Índice general

v

Índice general

Índice de figuras

VII

Índice de figuras

Índice de tablas

IX

Índice de tablas

Capítulo 1

Introducción general

En este capítulo se presenta el marco de referencia y la justificación del trabajo Modernización de contadores de tránsito con comunicación bidireccional. Se expone el contexto que motivó su desarrollo, los problemas detectados en la infraestructura actual, una revisión del estado del arte, la propuesta de valor y el alcance del prototipo planteado.

1.1. Motivación

Este trabajo propone la modernización de los contadores de tránsito instalados en rutas nacionales mediante la renovación de su arquitectura de comunicaciones.

1.1.1. Contexto actual

Actualmente, los equipos registran el paso de vehículos y transmiten eventos al servidor central a través de enlaces GPRS tercerizados. No obstante, no admiten la recepción de comandos ni la obtención de diagnósticos remotos. Esta limitación reduce la capacidad operativa, incrementa los costos de mantenimiento y demora la resolución de fallas, debido a que todo ajuste o reparación requiere una intervención presencial [1] , [2].

1.1.2. Limitaciones del desarrollo previo

La propuesta se origina a partir del desarrollo de un contador de tránsito destinado a registrar y transmitir información en campo. Sin embargo, la conexión remota de este dispositivo presenta limitaciones en cuanto a su capacidad de transmisión de datos y carece de mecanismos de control remoto, lo que dificulta tanto la supervisión del funcionamiento como la actualización de sus parámetros. Frente a estas restricciones, surge la necesidad de modernizar el sistema mediante la incorporación de una comunicación bidireccional confiable [3], [2].

El análisis del sistema vigente permitió identificar las siguientes limitaciones que motivan el rediseño:

- Comunicación unidireccional. Los contadores envían datos al servidor, pero no existe un canal para enviar configuraciones, consultas y comandos desde el servidor hacia los equipos. Esta limitación impide realizar diagnósticos remotos y ejecutar acciones correctivas sin presencia física.

1

Capítulo 1

Introducción general

En este capítulo se presenta el marco de referencia y la justificación del trabajo Modernización de contadores de tránsito con comunicación bidireccional. Se expone el contexto que motivó su desarrollo, los problemas detectados en la infraestructura actual, una revisión del estado del arte, la propuesta de valor y el alcance del prototipo planteado.

1.1. Motivación

Este trabajo propone la modernización de los contadores de tránsito instalados en rutas nacionales mediante la renovación de su arquitectura de comunicaciones. Actualmente, los equipos registran el paso de vehículos y transmiten eventos al servidor central a través de enlaces GPRS tercerizados. No obstante, no admiten la recepción de comandos ni la obtención de diagnósticos remotos. Esta limitación reduce la capacidad operativa, incrementa los costos de mantenimiento y demora la resolución de fallas, debido que todo ajuste o reparación requiere una intervención presencial [1, 2].

La propuesta se origina a partir del desarrollo de un contador de tránsito destinado a registrar y transmitir información en campo. Sin embargo, la conexión remota de este dispositivo presenta limitaciones en cuanto a su capacidad de transmisión de datos y carece de mecanismos de control remoto, lo que dificulta tanto la supervisión del funcionamiento como la actualización de sus parámetros. Frente a estas restricciones, surge la necesidad de modernizar el sistema mediante la incorporación de una comunicación bidireccional confiable [3, 2]. Para este propósito se plantearon tres metas principales: en primer lugar, garantizar la entrega de eventos, incluso en condiciones de conectividad intermitente. En segundo lugar, implementar mecanismos de encolamiento y reintento en el dispositivo que eviten duplicaciones y pérdidas de información y, por último, habilitar la ejecución remota de comandos y la actualización de parámetros desde un servidor central, con una confirmación explícita del estado del equipo.

Los objetivos específicos buscan fortalecer la autonomía operativa de la institución y a reducir costos recurrentes asociados a la dependencia de proveedores externos de conectividad [4, 5]. En particular, el trabajo tiene por objetivos:

- Disminuir la dependencia de enlaces tercerizados mediante una arquitectura configurable.
- Permitir la modificación remota de parámetros operativos y la carga de ajustes sin requerir desplazamientos.

- Dependencia de proveedores GPRS tercerizados. La dependencia de servicios contratados genera costos recurrentes y limita el control sobre la calidad y disponibilidad de la conectividad.
- Imposibilidad de actualización remota. Cualquier modificación de parámetros o ajustes de operación requiere intervención en el sitio. Esto incrementa tiempos de mantenimiento, costos logísticos y complica la aplicación rápida de mejoras.
- Falta de telemetría y diagnóstico preventivo. No se dispone de métricas sistemáticas sobre el estado operativo de los equipos (batería, temperatura, errores de hardware o comunicación).
- Riesgo de pérdida de datos ante conectividad intermitente. La ausencia de mecanismos de encolamiento persistente y de políticas claras de reenvío eleva la probabilidad de pérdida o duplicación de eventos cuando la red es inestable.

Estas deficiencias afectan la calidad del servicio de monitoreo, reducen la eficiencia operativa y constituyen los requisitos funcionales que orientan el diseño del prototipo.

1.1.3. Impacto esperado

La modernización permite reducir los desplazamientos de personal técnico y acortar los tiempos de resolución de incidentes con la consiguiente disminución de costos logísticos. La incorporación de telemetría facilita la planificación de intervenciones preventivas en lugar de responder exclusivamente a fallas, lo que optimiza la disponibilidad del servicio y la calidad de los datos recolectados. Además, la adopción de protocolos estandarizados y componentes de código abierto promueve la escalabilidad y la replicabilidad de la solución en distintos tramos de la red vial [4] ,[5] ,[6].

1.1.4. Diseño conceptual

La propuesta separa de forma explícita el transporte de mensajes (broker MQTT) de los servicios de aplicación (API REST, almacenamiento y frontend). Esta separación facilita la interoperabilidad con plataformas institucionales existentes y habilita opciones de despliegue flexibles: uso de brokers externos, instalación de brokers locales o modelos híbridos según políticas institucionales y condiciones de conectividad.

1.2. Objetivos

1.2.1. Objetivo general

Modernizar los contadores de tránsito en rutas nacionales mediante la renovación de su arquitectura de comunicaciones para habilitar comunicación bidireccional confiable, control remoto y telemetría de estado.

1.2.2. Objetivos específicos

- Garantizar la entrega de eventos aun con conectividad intermitente.

- Incorporar telemetría de estado (nivel de batería, temperatura y códigos de error) para facilitar el mantenimiento preventivo.
- Validar la viabilidad técnica, la robustez operativa.

El alcance técnico incluye la adaptación de un contador existente para comunicarse bidireccionalmente por GPRS, que emplee MQTT como protocolo de mensajería ligera y confiable. Asimismo, comprende el desarrollo de un backend que reciba, procese y persista los eventos en una base de datos relacional. Además, una interfaz web básica que permita la visualización en tiempo real y el envío de comandos hacia los dispositivos. Se priorizan las funciones que validen la cadena completa: adquisición de eventos desde el sistema de detección, encolamiento local con política FIFO, reenvío seguro cuando haya conectividad, recepción y ejecución de comandos y verificación del estado del dispositivo tras cada acción [1].

Desde la perspectiva de operación y mantenimiento, la modernización permite reducir los desplazamientos de personal técnico y acortar los tiempos de resolución de incidentes con la consiguiente disminución de costos logísticos. La incorporación de telemetría facilita la planificación de intervenciones preventivas en lugar de responder exclusivamente a fallas, lo que optimiza la disponibilidad del servicio y la calidad de los datos recolectados. Además, la adopción de protocolos estandarizados y componentes de código abierto promueve la escalabilidad y la replicabilidad de la solución en distintos tramos de la red vial [6, 7, 8].

Por último, la propuesta separa de forma explícita el transporte de mensajes (broker MQTT) de los servicios de aplicación (API REST, almacenamiento y frontend). Esta separación facilita la interoperabilidad con plataformas institucionales existentes y habilita opciones de despliegue flexibles: uso de brokers externos, instalación de brokers locales o modelos híbridos según políticas institucionales y condiciones de conectividad. En síntesis, el trabajo busca ofrecer una solución técnica y económica que permita gestionar de manera proactiva y eficiente la infraestructura de conteo de tránsito que proporcione una base verificable para su posterior escalado e integración institucional.

1.2. Limitaciones sistema actual

El análisis del sistema vigente permitió identificar las siguientes limitaciones que motivan el rediseño:

- Comunicación unidireccional. Los contadores envían datos al servidor, pero no existe un canal para enviar configuraciones, consultas y comandos desde el servidor hacia los equipos. Esta limitación impide realizar diagnósticos remotos y ejecutar acciones correctivas sin presencia física.
- Dependencia de proveedores GPRS tercerizados. La dependencia de servicios contratados genera costos recurrentes y limita el control sobre la calidad y disponibilidad de la conectividad.
- Imposibilidad de actualización remota. Cualquier modificación de parámetros o ajustes de operación requiere intervención en el sitio. Esto incrementa tiempos de mantenimiento, costos logísticos y complica la aplicación rápida de mejoras.

- Implementar mecanismos de encolado y reintento que eviten duplicaciones y pérdidas de información.
- Habilitar la ejecución remota de comandos y la actualización de parámetros desde un servidor central, con confirmación explícita del estado del equipo.
- Disminuir la dependencia de enlaces tercerizados mediante una arquitectura configurable.
- Permitir la modificación remota de parámetros operativos y la carga de ajustes sin desplazamientos.
- Incorporar telemetría de estado (batería, temperatura y códigos de error) para mantenimiento preventivo.
- Validar la viabilidad técnica y la robustez operativa del sistema

1.3. Estado del arte y propuesta de valor

En el mercado existen soluciones comerciales [7], [8], que ofrecen gestión remota y comunicación bidireccional para dispositivos de campo. Dichas soluciones suelen incluir plataformas propietarias con soporte técnico, servicios administrados y herramientas de análisis avanzadas. Estas alternativas, sin embargo, presentan costos elevados de adquisición y mantenimiento, así como dependencias tecnológicas que restringen la posibilidad de adaptación a condiciones locales específicas.

En el ámbito académico y técnico también se han documentado diversas experiencias orientadas a la gestión remota de infraestructuras distribuidas mediante protocolos de mensajería ligera como MQTT o tecnologías de comunicación celular [9], [10]. Estos trabajos resaltan la eficiencia de los modelos de publicación y suscripción, especialmente en entornos con restricciones de conectividad, pero en muchos casos se centran en aplicaciones industriales que difieren de las condiciones propias de las rutas nacionales [11].

Además, se han desarrollado plataformas de monitoreo basadas en API REST y bases de datos relacionales, que permiten la integración con interfaces web para la visualización y control [12]. Estas experiencias muestran la tendencia hacia arquitecturas abiertas y modulares, aunque su adopción práctica suele estar ligada a contextos con mayor disponibilidad de infraestructura y recursos.

En síntesis, el estado del arte revela que existen soluciones maduras para la gestión remota y la comunicación bidireccional de dispositivos, pero que dichas alternativas presentan limitaciones en términos de costo, flexibilidad o adecuación a escenarios con conectividad intermitente. Esta situación abre una oportunidad para explorar enfoques adaptados a las necesidades específicas del entorno vial argentino.

En conclusión, si bien el estado del arte muestra un abanico de soluciones maduras orientadas a la gestión remota y a la comunicación bidireccional, la mayoría de ellas se ve condicionada por altos costos, rigidez tecnológica o requerimientos de infraestructura poco compatibles con entornos de conectividad limitada. Estas restricciones ponen en evidencia un vacío que habilita la exploración de alternativas adaptadas a las particularidades del ámbito vial en Argentina.

1.3. Estado del arte y propuesta de valor 3

- Falta de telemetría y diagnóstico preventivo. No se dispone de métricas sistemáticas sobre el estado operativo de los equipos (batería, temperatura, errores de hardware o comunicación).
- Riesgo de pérdida de datos ante conectividad intermitente. La ausencia de mecanismos de encolamiento persistente y de políticas claras de reenvío eleva la probabilidad de pérdida o duplicación de eventos cuando la red es inestable.

Estas deficiencias afectan la calidad del servicio de monitoreo, reducen la eficiencia operativa y constituyen los requisitos funcionales que orientan el diseño del prototipo.

1.3. Estado del arte y propuesta de valor

En el mercado existen soluciones comerciales [9] que ofrecen gestión remota y comunicación bidireccional para dispositivos de campo. Dichas soluciones suelen incluir plataformas propietarias con soporte, servicio administrado y herramientas de análisis. No obstante, su adopción conlleva costos elevados y, en muchos casos, dependencias tecnológicas que dificultan la adaptación a especificidades locales.

La propuesta de este trabajo adopta un enfoque alternativo: emplear tecnologías de código abierto y protocolos estandarizados, especialmente MQTT, para construir una arquitectura flexible, escalable y de bajo costo. La separación entre el transporte de mensajes (broker MQTT) y los servicios de aplicación (API REST, base de datos y frontend) facilita la interoperabilidad y permite operar con brokers locales o externos según requieran las políticas institucionales.

Adicionalmente, la solución incorpora mecanismos diseñados para condiciones operativas típicas de rutas nacionales: operación con conectividad intermitente, envío diferido con políticas FIFO, control de reintentos para evitar duplicados y autenticación de credenciales en el broker MQTT. Este conjunto de medidas asegura integridad y trazabilidad de los eventos sin depender exclusivamente de servicios propietarios.

El valor diferencial reside en ofrecer una alternativa económicamente viable que preserve funcionalidad y escalabilidad, y que a su vez permita a la institución mantener control operativo sobre la infraestructura. La propuesta facilita la migración progresiva desde soluciones tercerizadas hacia una plataforma propia o semiautónoma, que reduce costos recurrentes y mejora la capacidad de respuesta ante incidentes.

1.4. Alcance

El trabajo desarrolla un prototipo funcional destinado a validar las hipótesis técnicas y operativas. El alcance comprende los siguientes objetivos y límites:

- Rediseño de comunicaciones: implementación de un modelo bidireccional y seguro entre los dispositivos de conteo y el servidor central, basado en MQTT sobre GPRS y con autenticación por credenciales.

1.4. Alcance

El trabajo desarrolla un prototipo funcional destinado a validar las hipótesis técnicas y operativas. El alcance comprende los siguientes objetivos y límites:

- Rediseño de comunicaciones: implementación de un modelo bidireccional y seguro entre los dispositivos de conteo y el servidor central, basado en MQTT sobre GPRS y con autenticación por credenciales.
- Gestión de mensajes en el dispositivo: encolamiento en memoria RAM con política FIFO, control de reintentos ante fallos de conexión y lógica de descarte cuando la cola alcance su límite definido.
- Backend y persistencia: desarrollo de una API REST que se suscriba al broker MQTT, procese los eventos y almacene los registros en una base de datos relacional para consulta histórica.
- Interfaz web básica: panel de visualización en tiempo real de eventos de tránsito y módulo para envío de comandos y verificación de estado de los dispositivos.
- Funciones de telemetría: reporte de nivel de batería, temperatura y códigos de error para facilitar el diagnóstico remoto.

El alcance técnico incluye la adaptación de un contador existente para comunicarse bidireccionalmente por GPRS, que emplee MQTT como protocolo de mensajería ligera y confiable. Asimismo, comprende el desarrollo de un backend que reciba, procese y persista los eventos en una base de datos relacional. Además, una interfaz web básica que permita la visualización en tiempo real y el envío de comandos hacia los dispositivos. Se priorizan las funciones que validen la cadena completa: adquisición de eventos desde el sistema de detección, encolamiento local con política FIFO, reenvío seguro cuando haya conectividad, recepción y ejecución de comandos y verificación del estado del dispositivo tras cada acción.

La figura 1.1 muestra el diagrama en bloques del sistema. El dispositivo de conteo envía datos por GPRS a un broker MQTT, el servidor central los procesa y almacena en una base de datos, mientras que una API REST permite su consulta y la ejecución de comandos desde una interfaz web.

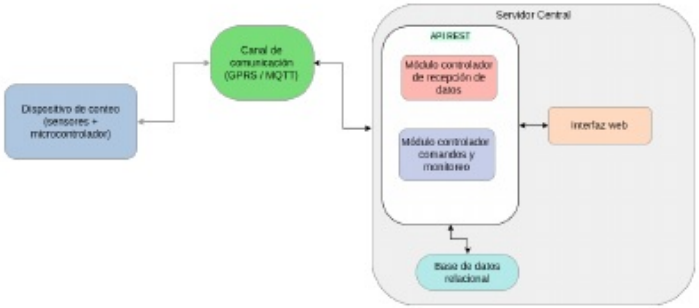

FIGURA 1.1. Diagrama en bloques del sistema.

4 Capítulo 1. Introducción general

- Gestión de mensajes en el dispositivo: encolamiento en memoria RAM con política FIFO, control de reintentos ante fallos de conexión y lógica de descarte cuando la cola alcance su límite definido.
- Backend y persistencia: desarrollo de una API REST que se suscriba al broker MQTT, procese los eventos y almacene los registros en una base de datos relacional para consulta histórica.
- Interfaz web básica: panel de visualización en tiempo real de eventos de tránsito y módulo para envío de comandos y verificación de estado de los dispositivos.
- Funciones de telemetría: reporte de nivel de batería, temperatura y códigos de error para facilitar el diagnóstico remoto.

La figura 1.1 muestra el diagrama en bloques del sistema. El dispositivo de conteo envía datos por GPRS a un broker MQTT, el servidor central los procesa y almacena en una base de datos, mientras que una API REST permite su consulta y la ejecución de comandos desde una interfaz web.

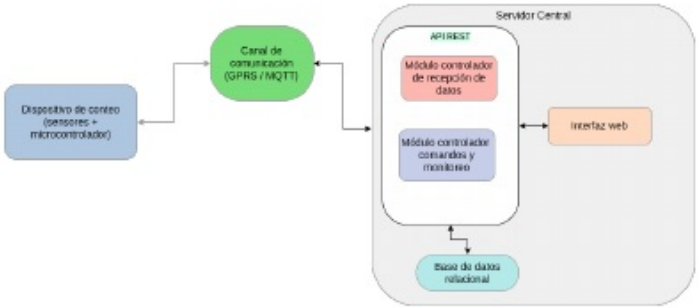

FIGURA 1.1. Diagrama en bloques del sistema.

Capítulo 2

Introducción Específica

En este capítulo se detallan los aspectos técnicos específicos que constituyen la base del trabajo. En primer lugar, se describen los protocolos de comunicación que se emplean y la función de cada uno de ellos en la transmisión de datos. Luego, se presentan los componentes de hardware que utiliza el prototipo. A continuación, se analizan las tecnologías de software que integran la solución, la herramienta de control de versiones adoptada para el desarrollo colaborativo y la gestión del código fuente.

2.1. Protocolos de comunicación

El diseño de un sistema de conteo de tránsito con comunicación bidireccional demanda la incorporación de protocolos que aseguren confiabilidad y eficiencia en el intercambio de datos. En este trabajo se integran tres tecnologías principales: RS-232, MQTT y API REST, cada una con un rol específico.

- RS-232: es un estándar de comunicación serial utilizado tradicionalmente en sistemas embebidos. Permite la transmisión de datos punto a punto entre el contador de tránsito y el microcontrolador ESP32-C3 [13]. Su simplicidad lo hace adecuado para distancias cortas y ambientes donde la interferencia es controlada. Aunque se trata de un protocolo clásico, su adopción garantiza compatibilidad con dispositivos que aún dependen de interfaces seriales [14], [15].
- MQTT: este protocolo de mensajería ligera se emplea tanto para la transmisión de eventos de tránsito desde los dispositivos hacia el servidor central como para la recepción de comandos en sentido inverso. MQTT opera sobre TCP/IP [16] y emplea un modelo de publicación/suscripción a través de un broker, lo que facilita la escalabilidad y la integración de múltiples dispositivos. Además, permite implementar mecanismos de calidad de servicio (QoS) que reducen la probabilidad de pérdida de datos en condiciones de conectividad inestable, lo que resulta crucial para el escenario de rutas nacionales [17].
- API REST: constituye la interfaz de comunicación entre el backend y los clientes web. Su inclusión en el proyecto posibilita la consulta y el envío de información de manera estructurada, mediante operaciones estándar (GET, POST, PUT, DELETE). La API REST asegura la interoperabilidad con diferentes plataformas y brinda flexibilidad para desarrollar aplicaciones adicionales que utilicen los datos recolectados [18], [19] .

5

Capítulo 2

Introducción Específica

En este capítulo se detallan los aspectos técnicos específicos que constituyen la base del proyecto. En primer lugar, se describen los protocolos de comunicación empleados y su función en la transmisión de datos. Luego, se presentan los componentes de hardware utilizados en la implementación del prototipo. A continuación, se analizan las tecnologías de software que integran la solución, la herramienta de control de versiones adoptada para el desarrollo colaborativo y la gestión del código fuente.

2.1. Protocolos de comunicación

El diseño de un sistema de conteo de tránsito con comunicación bidireccional demanda la incorporación de protocolos que aseguren confiabilidad y eficiencia en el intercambio de datos. En este proyecto se integran tres tecnologías principales: RS-232, MQTT y API REST, cada una con un rol específico.

- RS-232: es un estándar de comunicación serial utilizado tradicionalmente en sistemas embebidos. Permite la transmisión de datos punto a punto entre el contador de tránsito y el microcontrolador ESP32-C3. Su simplicidad lo hace adecuado para distancias cortas y ambientes donde la interferencia es controlada. Aunque se trata de un protocolo clásico, su adopción garantiza compatibilidad con dispositivos que aún dependen de interfaces seriales [10] [11].
- MQTT: este protocolo de mensajería ligera se emplea tanto para la transmisión de eventos de tránsito desde los dispositivos hacia el servidor central como para la recepción de comandos en sentido inverso. MQTT opera sobre TCP/IP y emplea un modelo de publicación/suscripción a través de un broker, lo que facilita la escalabilidad y la integración de múltiples dispositivos. Además, permite implementar mecanismos de calidad de servicio (QoS) que reducen la probabilidad de pérdida de datos en condiciones de conectividad inestable, lo cual resulta crucial para el escenario de rutas nacionales [12].
- API REST: constituye la interfaz de comunicación entre el backend y los clientes web. Su inclusión en el proyecto posibilita la consulta y el envío de información de manera estructurada, mediante operaciones estándar (GET, POST, PUT, DELETE). La API REST asegura la interoperabilidad con diferentes plataformas y brinda flexibilidad para desarrollar aplicaciones adicionales que utilicen los datos recolectados [13] [14] .

La combinación de estos protocolos permite cubrir distintos niveles de la arquitectura: comunicación local (RS-232), comunicación de dispositivos con el servidor (MQTT) y comunicación entre el servidor y las aplicaciones de usuario (API REST). De esta forma, se garantiza un flujo de datos seguro, confiable y bidireccional.

2.2. Componentes de hardware utilizados

El prototipo se implementa con un conjunto de componentes de hardware seleccionados por su disponibilidad, costo y adecuación al entorno de operación.

- Contador de tránsito: dispositivo de campo que se encarga de detectar el paso de vehículos, clasificarlos y generar eventos para su posterior transmisión. También, tiene la capacidad de recibir comandos.
- ESP32-C3: microcontrolador de bajo consumo que actúa como unidad de procesamiento y comunicación. Integra conectividad y capacidad de comunicación serial, lo que lo hace adecuado para la interacción con el contador y con el módulo GPRS [20].
- Módulo GPRS SIM800L: permite la conexión de los dispositivos a la red celular, asegurando la transmisión de datos al servidor central mediante MQTT. Se eligió por su bajo costo, disponibilidad en el mercado y facilidad de integración con el ESP32-C3.
- Servidor central: responsable de recibir los datos transmitidos, almacenarlos en una base de datos y responder a las solicitudes de los usuarios.
- PC con navegador web: constituye el medio de interacción del usuario final con el sistema, a través de la interfaz desarrollada.

La integración de estos componentes asegura que el sistema pueda operar en condiciones reales y que cumpla con los requisitos.

2.3. Tecnologías de software aplicadas

El desarrollo del prototipo se sustenta en tecnologías de software abiertas y estandarizadas, seleccionadas por su solidez, disponibilidad y capacidad de integración.

- MQTT con Eclipse Mosquitto. La mensajería ligera y bidireccional se implementa mediante el protocolo MQTT, operando sobre el broker Eclipse Mosquitto [21]. Este software es ampliamente utilizado en entornos de Internet de las Cosas (IoT).
- API REST con Node.js y Express. El backend se desarrolla en Node.js [22], un entorno de ejecución orientado a aplicaciones de red no bloqueantes, y se estructura con Express [23], un framework ligero que facilita la creación de servicios RESTful.
- Base de datos relacional MySQL. La persistencia de los datos se implementa en MySQL [24], un sistema gestor de bases de datos relacionales ampliamente adoptado. En ella se almacenan de manera estructurada tanto los eventos de tránsito como los estados reportados por los equipos.

6 Capítulo 2. Introducción Específica

La combinación de estos protocolos permite cubrir distintos niveles de la arquitectura: comunicación local (RS-232), comunicación de dispositivos con el servidor (MQTT) y comunicación entre el servidor y las aplicaciones de usuario (API REST). De esta forma se garantiza un flujo de datos seguro, confiable y bidireccional.

2.2. Componentes de hardware utilizados

El prototipo se implementa con un conjunto de componentes de hardware seleccionados por su disponibilidad, costo y adecuación al entorno de operación.

- Contador de tránsito: dispositivo de campo encargado de detectar el paso de vehículos, clasificarlos y generar eventos que serán transmitidos. También, tiene la capacidad de recibir comandos.
- ESP32-C3: microcontrolador de bajo consumo que actúa como unidad de procesamiento y comunicación. Integra conectividad y capacidad de comunicación serial, lo que lo hace adecuado para la interacción con el contador y con el módulo GPRS [15].
- Módulo GPRS SIM800L: permite la conexión de los dispositivos a la red celular, asegurando la transmisión de datos al servidor central mediante MQTT. Se eligió por su bajo costo, disponibilidad en el mercado y facilidad de integración con el ESP32-C3.
- Servidor central: responsable de recibir los datos transmitidos, almacenarlos en una base de datos y responder a las solicitudes de los usuarios.
- PC con navegador web: constituye el medio de interacción del usuario final con el sistema, a través de la interfaz desarrollada.

La integración de estos componentes asegura que el sistema pueda operar en condiciones reales y que cumpla con los requisitos.

2.3. Tecnologías de software aplicadas

El desarrollo del prototipo se sustenta en tecnologías de software abiertas y estandarizadas, seleccionadas por su solidez, disponibilidad y capacidad de integración.

- MQTT con Eclipse Mosquitto. La mensajería ligera y bidireccional se implementa mediante el protocolo MQTT, operando sobre el broker Eclipse Mosquitto [16]. Este software es ampliamente utilizado en entornos de Internet de las Cosas (IoT).
- API REST con Node.js y Express . El backend se desarrolla en Node.js [17], un entorno de ejecución orientado a aplicaciones de red no bloqueantes, y se estructura con Express [18], un framework ligero que facilita la creación de servicios RESTful.
- Base de datos relacional MySQL. La persistencia de los datos se implementa en MySQL [19], un sistema gestor de bases de datos relacionales ampliamente adoptado. En ella se almacenan de manera estructurada tanto los eventos de tránsito como los estados reportados por los equipos.

- Interfaz web con Ionic y Angular . Para la interacción con el usuario final se diseñó una aplicación accesible desde cualquier navegador, construida con Ionic [25] y Angular [26]. La primera aporta componentes visuales responsivos que aseguran usabilidad tanto en entornos de escritorio como en dispositivos móviles.
- Firmware para ESP32-C3 con ESP-IDF. El microcontrolador ejecuta un firmware desarrollado en C/C++, el cual utiliza el framework oficial de ESP-IDF [27], el framework oficial de Espressif. Este entorno proporciona bibliotecas optimizadas. El firmware controla la captura de datos desde el contador mediante RS-232, gestiona la comunicación con el módulo SIM800L mediante comandos AT y establece la conexión con el broker Mosquitto a través de MQTT.

2.4. Software de control de versiones

Para la gestión del código fuente se empleó GitHub [28], una plataforma que se basa en el sistema de control de versiones Git. Esta herramienta permitió almacenar el repositorio central de manera segura, registrar el historial de cambios y garantizar la trazabilidad de cada modificación realizada durante la etapa de desarrollo del prototipo.

El repositorio incluye el firmware del ESP32-C3, el backend en Node.js con Express y la interfaz web desarrollada con Ionic y Angular. Esto facilita mantener el código organizado, con la posibilidad de crear ramas independientes para implementar nuevas funciones y, posteriormente, integrarlas al código principal mediante pull requests, lo que permite revisar y validar los cambios antes de su incorporación definitiva.

Asimismo, se emplean issues para documentar incidencias, planificar tareas y realizar el seguimiento de los avances. Esta práctica favorece la organización del trabajo y permite mantener un registro claro de los problemas detectados y las decisiones adoptadas.

2.4. Software de control de versiones 7

- Interfaz web con Ionic y Angular . Para la interacción con el usuario final se diseñó una aplicación accesible desde cualquier navegador, construida con Ionic [20] y Angular [21]. Ionic aporta componentes visuales responsivos que aseguran usabilidad tanto en entornos de escritorio como en dispositivos móviles.
- Firmware para ESP32-C3 con ESP-IDF. El microcontrolador ejecuta un firmware desarrollado en C/C++ sobre ESP-IDF [22], el framework oficial de Espressif. Este entorno proporciona librerías optimizada. El firmware controla la captura de datos desde el contador mediante RS-232, gestiona la comunicación con el módulo SIM800L mediante comandos AT y establece la conexión con el broker Mosquitto a través de MQTT.

2.4. Software de control de versiones

Para la gestión del código fuente se empleó GitHub [23], una plataforma basada en el sistema de control de versiones Git. Esta herramienta permitió almacenar el repositorio central de manera segura, registrar el historial de cambios y garantizar la trazabilidad de cada modificación realizada durante el desarrollo del prototipo.

El repositorio incluye el firmware del ESP32-C3, el backend en Node.js con Express y la interfaz web desarrollada con Ionic y Angular. Esto facilita mantener el código organizado, con la posibilidad de crear ramas independientes para implementar nuevas funciones y, posteriormente, integrarlas al código principal mediante pull requests, lo que permite revisar y validar los cambios antes de su incorporación definitiva.

Asimismo, se emplean issues para documentar incidencias, planificar tareas y realizar el seguimiento de los avances. Esta práctica favorece la organización del trabajo y permite mantener un registro claro de los problemas detectados y las decisiones adoptadas.

Capítulo 3

Diseño e implementación

En este capítulo se describe la arquitectura global del prototipo, se detalla cada módulo hardware y software que lo compone, y se documentan las decisiones de implementación, los criterios de diseño y las pruebas preliminares realizadas. Se explican los flujos de datos entre el dispositivo de campo, el broker MQTT, el backend (API REST), la interfaz web, se resumen las consideraciones para el despliegue y el monitoreo post-implantación.

3.1. Arquitectura del sistema

La arquitectura propuesta separa de forma explícita el dispositivo de campo (contador + ESP32-C3 + SIM800L), el transporte de mensajes (broker MQTT) y los servicios de aplicación (API REST, persistencia y frontend). Esta separación facilita la interoperabilidad y permite desplegar la solución de forma local, remota o híbrida según las políticas institucionales

3.1.1. Flujo de datos

- Detección: el contador detecta un paso y envía una trama por RS-232 al ESP32-C3.
- Preprocesado en nodo: el firmware valida la trama, añade sello temporal y metadatos, y encola el evento en memoria (FIFO).
- Transmisión: cuando la conexión GPRS está disponible, el nodo publica el evento en el tópico MQTT `devices/id/events`.
- Ingesta y persistencia: el broker Mosquitto entrega el mensaje al suscriptor backend, el servicio valida el payload y persiste el registro en la base de datos MySQL.
- Visualización/Control: la interfaz web consulta la API REST para datos históricos y recibe notificaciones en tiempo real.
- Emisión de comandos (desde UI): el operador genera un comando en la interfaz, la UI envía `POST /api/devices/id/comm` al backend, que crea un `cmd_id` único y publica en `devices/id/comm`.
- Recepción en nodo y entrega al contador: el ESP32-C3, suscrito a `devices/id/comm`, recibe el comando, valida `cmd_id` y lo envía al contador por RS-232, se aplica un timeout configurable por comando.

Bibliografía

[1] Juan Asiain et al. «LoRa-Based Traffic Flow Detection for Smart-Road». En: *Sensors* 21.2 (2021), pág. 338. DOI: 10.3390/s21020338. URL: https://www.mdpi.com/1424-8220/21/2/338.

[2] Jan Micko et al. «Review of IoT Sensor Systems Used for Monitoring the Road Infrastructure». En: *Sensors* 23.9 (2023), pág. 4469. DOI: 10.3390/s23094469. URL: https://www.mdpi.com/1424-8220/23/9/4469.

[3] Gianluca Peruzzi et al. «Combining LoRaWAN and NB-IoT for Edge-to-Cloud Low-Power Connectivity». En: *Applied Sciences* 12.3 (2022), pág. 1497. DOI: 10.3390/app12031497. URL: https://www.mdpi.com/2076-3417/12/3/1497.

[4] FHWA. *Traffic Monitoring Guide*. Inf. téc. Federal Highway Administration, 2022. URL: https://www.fhwa.dot.gov/policyinformation/tmguide/2022_TMG_Final_Report.pdf.

[5] FHWA. *Traffic Detector Handbook, 3rd Edition*. Inf. téc. Federal Highway Administration, 2006. URL: https://www.fhwa.dot.gov/publications/research/operations/its/06108/06108.pdf.

[6] Miovision. *TrafficLink / Managed Connectivity*. https://miovision.com/trafficlink/. Soluciones comerciales. 2023.

[7] Sensys Networks. *Documentación técnica y productos*. https://sensysnetworks.com/. 2023.

[8] MetroCount. *Contadores y guías técnicas*. https://www.metrocount.com/. 2023.

[9] *Exemys Managed Connectivity*. Accedido: 16-Sep-2025. 2025. URL: https://www.exemys.com/site/index.shtml.

[10] Analog Devices. *Fundamentals of RS-232 Serial Communications*. https://www.analog.com/en/resources/technical-articles/fundamentals-of-rs232-serial-communications.html. Accedido: 11-Sep-2025. 2020.

[11] Texas Instruments. *RS-232 Glossary and Selection Guide*. https://www.ti.com/lit/SLLA607. Accedido: 11-Sep-2025. 2016.

[12] OASIS y MQTT.org. *MQTT Version 5.0 Specification*. https://mqtt.org/mqtt-specification/. Accedido: 11-Sep-2025. 2019.

[13] IBM. *What Is a REST API (RESTful API)?* https://www.ibm.com/think/topics/rest-apis. Accedido: 11-Sep-2025. 2021.

[14] Microsoft Azure Architecture Center. *Web API Design Best Practices*. https://learn.microsoft.com/en-us/azure/architecture/best-practices/api-design. Accedido: 11-Sep-2025. 2022.

[15] Espressif Systems. *ESP-IDF Programming Guide for ESP32-C3*. https://docs.espressif.com/projects/esp-idf/en/stable/esp32c3/index.html. Consultado el 11 de septiembre de 2025. 2024.

[16] Eclipse Foundation. *Eclipse Mosquitto: An Open Source MQTT Broker*. https://mosquitto.org/. Consultado el 11 de septiembre de 2025. 2025.

[17] OpenJS Foundation. *Node.js JavaScript Runtime*. https://nodejs.org/. Consultado el 11 de septiembre de 2025. 2025.

- Ejecución y ack: el contador ejecuta la orden y responde por RS-232, el firmware publica el ack/resultado en `devices/id/status` con `cmd_id` y `status` (`ok`, `failed`, `timeout`, `value`).
- Actualización en backend y UI: el suscriptor MQTT del backend recibe el ack, actualiza la tabla `comm` (campo `status`, `ack_ts`) y notifica a la UI para que el operador vea el resultado.

3.1.2. Descripción ampliada de bloques y responsabilidades

- Bloque Dispositivo de campo: el nodo de campo integra el contador existente (salida RS-232), un microcontrolador ESP32-C3 y un módem GPRS SIM800L. El firmware, desarrollado sobre `ESP-IDF`, realiza las siguientes funciones: lectura continua de la trama serial, parsing tolerante a ruido, preprocesado (validación, normalización de campos y asignación de sello temporal UTC), encolamiento FIFO de eventos, gestión de reintentos y publicación MQTT cuando hay conectividad. Además, el nodo se suscribe a los tópicos de comandos y publica telemetría y acks. En el nodo se implementa persistencia mínima (registro de comandos pendientes y últimas N tramas) para recuperación tras reinicio.
- Bloque Transporte (broker MQTT): el broker actúa como bus de mensajes desacoplado. Se recomienda emplear Eclipse Mosquitto en la etapa inicial y evaluar brokers gestionados para despliegues a mayor escala. El broker gestiona autenticación por credenciales, control de tópicos y cifrado. Se emplean tópicos jerárquicos por dispositivo para facilitar filtrado y autorización:
  - `devices/{id}/events`
  - `devices/{id}/comm`
  - `devices/{id}/status`
- Bloque Servidor central: el servidor central reúne dos responsabilidades principales:
  - Componente suscriptor MQTT que valida, transforma y enruta mensajes hacia la lógica de negocio y la persistencia.
  - API REST que expone servicios de consulta, gestión y emisión de comandos. Esta separación permite que consumidores adicionales (por ejemplo, módulos analíticos) se suscriban al broker sin impactar la disponibilidad de la API. La persistencia se implementa en MySQL con un esquema relacional que soporta consultas por rango temporal, índices para rendimiento y auditoría de comandos.
- Bloque Cliente/Visualización: la interfaz web consume la API REST para consultas históricas y para recibir eventos en tiempo real. Se eligió Ionic + Angular por su compatibilidad con entornos de escritorio y móviles y por facilitar un despliegue unificado. Las funciones principales del cliente son: visualización de eventos en tiempo real, consulta histórica filtrada, envío de comandos remotos con seguimiento de estado y panel de telemetría para mantenimiento preventivo.

[18] Express.js Foundation. *Express: Fast, unopinionated, minimalist web framework for Node.js*. https://expressjs.com/. Consultado el 11 de septiembre de 2025. 2025.
[19] Oracle Corporation. *MySQL: The World's Most Popular Open Source Database*. https://www.mysql.com/. Consultado el 11 de septiembre de 2025. 2025.
[20] Ionic Team. *Ionic Framework - Cross-platform mobile apps with web technologies*. Último acceso: 19 de septiembre de 2025. 2025. URL: https://ionicframework.com/.
[21] Angular Team. *Angular - One framework. Mobile desktop*. Último acceso: 19 de septiembre de 2025. 2025. URL: https://angular.io/.
[22] Espressif Systems. *ESP-IDF Programming Guide*. Último acceso: 19 de septiembre de 2025. 2025. URL: https://docs.espressif.com/projects/esp-idf/en/latest/esp32/.
[23] GitHub, Inc. *GitHub: Where the world builds software*. Último acceso: 19 de septiembre de 2025. 2025. URL: https://github.com/.