical response flows, for instance:
  Rate Flood Detected: gateway flags anomaly → throttle engaged → alert operations dashboard → initiate traffic shaping upstream → monitor recovery → log and postmortem.
- **Federated Policy & Identity Management:** For multi-agency smart-city deployments, devise mechanisms to federate trust and policy agreements, allowing departments to retain autonomy while sharing core telemetry infrastructure securely.
- **Synthetic Baseline Regression:** Build on the existing test harness to automate regression comparisons against known-good synthetic baselines, detecting behavioral drift early (e.g., unexpected latency spikes or changes in success rates).

Proposed Improvements

1. **Adaptive Anomaly Detection:** Replace static rate thresholds with dynamic, statistically-informed detectors (e.g., EWMA, clustering, or lightweight ML) that learn normal behavior over time, reducing both false positives and negatives as traffic patterns evolve.
2. **Mutual TLS Authentication:** Enforce client certificates for sensors to strengthen identity validation, eliminate the need for hostname-check disabling, and support certificate rotation and revocation.
3. **IDS/IPS Layer:** Integrate Snort/Suricata to provide signature-based and behavioral detection of known exploits and complex multi-stage attacks, complementing existing rate-based defenses.
4. **Versioned Policy Distribution:** Serve signed policy blobs with version metadata, enable rollback on misconfiguration, and use delta updates to minimize churn and ensure consistency across gateways.
5. **Backpressure Signaling:** Implement explicit upstream feedback mechanisms so sensors can slow their emission rates when gateway load is high, avoiding reactive dropping and smoothing load spikes.
6. **Centralized Logging/Correlation:** Aggregate logs into a unified system (e.g., ELK stack or vector) with consistent UTC timestamps and distributed tracing identifiers to enable root-cause analysis and faster incident response.
7. **Traffic Visualization Dashboard:** Provide real-time dashboards for operators showing latency trends, success rates, policy compliance, and anomaly alerts to aid rapid decision-making.
8. **Automated Regression Harness:** Enhance the embedded test runner to automatically compare current outputs against known good synthetic baselines, detect drift, and integrate into CI pipelines.
9. **DDoS Scrubbing Proxies:** Introduce front-line scrubbing or rate-limiting proxies to absorb volumetric amplification attacks before they reach the gateway, preserving core service availability.
10. **Secure Secret Provisioning:** Automate the lifecycle of certificates and keys using a secrets vault, enabling secure issuance, rotation, expiration handling, and audit logging to eliminate manual key management risks.

Conclusion

The implemented MVP architecture demonstrates a layered defense and scalable telemetry pipeline in simulation. Synthetic results show predictable behavior under load and structured mitigation of basic attack classes. Although live execution data was not captured, the synthetic evaluation provided actionable insight into system behavior, bottlenecks, and security posture. Importantly, the extension toward a Smart-City IoT security framework fills critical governance, risk, and operational gaps - introducing explicit threat modeling, risk assessment, policy lifecycle, device lifecycle management, incident response preparation, and federated trust concepts. The proposed phased roadmap (immediate, mid-term, and long-term) creates a pragmatic path to production readiness, improving robustness, observability, identity assurance, and adaptive defense. Next steps are: performing formal threat modeling and risk prioritization; deploying in a staging environment with real telemetry; integrating centralized logging and trace correlation; enforcing mutual TLS; implementing adaptive anomaly detection; and building real-time dashboards and incident response playbooks to validate and iterate on the framework.