MKA Meets Multidrop: Optimizing Time-to-Key-Agreement on 10BASE-T1S Ethernet

Jonathan Ndop[1,3], Isaac Molina[2], Guillermo Oliver[2], Friedrich Wiemer[1], and Axel Sikora[3]

[1] Robert Bosch GmbH, Stuttgart, Germany, firstname.lastname@de.bosch.com
[2] Analog Devices, Valencia, Spain, firstname.lastname@analog.com
[3] Institute of Reliable Embedded Systems and Communication Electronics (ivESK), Offenburg University of Applied Sciences, Offenburg, Germany, firstname.lastname@hs-offenburg.de

Abstract. The MACsec Key Agreement protocol, defined in IEEE 802.1X, manages and distributes ephemeral Secure Association Keys for Ethernet links protected with MAC security (IEEE 802.1AE). Prior work has shown that baseline MKA may scale poorly on shared medium Ethernet multidrop links and that formal worst-case bounds significantly exceed automotive startup targets, motivating alternative solutions such as In-line Key Agreement. However, in practice, to preserve compatibility, integration effort, and alignment of standardization, automotive systems are more likely to optimize a MACsec/MKA architecture than to completely replace it. This paper presents novel automotive MKA optimizations targeting secure startup times on shared medium networks and evaluates them through detailed network simulations.
Unlike previous work focused on baseline MKA or deterministic worst-case analysis, we study the full startup-time distribution of the optimized protocol under realistic startup scenarios. We quantify the effect of the proposed optimizations on Time-To-Key-Agreement and show how the resulting empirical distributions can be translated into conservative simulation-derived practical startup-time bounds suitable for OEM timing budgets. The resulting bounds are intended as simulation-derived engineering bounds under the modeled startup assumptions, complementing formal worst-case analysis with distributional information on typical, tail, and upper-end behavior.

Keywords: Automotive Ethernet · Key Agreement · MKA · MACsec

1 Introduction

Automotive Ethernet is rapidly becoming a key communication technology in modern in-vehicle networks, and this evolution is accompanied by a growing need for built-in link-layer security. MAC Security (MACsec), defined in IEEE 802.1AE [1], together with the MACsec Key Agreement protocol (MKA) defined in IEEE 802.1X [3], offers a standards-based mechanism to establish and distribute Secure Association Keys (SAKs) and protect Ethernet communication.

For automotive deployment, however, security alone is not sufficient: Original Equipment Manufacturers (OEMs) also require secure communication to become available within a predictable, tightly bounded startup time. This is particularly challenging on shared 10BASE-T1S networks, where key-agreement frames from multiple nodes are serialized on the same shared medium and compete with one another, as well as with ordinary traffic [8]. The scaling issue is therefore mainly communication-driven: more nodes create more MKA Protocol Data Units (MKPDUs) and more shared-medium waiting time before all participants reach a secured state, and deterministic worst-case bounds for baseline MKA can become large in such settings.

Practical automotive systems are unlikely to replace MKA with a different key-management architecture; a more realistic path is to retain a standards-based MACsec/MKA stack and improve its startup behavior through targeted optimizations, as already reflected in the OPEN Alliance automotive MKA work [9]. Existing worst-case analyses focus on baseline protocols and deterministic abstractions, whereas system designers also need to understand how optimized MKA behaves across many startup scenarios, how large the tail of the startup-time distribution becomes, and how conservative design bounds can be extracted from this behavior.

This paper addresses that gap. We study optimized automotive MKA on shared 10BASE-T1S networks using detailed network simulations implemented in RTaW-Pegase [5], with explicit modeling of automotive startup behavior and protocol-level optimization mechanisms. Our contribution is twofold. First, we introduce automotive MKA optimizations, combining protocol refinements with implementation-level enhancements aimed at reducing secure-startup latency and unnecessary message bursts. Second, we define a rigorous secure-startup metric and characterize its behavior statistically across a controlled design space, identifying the mechanisms that dominate average, tail, and upper-end behavior, so that conservative startup bounds can be derived for OEM timing budgets and architectural design decisions.

The simulation study is intended to isolate the protocol-level timing effects of the proposed optimizations under controlled automotive startup conditions. The resulting timing distributions therefore characterize the modeled 10BASE-T1S startup scenario and are intended to complement, rather than replace, formal timing analysis and hardware validation.

The remainder of this paper is organized as follows: Section 2 presents the related work, Section 3 provides the prior knowledge, Section 4 presents our optimizations, Section 5 shows the system and simulation model. In Section 6 we discuss results and conclude in Section 7.

2 Related Work

In this section, we present a representative selection of prior work related to secure key establishment in automotive and embedded networks. A first distinction is the trust-establishment model. Several secure communication protocols rely on

asymmetric authentication, certificates, or public-key key exchange to establish fresh session keys. Automotive MKA, in contrast, is commonly considered with pre-shared keys (PSKs), especially in the OPEN Alliance automotive profile [9], so that startup does not depend on a full certificate-based handshake. Our work therefore does not study the initial provisioning of long-term trust material; it focuses on the timed distribution and activation of SAKs once the MKA participants already share the required trust.

Secure in-vehicle communication has been studied through several complementary approaches, including TLS-based protection [13] for Automotive Ethernet, group key agreement for VANETs [7], and distributed group key agreement for constrained cyber-physical buses [11]. These works establish the relevance of dynamic key establishment, group membership handling, and resource-aware security mechanisms in automotive and embedded networks. More recent benchmarking work [6] compares MACsec/MKA, IPsec/IKEv2, and TLS on automotive-grade hardware, while broader systematization work [12] shows that newer in-vehicle buses like 10BASE-T1S and CAN XL do not automatically eliminate security challenges. However, these contributions do not analyze the timing distribution of MKA startup on shared 10BASE-T1S multidrop links. The closest work to this timing question is therefore the formal worst-case analysis of [8] on shared 10BASE-T1S, discussed next.

2.1 Formal Worst-Case Analyses

The recent study by Ljubevska et al. [8] investigates the timing behavior of key agreement protocols over shared 10BASE-T1S multidrop networks. Using network calculus, a formal worst-case analysis is conducted. The focus is on a comparison between baseline MKA and Infineon's In-Line Key Agreement (IKA) protocol under a deterministic queueing model with PLCA service curves and competing control/data traffic. In that framework, the authors derive worst-case startup bounds as a function of message size, node count, and traffic prioritization. It is then shown that baseline MKA scales poorly in multidrop settings, especially for small MACsec PDUs (MPDUs) where repeated blocking and reduced service share at the Key Server become dominant.

That work is important because it establishes, in a formal and protocol-aware way, that baseline MKA can become problematic on shared 10BASE-T1S networks, with the worst-case behavior exceeding automotive startup expectations. At the same time, the analysis remains intentionally conservative. The authors model completion at the point where the head node sends its final key-agreement message, assuming that each such message experiences a worst-case delay then multiplies it by the number of broadcasts. Furthermore, the paper highlights that the framework does not quantify how frequently such worst cases occur.

Our work starts where that analysis stops. Instead of comparing baseline MKA to a different protocol family, we focus on optimizing MKA itself for automotive scenarios. Moreover, rather than deriving deterministic worst-case bounds from an abstract service model, we use simulation to model realistic execution constraints of automotive software components while capturing MKA

protocol execution, startup ordering, and optimization dynamics in more detail. This makes it possible to examine maximum timing behavior as well as the full distribution of startup times and communication overhead, which is particularly relevant for engineering startup budgets in OEM design.

2.2 Automotive MKA Standardization

Another relevant line of work is the ongoing OPEN Alliance effort to define automotive profiles for MACsec [10] and MKA [9]. In that work, startup latency is treated as a requirement of utmost priority: Automotive MKA is expected to reduce worst-case startup time down to 30 ms for a single switched Ethernet link [9], and worst-case startup is defined as the interval from link-up until both sides reach a secured receiving and transmitting state. The same work also fixes the general optimization direction by simplifying baseline IEEE 802.1X MKA, preferring pre-shared keys (PSKs), and introducing timing-oriented measures such as immediate message transmission on peer-list changes, faster group formation sequences and fixed key server roles.

This is directly relevant to our contribution because it shows that optimized Automotive MKA is already an active standardization topic, not just a local implementation choice. The current OPEN Alliance scope is centered on point-to-point or switched automotive Ethernet. Shared 10BASE-T1S can be understood as the group case of the same startup problem: a point-to-point link corresponds to the special case $N = 2$. The same optimization principles remain applicable, but their timing impact must be re-evaluated when peer-list evolution, SAK distribution, rekeying, and PLCA medium access involve more than two nodes. This is the gap addressed by our work.

3 Background

This section recalls MACsec and MKA protocol elements and system constraints that motivate the present study. The emphasis is on the pieces that matter for startup timing on automotive Ethernet, especially on shared 10BASE-T1S links.

3.1 MACsec and MKA Essentials

MACsec protects Ethernet frames at Layer 2, while MKA provides the control-plane procedure used to establish and maintain the cryptographic context required by MACsec. MKA is based on the Extensible Authentication Protocol over Local Area Network (EAPOL) [3]. At a high level, long-lived trust is anchored in a Connectivity Association Key (CAK). From that trust anchor, the participants derive and distribute short-lived Secure Association Keys (SAKs), which are the keys actually used to protect data traffic by means of unidirectional secure channels (SCs) between each communicating peer. Within a Connectivity Association, one participant acts as Key Server (KS) and coordinates SAK generation and distribution. Peers maintain protocol state about other members, typically

Phase	#	Dir	MI+MN	LPL : PPL :	Other parameter sets
Liveliness	1	A → B	A+1 :	: :	
	2	B → A	B+1 :	: A+1 :	
SAK Dist.	3	A → B	A+2 : B+1 :		: DistSAK; SAKuse lrx=1, ltx=0
	4	B → A	B+2 : A+2 :		: SAKuse lrx=1, ltx=0
TX Enabled	5	A → B	A+3 : B+2 :		: SAKuse lrx=1, ltx=1
	6	B → A	B+3 : A+3 :		: SAKuse lrx=1, ltx=1

Fig. 1. Simplified MKA exchange showing liveliness determination, SAK distribution, and SAK-protected communication. Peer A is the Key Server, peer B the responder; MI+MN identifies the sender by Member Identifier and Message Number. Notation from [3, §9.17].

through potential and live peer lists, and they exchange periodic control MACsec Key Agreement PDUs (MKPDUs) at a nominal rate of $2s$ ("Hello" timer), to establish liveliness and synchronize key state. Once a new SAK is distributed by the KS, peers confirm installation and usage through the corresponding SAK Use parameter set.

MKA startup is a sequence of state-building exchanges between peers. Fig. 1 shows the simplified two-peer case used here, assuming a fixed Key Server role (as recommended in [9]). Each participant first announces itself with an MKPDU carrying its randomly chosen Member Identifier (MI), and a Message Number (MN), and records the sender of any received MKPDU in its Potential Peer List (PPL). A peer is moved from the PPL to the Live Peer List (LPL) once it sees its own MI echoed back in a MAC-protected MKPDU. Thus, the own MI acts as a cryptographic challenge and the inclusion in the responders' peer list as the corresponding freshness proof. The Key Server then distributes a fresh Secure Association Key (SAK) together with a SAK Use parameter set; the receiving peer installs the key and acknowledges with `SAK Use.lrx=TRUE`. Once all live peers have confirmed, the Key Server enables transmission by setting `SAK Use.ltx=TRUE` in its next MKPDU, which the peers mirror to complete key agreement.

3.2 Shared Medium 10BASE-T1S Specifics

The 10BASE-T1S PHY was introduced in IEEE 802.3cg-2019 [2]. It supports a half-duplex shared medium mode, also referred to as multidrop, using a single balanced twisted-pair of conductors. The timing problem becomes sharper on these links because MKA frames from all nodes are serialized on the same medium: a delayed or additional MKPDU does not only affect the sender, it can also postpone the transmit opportunities of other participants. Unlike point-to-point Ethernet, multiple nodes share the same medium and therefore share access to transmission opportunities.

In 10BASE-T1S, access is coordinated through Physical Layer Collision Avoidance (PLCA) defined in IEEE 802.3 [4] clause 148, which organizes transmit opportunities in a round-robin fashion. All nodes get a Transmit Opportunity (TO) in each communication cycle, in which it is allowed to send a frame or multiple frames in burst mode. The duration of the transmit opportunity of each node will vary depending on the size of the frame sent by the node. The principle is shown in Fig. 2 with three communication cycles for four nodes on a bus, a TO is dropped for a non-transmitting node after the default timer of 2.4 μs. This removes collisions, but does not remove competition: every control frame still consumes a transmit opportunity, and every participant can delay the others.

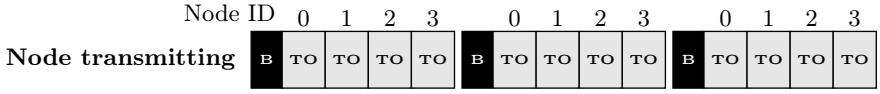

Fig. 2. Illustration of PLCA cycles with beacon (B) and transmit opportunities (TOs).

That distinction matters for MKA. In a point-to-point link, secure startup is a two-party exchange. In a multidrop segment, startup is a group key agreement. Peer discovery, proving liveness, and SAK distribution interact with the order in which nodes come online and with the timing of access to the shared medium. As a result, protocol bursts that are relatively harmless in a point-to-point setting can become much more consequential on a shared segment. This is precisely why earlier work [8] identified a real-time tension between key-agreement traffic and ordinary traffic in multidrop networks.

Next, we look at optimizations from [9] and ours for shared-medium MKA.

4 Automotive MKA Optimizations for 10BASE-T1S

We group the optimizations by their status: those already mandated by the OPEN Alliance AutoMKA v1 profile [9], and those we propose for v2.

4.1 AutoMKA v1 optimizations

Optimization 1 (Ramp-up phase – AutoMKA-03004). *During startup, replaces the fixed 2 s* `HelloTime` *by an exponentially increasing sequence:*

$$r_0,\ r_1 = 2r_0,\ r_2 = 4r_0,\ \ldots,\ r_i = 2^i r_0,$$

falling back to `HelloTime` *cycles, once* r_n *got bigger than* `HelloTime`*.*

The ramp-up phase results in MKA participants sending MKPDUs quicker, after pauses of r_i each. The starting value of r_0 is user configurable and suggested to be in the range of 10 to 20 ms. Without the ramp-up phase, a single lost

early MKPDU delays discovery by up to one `HelloTime`, which is well above automotive startup targets. The change affects timing only; MKPDU contents are unchanged.

Optimization 2 (Immediate response – AutoMKA-03003). *Any MKPDU that changes the local protocol state (new peer in PPL/LPL, new `DistributeSAK`, change of `SAK use`) triggers the node to send an extra MKPDU instead of waiting for the next periodic hello.*

This avoids state-change latencies of up to one `HelloTime` and is the main reason AutoMKA v1 reaches sub-second key-agreement on a 2-peer switched link.

Optimization 3 (Reset and Late-join rekey – AutoMKA-03006/-03011). *A received message from a known peer with a different MI, implies a reset of that peer has happened. All automotive MKA instances will react by adding the new peer in the PPL and removing the old one from the PPL or LPL.*

After the initial SAK has been distributed, any peer that becomes live but is not in the acknowledge-set of the current SAK triggers an immediate KS rekey (new SAK, new key number).

The reset optimization prevents a downtime of the link of up to 6s because the new peer does not yet have the new SAK. The Late join optimization allows group-key hygiene and so the late joiner can install a SAK it actually witnessed.

4.2 AutoMKA v2 proposals

In the following, we propose additional optimizations for a future AutoMKA v2 profile and briefly motivate each mechanism.

Optimization 4 (Retransmission policy). *Until initial key agreement completes, the node re-sends its last MKPDU at a short, fixed interval τ_r, e.g., $20ms \leq \tau_r \leq 30ms$, instead of relying on the hello/ramp-up cadence.*

Retransmissions are stopped based on the following conditions: the KS stops once it has at least one peer; a peer stops once the KS appears in its LPL. This replaces (rather than complementing) the v1 ramp-up phase, giving a simpler method compared to a specific ramp schedule, and is resource-efficient since retransmitted MKPDUs need no crypto recomputation.[4]

This optimization therefore primarily affects discovery latency and control-frame load. For the immediate-response triggers, we suggest the following adaptation.

Optimization 5 (Reduced peer bursts – restricted response). *A peer emits an MKPDU with its updated state (PPL and LPL) only on received frames that*

[4] This optimization is following M. Seaman's proposal, *MKA optimizations*, IEEE 802.1 contribution, March 2025. `https://www.ieee802.org/1/files/public/docs2025/x-seaman-mka-optimizations-0325-v06.pdf`

(i) originated from a key server (i.e., carry the key-server *flag)* and
(ii) either
(ii.a) do not yet list the peer in their LPL or
(ii.b) carry a `SAK use` *parameter set.*

This is driven by the need to reduce frames on the medium, making peers' answers more robust. Without this, every peer frame potentially triggers a response on all the other peers, multiplying per-cycle frame counts by N and wasting TOs. Only the gating predicate changes; the rule's priority is unchanged.

Optimization 6 (SAK-distribution with non-empty potential peer list). *The KS no longer waits for an empty PPL before distributing the first SAK; it distributes as soon as its LPL changes. On shared medium, startup times should depend as little as possible on the number of nodes or their boot-up sequence.*

Under the IEEE-conformant rule (section 9.8 of [3]), a single straggler in the PPL would otherwise delay the first SAK – and therefore all peers – until it becomes live or its PPL entry ages out. The trade-off is one extra rekey per straggler against a much shorter TTKA tail for early peers.

On peer side, the KS's MKPDU m distributing the SAK may contain updated PPL or LPL entries, which may itself lead to updates of the peer's peer lists. The peer accepts SAKs only from a KS it considers live. The liveliness state the peer uses as basis for this decision is the one *after* updating its peer lists based on the received m. In other words, Opt 6 above creates a window in which a peer is in the KS's PPL but not yet in its LPL when the KS distributes the SAK (in m). Three coordinated rules let such a peer install the SAK without waiting for a rekey: (a) the peer installs as soon as the KS appears in its own LPL, updated based on m, (b) the KS extends its expected acknowledge-set (received `SAK use`) from LPL peers to LPL $\cup$ PPL peers, and (c) to account for these peers that potentially move from the PPL to the LPL of the KS, the other peers must also remember the KS's PPL and LPL to create SCs accordingly for all peers once liveliness has been proven.

Optimization 7 (Formation window – KS-side burst suppression). *As a companion to PPL gating, when the KS first becomes ready to distribute it opens a fixed window W, e.g., of 12ms, during which further state changes update its peer lists but do not trigger additional* `DistributeSAK` *sends; at $t = W$ the KS emits a single consolidated* `DistributeSAK`*. The window applies only to* `DistributeSAK` *– periodic Hello MKPDUs and immediate responses still go out.*

This optimization, combined with Opt 6 directly determines when the Key Server forms a group and distributes a SAK. The goal of the formation window is to reduce the number of `DistributeSAK` frames and thus of SAK formations.

Optimization 8 (KS final `SAK use` timeout). *After distributing a SAK, the KS waits at most T_{N2}, e.g. ≈ 50 ms for* `SAK use.lrx` *from every peer in the KS's acknowledgment-set. Peers that have not acknowledged by the deadline are dropped from the LPL, the round terminates with* `SAK use.ltx` *on the reduced set, and the dropped peers are picked up by the next rekey.*

Peer aging by MKA Lifetime (typically 6 s, IEEE 802.1X-2020 §9.4.3) already handles persistent peer loss; T_{N2} avoids a long 6 s wait on every missed acknowledgment.

5 System and Protocol Model

The evaluation considers secure startup on a shared 10BASE-T1S segment. The objective of the model is to capture the timing behavior of optimized Automotive MKA during startup, including node boot-up order, protocol response rules, processing delays, and shared-medium access effects. The model is intentionally narrower than a full vehicle communication model: it focuses on the key-agreement phase and the protocol mechanisms that determine when nodes become able to communicate securely.

5.1 Network and Protocol Model

We model one node as the Key Server and all other nodes as MKA peers. Nodes become active with randomized startup offsets, so each run represents a different startup ordering. A bounded clock-drift configuration is applied per node to avoid assuming perfectly synchronized participants. During the startup phase, the traffic of interest is the MKA control traffic required for peer discovery, liveliness determination, SAK distribution, and SAK-use confirmation. No additional application traffic is injected. This isolates the timing cost of the key-agreement procedure itself, without mixing it with unrelated payload contention.

Processing is not treated as instantaneous. Each outgoing MKPDU is delayed by a protocol-processing component and a small jitter term. Operations involving SAK generation, SAK installation, or SAK-use transition include an additional key-handling delay. This abstraction represents the practical fact that Automotive MKA startup is affected by frame exchange on the wire, as well as software scheduling, cryptographic processing, HSM interaction, and possible external-PHY configuration latency. The exact numerical values are treated as simulation parameters so that sensitivity to implementation timing can be explored.

The optimized protocol follows the usual MKA progression from peer discovery to liveliness, SAK distribution, and SAK-use confirmation, however with our automotive-specific optimizations 1 to 8 applied. The model also includes the final SAK-use transition in which nodes indicate readiness to use the installed SAK for secure transmission.

5.2 Simulation Environment and Recorded Metrics

The simulations are implemented in RTaW-PEGASE [5] as event-driven protocol models. This choice is appropriate as the behavior of optimized MKA on a shared segment is shaped by interactions that are difficult to reduce to message counts alone: startup offsets, peer-list evolution, execution-cycle granularity, formation-window timing, PLCA access order, retransmissions, and rekey events.

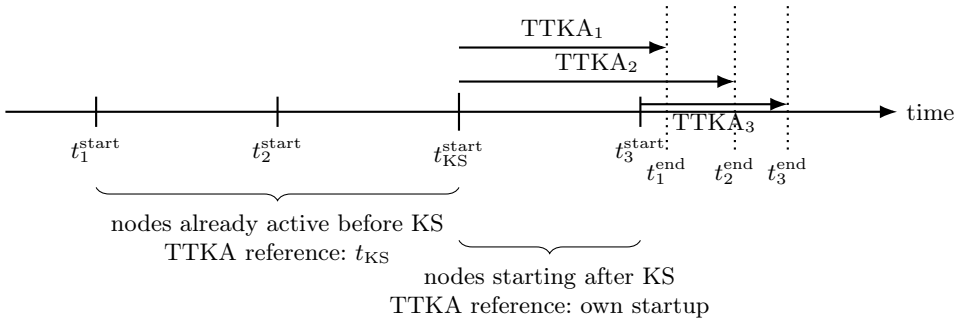

Fig. 3. TTKA reference convention on a 10BASE-T1S multidrop segment. Nodes that start before the Key Server are timed from the Key Server startup instant; nodes that start after the Key Server are timed from their own startup instant.

The simulator models the order and timing of protocol events, frame scheduling, node startup, processing delay, and the shared-medium communication behavior needed for the evaluation. It does not aim to reproduce every micro-architectural detail of a specific ECU or PHY implementation, nor does it model arbitrary physical faults or full application traffic. We do, however, model a typical software architectural implication from MKA deployments in a typical AUTOSAR-based system: in these, software components are typically executed cyclically, with a system-designer-configurable execution cycle. Typical execution cycles for software components are 5 to 10 ms, while cycles as short as 1 ms are possible, too. We denote the execution cycle by c.

The purpose of our simulation is a controlled timing evaluation of the key-agreement mechanism.

The main timing metric is the *Time-to-Key-Agreement* (TTKA), which we define at node level. For a given node, key agreement is complete when that node has completed the first secure-startup sequence for the first SAK it actually uses. In the protocol variant studied here, this means that the node has received and installed the relevant SAK and has completed the associated SAK-use signaling required to enter the secure communication state. This is the "Transmission Enabled" step described by Fig. 1. This endpoint is deliberately stricter than a metric that stops when the Key Server sends a final control message, since secure communication is only useful once the receiving node has completed its own transition.

Formally:

Definition 1 (Time-to-Key-Agreement). *For each node i, let t_i^{start} denote the startup time of node i, and t_i^{end} be the time at which node i completes the transmission enabled exchange for the first SAK it uses. Let $t_{\text{KS}}^{\text{start}}$ denote the startup time of the Key Server.*

The TTKA of node i is defined as:

$$\text{TTKA}_i \coloneqq t_i^{\text{end}} - \max\left(t_i^{\text{start}},\, t_{\text{KS}}^{\text{start}}\right). \tag{1}$$

For the startup times, we denote by $m_{\max}$ the upper bound of the startup-offset interval, such that $\forall i : t_i^{\text{start}} < m_{\max}$.

This definition is illustrated on Fig. 3 and prevents early-starting nodes from being penalized for idle time that occurs before the protocol can actually proceed.

For scenarios with late joiners, our simulation also records the number of SAK distributions, since additional rekey events can affect both TTKA and control-plane load.

Together, these metrics (TTKA and number of SAK distributions) make it possible to evaluate whether an optimization reduces startup time, and also whether it does so by reducing the mean startup time or the worst observed startups.

6 Results and Analysis

The following results are obtained from simulation campaigns designed to analyse how optimized Automotive MKA behaves under different startup and scheduling conditions. We consider ($n = 2, 4, 8$) nodes. The two-node case provides the smallest group and a reference close to the point-to-point MKA case, four nodes represent an intermediate multidrop configuration, and eight nodes correspond to the maximum segment size supported on shared 10BASE-T1S [2]. We also vary the MKA execution cycle c, the maximum startup-offset interval m_{max}, whether the formation window (with 12 ms) optimization is enabled, and whether SAK distribution is constrained by the PPL non-empty optimization. For every parameter set, 1000 randomized startup runs are executed, and the resulting per-node TTKA is recorded. The 1000 runs per parameter set are used to characterize the main distributional effects and commonly observed tail behavior within the sampled scenario space, it is not intended as an exhaustive exploration of all possible timing alignments. The analysis therefore focuses on typical startup behavior and the spread and upper-end behavior induced by startup ordering and protocol timing.

6.1 TTKA Distributions

Fig. 4 compares the TTKA distributions for three protocol variants at $c = 5\,\text{ms}$ and $m_{\max} = 50\,\text{ms}$. Each row corresponds to a different network size, with $n \in \{2, 4, 8\}$, while each column corresponds to a different optimization configuration. The x-axis shows the TTKA-bins, with the y-axis reporting the number of samples per TTKA-bin. The relevant information is the horizontal spread and the shape of each distribution.

- With a 12 ms formation window enabled and the PPL non-empty optimization, the TTKA remains concentrated in a relatively compact interval for all tested network sizes. The formation window deliberately delays the Key Server response, while grouping peer-list updates before SAK distribution. As a result, most nodes complete within a narrow timing range, and the distribution remains stable as the number of nodes increases.

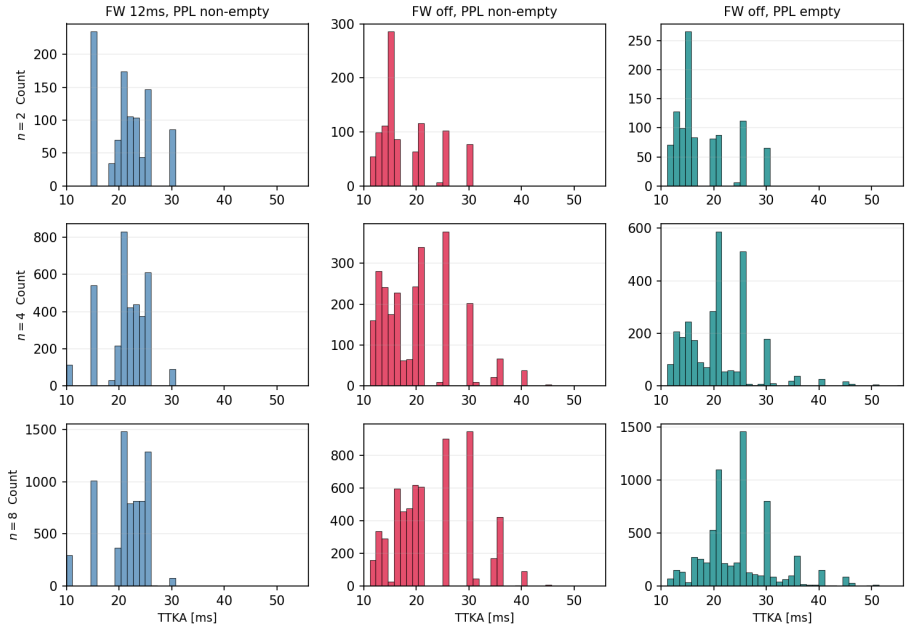

Fig. 4. TTKA distribution for different optimization variants and network sizes ($c = 5\,\text{ms}$, $m_{\max} = 50\,\text{ms}$).

- When the formation window is disabled while keeping the PPL non-empty optimization, the distributions become more visibly multi-modal. This is expected because the protocol no longer waits for a fixed grouping interval. Instead, the timing depends more directly on the relative order in which nodes start, prove liveliness, and trigger Key Server responses. The effect becomes clearer as the number of nodes increases. For $n = 4$ and $n = 8$, samples are spread over several timing bands, roughly corresponding to different numbers of MKA execution cycles needed before the node reaches the transmission enabled state.
- The PPL-empty condition produces the broadest and least regular distributions. In this variant, the Key Server waits until its potential peer list is empty before distributing the SAK. This delays the first key distribution when newly discovered nodes have not yet completed liveliness determination. As a result, even nodes that would otherwise be ready for key installation may be delayed by other nodes still present in the potential peer list. The effect is visible through the longer right tail, especially for $n = 4$ and $n = 8$, where samples extend over 50 ms.

Overall, Fig. 4 shows that the formation window introduces a trade-off: it is beneficial when the design objective is a tight startup bound, but can be counterproductive for the fastest individual completions.

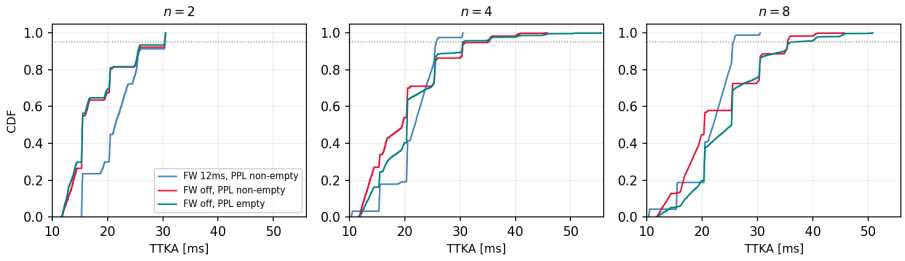

Fig. 5. TTKA CDF comparison across optimization variants and network sizes ($c = 5\,\text{ms}$, $m_{\max} = 50\,\text{ms}$).

The same trade-off is visible more directly in the CDF representation in Fig. 5. The curves show the fraction of TTKA samples that complete below a given time. For small networks, the three variants behave similarly, since peer-list interactions are limited. As the number of nodes increases, disabling the formation window allows a fraction of nodes to complete earlier, especially when SAK distribution is allowed with a non-empty PPL. However, this comes at the cost of a slower-rising curve and a longer right tail, meaning that a non-negligible fraction of nodes complete later. With the formation window enabled, the earliest completions are shifted slightly to the right, but the curve becomes steeper and reaches high completion ratios over a narrower interval. This confirms that the formation window mainly acts as a predictability mechanism: it does not minimize every individual TTKA, but it reduces dispersion and improves the upper-end behavior of the startup distribution. That means, on the variants without the formation window, the mean TTKA is smaller, but the formation window allows better worst observed TTKA values.

6.2 Rekeying Behavior

Fig. 6 relates the node startup offset to the SAK number first installed by that node. The x-axis is the startup-time difference with respect to the Key Server. Negative values correspond to nodes that are already active before the Key Server starts; positive values correspond to nodes that become active afterwards. The y-axis is the per-node TTKA.

– The structure of Fig. 6 is mainly caused by the discrete MKA execution cycle. For nodes starting before the Key Server, TTKA is measured from the Key Server startup instant, but their next MKPDU transmission depends on their local cycle phase. As the startup offset changes, this phase shifts modulo the cycle length, creating the diagonal sawtooth bands on the left side of the plots. For nodes starting after the Key Server, TTKA is measured from the node's own startup instant. Their completion time is then dominated by the number of MKA execution rounds needed after startup, which produces the horizontal bands on the right side.

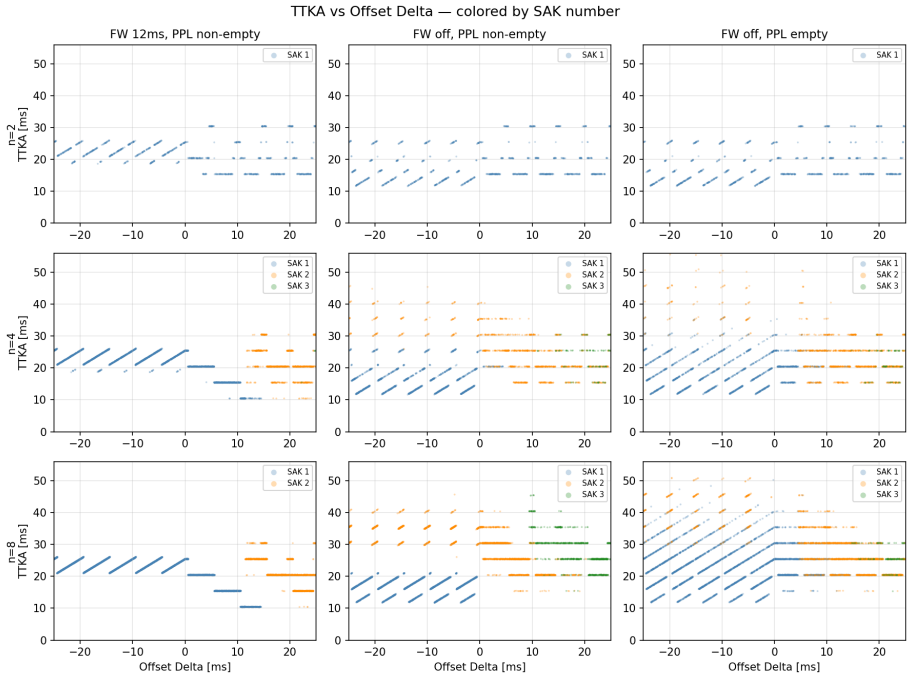

Fig. 6. TTKA vs. node-KS offset delta, colored by first installed SAK number ($c = 5\,\text{ms}$, $m_{\max} = 50\,\text{ms}$).

– The color coding shows how startup timing translates into rekeying. Most nodes that start before the Key Server install SAK 1, since they are already active and usually participate in the first liveliness and SAK-distribution wave. A small number of early-starting nodes install SAK 2 instead, mainly because their execution-cycle phase delays their liveliness proof beyond the first distribution decision. However, they do not miss the second distribution, so the negative-offset region contains SAK 1 and SAK 2, but not SAK 3. Nodes starting after the Key Server are more likely to miss the first wave; depending on their startup time and phase alignment, they install SAK 1, SAK 2, or, for larger networks such as $n = 8$, SAK 3. The presence of SAK 3 therefore indicates that startup has fragmented into more than two group formations. The formation window reduces this fragmentation, but only partially: in the $n = 8$ case it prevents the SAK 3 group, while SAK 2 installations remain. Thus, the formation window helps group nodes into earlier distributions, but it does not eliminate rekeying altogether.

Fig. 7 explores a related hypothesis: using different execution cycles for the Key Server and the peer nodes may be beneficial. The reason is that equal cycles can create persistent phase relationships. If the Key Server and peers execute with the same period, an unfavorable phase alignment can repeat across

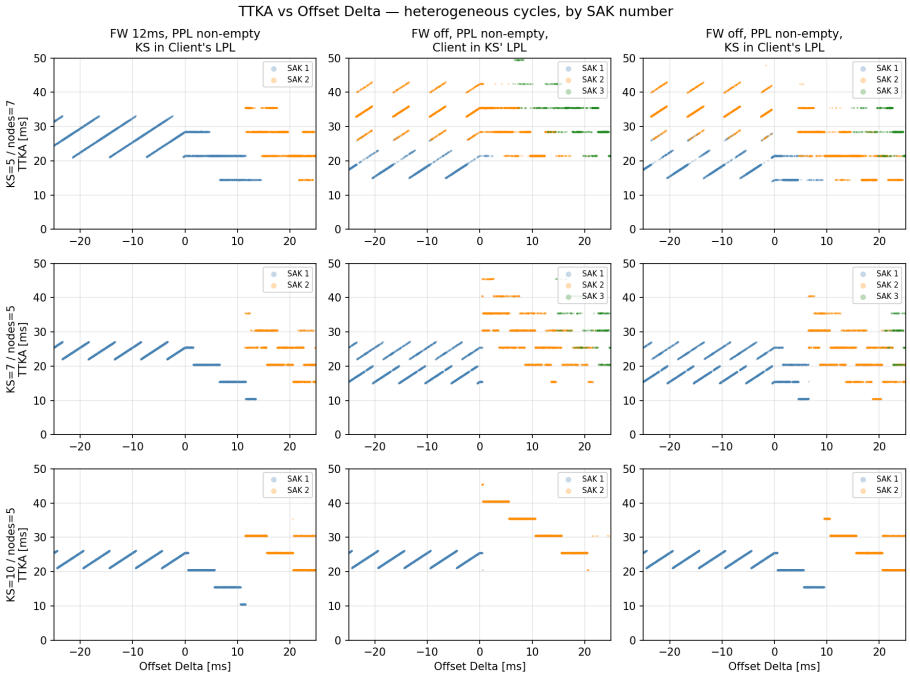

Fig. 7. TTKA vs. offset delta with heterogeneous execution cycles ($n = 8$, $m_{\max} = 50\,\text{ms}$), colored by SAK number.

several protocol steps. By contrast, heterogeneous cycles change the relative phase between the Key Server decision points and the peer response opportunities.

The experiments support this interpretation. When the Key Server executes faster than the peers, the Key Server can make distribution decisions before enough peer-side peer-list updates have arrived, which increases the chance of additional group formations. When the peers execute faster than the Key Server, they have more opportunities to respond and prove liveliness before the Key Server performs its next distribution decision. This acts like an implicit consolidation interval: the Key Server is slower to decide, while peers are faster to publish their updated state. In the shown cases, this reduces the occurrence of SAK 3 and confines most late nodes to SAK 2.

The trade-off is similar to the formation window trade-off: delaying Key Server decisions may slightly penalize the earliest possible completions, but it can reduce fragmentation of the agreement process and therefore improve predictability, especially in the variant without the formation window optimization. Between Fig. 6 and Fig. 7 the worst observed TTKA decreases from $45ms$ to $35ms$.[5]

[5] Fig. 6 last row, middle column vs Fig. 7 last row, right column.

7 Conclusion

This paper investigated whether Automotive MKA can be adapted to shared 10BASE-T1S multidrop links through targeted startup optimizations. We proposed and evaluated mechanisms such as restricted immediate responses, startup retransmission, SAK distribution with a non-empty PPL, formation window, and bounded waiting for SAK-use acknowledgments. The results show that optimized MKA startup is shaped by the protocol sequence itself, and by node startup order, execution-cycle phase, peer-list evolution, and rekeying behavior.

The PPL non-empty optimization removes a major source of blocking: the Key Server no longer has to wait until all potential peers become live before distributing the first SAK. In the evaluated ($n = 4$) and ($n = 8$) scenarios, retaining the PPL-empty condition produced the broadest TTKA distributions, with samples extending beyond $50ms$. The formation window plays a different role. With $12ms$, it may delay the fastest completions, but it groups peer-list updates into fewer SAK-distribution waves and produces a more compact TTKA distribution. It therefore acts primarily as a predictability mechanism rather than a best-case latency optimization.

The rekeying analysis confirms this interpretation. Nodes that are active before the Key Server usually join the first SAK distribution, while later nodes may require a second or third SAK depending on startup timing and execution-cycle alignment. In the ($n = 8$) experiments, enabling the formation window prevents the third SAK-distribution wave observed without it, although second-SAK installations remain. We also observed that heterogeneous execution cycles can provide a similar consolidation effect: when peers execute faster than the Key Server, they have more opportunities to publish updated state before the Key Server makes a distribution decision. In the evaluated configuration, this reduced the worst observed TTKA from approximately $45ms$ to $35ms$.

The study is subject to several limitations. The reported timing bounds are simulation-derived engineering bounds, not formal worst-case guarantees. They depend on the modeled startup assumptions, processing delays, clock-drift configuration, and protocol implementation choices. The simulations isolate MKA startup traffic and do not include arbitrary application traffic, physical-layer faults, packet loss, or detailed ECU/HSM micro-architectural behavior. These abstractions make it possible to study the optimizations in isolation, but further validation with mixed traffic and hardware measurements is needed before transferring numerical bounds directly into production timing budgets.

Overall, the results suggest that optimized MKA remains a viable solution for secure startup, even for larger groups of peers on shared 10BASE-T1S networks, provided that it is evaluated as a timed distributed protocol and not only as a message sequence. The main design lesson is that minimizing the fastest individual TTKA values and minimizing high-percentile or worst observed TTKA values are not the same objective. We expect the latter to be more relevant for OEM startup budgets. Future work should examine formal verification and cryptanalysis of MKA under these optimizations.

References

1. IEEE Standard for Local and metropolitan area networks-Media Access Control (MAC) Security. IEEE Std 802.1AE-2018 (Revision of IEEE Std 802.1AE-2006) pp. 1–239 (2018). https://doi.org/10.1109/IEEESTD.2018.8585421
2. IEEE Standard for Ethernet - Amendment 5: Physical Layer Specifications and Management Parameters for 10 Mb/s Operation and Associated Power Delivery over a Single Balanced Pair of Conductors. IEEE Std 802.3cg-2019 (Amendment to IEEE Std 802.3-2018 as amended by IEEE Std 802.3cb-2018, IEEE Std 802.3bt-2018, IEEE Std 802.3cd-2018, and IEEE Std 802.3cn-2019) pp. 1–256 (2020). https://doi.org/10.1109/IEEESTD.2020.8982251
3. IEEE Standard for Local and Metropolitan Area Networks–Port-Based Network Access Control. IEEE Std 802.1X-2020 (Revision of IEEE Std 802.1X-2010 Incorporating IEEE Std 802.1Xbx-2014 and IEEE Std 802.1Xck-2018) pp. 1–289 (2020). https://doi.org/10.1109/IEEESTD.2020.9018454
4. IEEE Standard for Ethernet. IEEE Std 802.3-2022 (Revision of IEEE Std 802.3-2018) pp. 1–7025 (2022)
5. RTaW-Pegase-Library. Tech. Rep. Version 4.6.3, RealTime-at-Work SAS, Nancy, France (Feb 2025)
6. Eugster, L., Wiemer, F., Pese, M.D., Hamad, M., Steinhorst, S.: Secure Automotive Ethernet: Implementing and Benchmarking MACsec, IPsec, and TLS (2026), preprint / forthcoming publication
7. Liu, L., Wang, Y., Zhang, J., Yang, Q.: A Secure and Efficient Group Key Agreement Scheme for VANET. Sensors **19**(3) (2019). https://doi.org/10.3390/s19030482, https://www.mdpi.com/1424-8220/19/3/482
8. Ljubevska, T., Zeh, A., Rama, D.E., Tindell, K.: Worst-Case Time Analysis of Key Agreement Protocols in 10BASE-T1S Automotive Networks. Cryptology ePrint Archive, Paper 2025/831 (2025), https://eprint.iacr.org/2025/831
9. OPEN ALLIANCE: Automotive MKA Specification for Automotive Ethernet and MACsec. Specification (Feb 2025), accessed: 2026-01-21
10. OPEN ALLIANCE: MACsec Specification for Automotive Ethernet. Specification (May 2025), accessed: 2026-01-21
11. Peters, S.N., Lautenschlager, L., Emeis, D., Lochert, J.: Secure Group Key Agreement on Cyber-Physical System Buses. arXiv preprint arXiv:2601.21966 (2026)
12. Serag, K., Tang, Z., Kim, S., Kumar, V., Dave, Tian, Zonouz, S., Beyah, R., Xu, D., Celik, Z.B.: SoK: Kicking CAN Down the Road. Systematizing CAN Security Knowledge (2025), https://arxiv.org/abs/2510.02960
13. Zelle, D., Krauß, C., Strauß, H., Schmidt, K.: On Using TLS to Secure In-Vehicle Networks. In: Proceedings of the 12th International Conference on Availability, Reliability and Security. ARES '17, Association for Computing Machinery, New York, NY, USA (2017). https://doi.org/10.1145/3098954.3105824, https://doi.org/10.1145/3098954.3105824

LLM usage considerations

This paper was written with the assistance of a large language model (LLM) for tasks such as drafting, editing, and formatting. Further, the LLM was used to refactor simulation code. To ensure validity of the simulation data, all results have been verified against an independent implementation of the same simulation model by a different author. All LLM outputs were carefully reviewed and edited by the authors to ensure accuracy, clarity, and adherence to scientific standards.