Lecture 8

Friday, 10 March 2023 08:33

OSCoAP is more efficient / usable in low-resource environment than DTLS

IoT uses mostly UDP and CoAP
- UDP ↑ transport layer
- CoAP ↑ application layer — could roughly be HTTP adjusted for low-power environment

TCP is traditionally used for reliable connections

UDP is connection-less, and does not require maintaining connection (and resources required for this)

delivery + duplicate protection not guaranteed → delegated to application layer

HTTP assumes TCP at transport layer ...

the border router generally has a proxy translating between HTTP and CoAP

servers may also use CoAP

CoAP supports use as publish + subscribe protocol

CoAP response codes are similar to those of HTTP

tokens are used to correlate responses with requests

CoAP logically has two 'layers':
- message layer
- request/response layer

CoAP header structure

the message ID is used to detect duplicates

confirmable messages require an ACK with the same message IDs (reset options are used when (non-) confirmable messages are not supported)

non-confirmable messages do not

CoAP has caching features

CoAP resource discovery

multicast address for list of available resources: GET /.well-known/core

CoAP has 4 security modes

0. no security (delegate to another layer) — note that security at MAC layer allows intermediate nodes to access content
1. pre-shared key — pre-programmed symmetric key
2. raw public key — asymmetric key pre-programmed, no PKI
3. certificates — X.509 certificates signed by CA, used for key exchange

ECDSA for authentication

ECDHE (with ephemeral keys) for key agreement

CoAP recommends the use of DTLS

CoAP does not specify headers, setup of secure layer, or how to use keys

DTLS and TLS are similarly different as CoAP and HTTP

DTLS ↓ datagram transport layer security

DTLS provides end-to-end security 'at transport layer' on top of UDP/TCP

→ compatible with TLS v1.3 → roughly equivalent security guarantees, without order protection / non-replayability

DTLS has explicit sequence numbers

DTLS (in particular handshake) allows for re-ordering packets & silently discarding messages

DTLS adds ACK messages

DTLS messages can be fragmented & re-assembled

DTLS adds HelloRetryRequest to avoid MitM & DoS

packet loss management is based on a re-transmission timer

initially 9 s, retransmits upon expiry, then double timer

record layer:
- sequence number allow re-ordering, decrypting individual records, protecting against replay
- epoch allows identification of key changes

only block ciphers, no stream ciphers are allowed

handshake protocol same as TLS v1.3, but adjusted for UDP

DTLS handshake
- clienthello — e.g. supported cipher suites
- hellorequestrequest + cookie
- clienthello + cookie → nonce which must be re-sent with cookie; used to prevent storm of grouped packets to spoofed clienthello IP
- serverhello — selected cipher suite

messages are grouped to conserve energy

DTLS is the main protocol in its niche, but it's not the most efficient

object security for CoAP — OSCoAP

multiple clients connect to one server

encryption / signing of CoAP message using pre-shared key material

authorization server puts keys (and access secrets) on client

and puts pre-signed resources (per client) (and access secrets) on server (since they have different keys)

client asks resource

server returns encrypted (and signed) resource

key is computed from access secret, message ID and sender ID

advantage: does not require handshake & record protocols

---

attacks

CoAP blocking — e.g. (selectively) blocking requests/responses

mitigation through e.g. encryption (does not always work), attack detection through confirmable messages, IDS

CoAP request delay — delay reaching destination

mitigating through IDS, shortening transmission window (but gives reduced robustness to network fluctuations)

CoAP relay attacks — records packets then relay them to other devices

car key with remote unlock

mitigation: measure RTT (round-trip time), use (signed) GPS location data

---

MQTT

message queuing telemetry transport

publish / subscribe messaging transport protocol

broker ('server'): hosts list of topics + subscribers, redirects updates on topics to subscribers

nodes:
- subscribers — subscribe to topics of interest, receives updates on topics
- publishers — publishes updates on topic, can also be a subscriber

MQTT runs over TCP not UDP → messages may not receive out of order in MQTT

MQTT-SN runs over UDP and has smaller headers and the concept of sleeping clients

(SN ↓ sensor network)

MQTT security features
- authentication → first username + password, later challenge-response
- authorization → use token / username to check permissions
- integrity → hashes or suggestion to use VPN
- confidentiality

many possible anomalies exist in MQTT