TECHNICAL REPORT

ISO/TR 23255

First edition
2022-04

Intelligent transport systems — Architecture — Applicability of data distribution technologies within ITS

Systèmes de transport intelligents — Architecture — Applicabilité des technologies de distribution des données dans les ITS

Reference number
ISO/TR 23255:2022(E)

ISO copyright office
CP 401 • Ch. de Blandonnet 8
CH-1214 Vernier, Geneva
Phone: +41 22 749 01 11
Email: copyright@iso.org
Website: www.iso.org

Published in Switzerland

Contents

Page

Foreword

ISO (the International Organization for Standardization) is a worldwide federation of national standards bodies (ISO member bodies). The work of preparing International Standards is normally carried out through ISO technical committees. Each member body interested in a subject for which a technical committee has been established has the right to be represented on that committee. International organizations, governmental and non-governmental, in liaison with ISO, also take part in the work. ISO collaborates closely with the International Electrotechnical Commission (IEC) on all matters of electrotechnical standardization.

The procedures used to develop this document and those intended for its further maintenance are described in the ISO/IEC Directives, Part 1. In particular, the different approval criteria needed for the different types of ISO documents should be noted. This document was drafted in accordance with the editorial rules of the ISO/IEC Directives, Part 2 (see www.iso.org/directives).

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO shall not be held responsible for identifying any or all such patent rights. Details of any patent rights identified during the development of the document will be in the Introduction and/or on the ISO list of patent declarations received (see www.iso.org/patents).

Any trade name used in this document is information given for the convenience of users and does not constitute an endorsement.

For an explanation of the voluntary nature of standards, the meaning of ISO specific terms and expressions related to conformity assessment, as well as information about ISO's adherence to the World Trade Organization (WTO) principles in the Technical Barriers to Trade (TBT), see www.iso.org/iso/foreword.html.

This document was prepared by Technical Committee ISO/TC 204, *Intelligent transport systems*.

Any feedback or questions on this document should be directed to the user's national standards body. A complete listing of these bodies can be found at www.iso.org/members.html.

Introduction

Since the early 2000s, various study, design and prototype efforts have been undertaken to explore the potential use of communications in the vehicular environment. The first of these to demonstrate at real scale was the Vehicle Infrastructure Initiative (VII), which demonstrated short range wireless-based probe data generation and traveller advisory message delivery. This suggested the viability of initial vehicle-to-vehicle and vehicle-to-infrastructure communications.

Subsequent projects worked to more formally define the "glue" components necessary to enable widespread deployment. Several of these projects concluded that a publish-subscribe data distribution paradigm was a necessary component of any connected vehicle implementation of significant scale. These conclusions and much of the supporting work eventually found its way into ITS architectures. Much of this material is currently included in the Architecture Reference for Cooperative and Intelligent Transportation (ARC-IT)[6].

More recent pilot projects and deployments in both the United States and Europe have included publish-subscribe technologies, but no independent, objective analyses of the advantages and disadvantages of using specific protocols to facilitate data exchange within ITS are available. This document describes such an analysis.

Intelligent transport systems — Architecture — Applicability of data distribution technologies within ITS

1 Scope

A variety of general-purpose data distribution technologies have emerged within the Information and Communications Technologies (ICT) industry. These technologies generally provide services at the Open System Interconnect (OSI) session, presentation and application layers (i.e. layers 5-7). Within Intelligent Transport Systems (ITS), these layers roughly correspond to the facilities layer of the ITS station (ITS-S) reference architecture, as defined within ISO 21217.

This document investigates the applicability of these data distribution technologies within the ITS environment.

2 Normative references

The following documents are referred to in the text in such a way that some or all of their content constitutes requirements of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

ISO/TS 14812, *Intelligent Transport Systems — Vocabulary*

3 Terms and definitions

For the purposes of this document, the terms and definitions given in ISO/TS 14812 and the following apply.

ISO and IEC maintain terminology databases for use in standardization at the following addresses:

— ISO Online browsing platform: available at https://www.iso.org/obp

— IEC Electropedia: available at https://www.electropedia.org/

3.1
data distribution functionality
DDF
facilities layer (OSI layers 5, 6, and 7) functionality comprised of a set of data distribution services that enables distribution of data throughout a communication network controlled by a set of policies, regulations and rules

Note 1 to entry: Each distinct data distribution technology has its own unique data distribution functionality.

3.2
data distribution service
DDS
element of a set of services that implements a data distribution functionality in a communication network

EXAMPLE 1 Publish: the provision of data from one entity to another, where the receiving entity has previously registered to receive such data from the entity providing the data.

EXAMPLE 2 Subscribe: mechanism by which one entity registers for the reception of particular data from another entity.

EXAMPLE 3 Discovery: mechanism by which entities implementing DDF learn necessary particulars about how to communicate with one another (e.g. lower layer network address, ports, etc.).

Note 1 to entry: This is a specific protocol standardized by the OMG.

4 Abbreviated terms

AES	Advanced Encryption Standard
AMQP	advanced message queuing protocol
API	application programming interface
ARC-IT	architecture reference for cooperative and intelligent transportation
ASN.1	abstract syntax notion one
AUTOSAR	automotive open system architecture
C-ITS	cooperative ITS
C2C	centre-to-centre
C2F	centre-to-field
C2P	centre-to-personal station
C2V	centre-to-vehicle
C2X	centre-to-anything
CoAP	constrained application protocol
ConOps	concept of operations
CSSDDS	commercial source software DDS
CSV	comma separated value
DSRC	dedicated short range communications
FEP-SDK	functional engineering platform software development kit
FIPS	Federal Information Processing Standard
HARTS	harmonized architecture reference for technical standards
HTTP	hypertext transfer protocol
IANA	internet assigned numbers authority
ICD	interface control document
ICT	information and communications technology
IDL	interface definition language
IEC	International Electrotechnical Commission
IEEE	Institute of Electrical and Electronic Engineers

IoT	internet of things
IP	internet protocol
ISO	Organization of International Standardization
ITS	intelligent transport systems
ITS-S	ITS station
ITS-SU	ITS station unit
JMS	Java Message Service
JSON	Javascript Object Notation
MQTT	message queuing telemetry transport
NTCIP	National Transportation Communications for ITS Protocol
OMG	object management group
OASIS	Organization for the Advancement of Structured Information Standards
OS	operating system
OSI	open system interconnect
OSS DDS	open source software DDS
PHP	hypertext preprocessor
REST	representational state transfer
RSS	really simple syndication
SNMP	simple network management protocol
SOAP	simple object access protocol
STOMP	simple text-oriented messaging protocol
TCP	transport control protocol
TLS	transport layer security
UDP	user datagram protocol
UML	Unified Modeling Language
V2I	vehicle-to-infrastructure (communications)
VII	vehicle infrastructure initiative
V2V	vehicle-to-vehicle (communications)
XML	Extensible Markup Language
XMPP	Extensible Messaging and Presence Protocol

5 Transitioning from traditional to cooperative thinking

5.1 General

5.1.1 Need for data exchanges

ITS is heavily dependent upon the exchange of various types of data between and among disparate types of physical entities. As described within the Architecture Reference for Cooperative and Intelligent Transportation (see Reference [6]), such entities include:

— centres (e.g. fixed-location facilities and cloud-based back-office services);

— field devices (e.g. devices along the roadside);

— vehicles;

— travellers (e.g. personal devices);

— support systems (typically fixed or back-office, that provide services enabling ITS, but do not directly provide ITS services).

The data that these systems exchange include:

— live elemental data (e.g. vehicle speed, location, signal timing information, device status, etc.);

— live aggregated data (e.g. average speeds, rain rates, etc.);

— status information (e.g. status of reversible flow lanes);

— (relatively) static data (e.g. map information);

— quasi-static information (e.g. road conditions, weather);

— exception reports (e.g. information on traffic incidents, realignment of lanes due to incidents or road work, etc.);

— control and configuration data (e.g. device control, software configuration);

— coordination data (e.g. exchanges to coordinate a response plan among centres);

— traffic regulations;

— software updates (e.g. for on-board applications);

— security material distribution including certificates and revocation lists.

Entities exchange data according to some well-characterized patterns. Certain kinds of entities typically exchange certain kinds of data, and some characteristics of those exchanges tend to be relatively consistent for similar kinds of data. For example:

— centres provide control and configuration data to field devices. These exchanges tend to be synchronous request-response-based exchanges and occur irregularly;

— centres exchange coordination data with one another. These exchanges vary in size and format and occur irregularly;

— field devices provide elemental and aggregated data to centres. These exchanges tend to be periodic and are often redistributed centre-to-centre;

— centres provide exceptional information to field, vehicle, and personal devices. As exceptions, these are irregular. The information is often geo-centric and needs to be disseminated to all entities within a defined area that can benefit from such information, such as the dissemination of traffic incidents to all vehicles upstream of the incident;

— centres provide information of wide utility (e.g. traffic regulations) to all vehicles in a geographic area. These exchanges vary in size, can be initiated by either party depending on the communications regime and are typically motivated by the geo-location of the vehicle or personal device;

— vehicles provide live and aggregated data to field devices and centres. These exchanges are typically periodic and can be pseudonymized.

Additionally, some of the information exchanged can be useful for supporting ITS services other than the ITS service for which the data was originally intended. Some of the information acquired and exchanged between entities when implementing a specific ITS service also has potential value outside ITS.

The challenges inherent in attempting to efficiently share information are two-fold. First, information has value; often those involved in the acquisition will attempt to retain ownership of and control over such information and to minimize privacy issues. Second, assuming the information can be made available from the source, the technical challenge arises of getting the information to the right places at the right times while implementing the established privacy policies. This document addresses the later issue, The former issue is addressed by regulatory entities and other stakeholders in the ITS community.

There are a variety of technical and institutional challenges related to successfully sharing data in a timely and secure manner. These challenges include:

— acquiring the data (e.g. through sensors);

— defining ownership and access rights for the data;

— securing the data (e.g. authentication, authorization, confidentiality, integrity, availability, etc.);

— achieving adequate market penetration of lower-layer communication technologies;

— agreeing on the upper-layer protocols for exchanging the data over the communication technologies;

— standardizing the definition of data for use in various contexts;

— defining performance criteria for different uses of the data;

— maintaining the interface over the life cycle of the involved physical objects. Operational lifetimes for ITS devices vary radically: field devices often have lifetimes of 15-20 years, vehicles closer to 10 (although often much longer) and smartphones can be as short as 18 months.

5.1.2 Data distribution functionality

A data distribution functionality (DDF) is implemented as a set of facilities layer (OSI layers 5, 6, and 7) services that enables distribution of data throughout a communication network controlled by a set of policies, regulations and rules. Through a standardized application programming interface (API), application processes can request information from (subscribe) and offer information to (publish) the communication network to which they are attached without needing to know anything about the details of how the information transfers take place. Using metadata and service configuration requests, a variety of policies, rules and regulations can be implemented. When using DDF, application processes no longer need to directly create and consume messages with other application processes to affect information exchange. Instead, application processes publish data they agree to share (and receive data they are interested in) through an API without having to know anything about the final destination(s) (or source) or having to conform to a particular message format for each end entity.

NOTE The Object Management Group (OMG) has created a set of standards for data distribution functionality (see 3.1) called the Data Distribution Service (DDS). This document uses the term “OMG DDS” to refer to OMG’s understanding of data distribution functionality and the term “DDF” to refer to a generic data distribution functionality.

5.2 Systems engineering process

5.2.1 Conceptualization

The systems engineering approach to designing any complex system is to work with the relevant stakeholders, including service providers and system integrators, to develop a "concept of operations", or ConOps. This involves describing in detail the service (the "why"), the actors participating in the service (the "who"), and the requirements on information to be generated and exchanged by entities engaged in the service (the "what").

Once agreement is reached on the ConOps, the implementers work together to develop a high-level design (i.e. an architecture) that defines the means by which the service will be implemented (the "how"), which (directly or indirectly) defines the details of how the information is encoded and transferred between physical objects. If the system is intended to support an open interface (i.e. so that competing manufacturers can interoperate), these design details need to be defined within open standards and developed with broad-based consensus.

5.2.2 System architecture

An architecture description of a specific system of interest can leverage existing work (e.g. reference architectures such as ARC-IT) to simplify the organization of content, provide a common language reference and assist in identifying implementation-relevant artifacts, in particular interfaces as well as the standards used to implement endpoints and interfaces. The reference architecture can illustrate where many information exchanges overlap or group together in patterns, which would suggest an opportunity for consolidation that is relevant to a DDF.

For example, if several information flows all have the same source and destination, then perhaps those information flows can share a DDF technology to provide some aspects of the information exchange. These patterns will generally become clear if a system architecture is illustrated.

The system architecture development process can suggest design paradigms for interfaces, where some interfaces are traditional 'mesh' interfaces (custom at each end), while others use a DDF to provide transport, publication and subscription management, with or without a hub/broker. More complex involvements can require hierarchies, which can be best noted if data dependencies are clearly shown.

5.2.3 System design

Open system design activities focus on developing interface control documents (ICDs), which specify the:

— rules for application processes needing to share data;

— data elements to be exchanged;

— messages that contain those elements;

— dialogues and patterns of message exchange, which suggest or require behaviours at end points;

— lower layer details (i.e. details of the network and transport layer and access layer of the ITS-S architecture).

Each DDF specification provides the messaging and dialogue components of the ICD. In many cases, other standards will define the other aspects of the ICD and the ICD itself becomes primarily a reference to a series of standards. Using DDF provides significant savings because it relies upon a single standard interface for exchanging any data rather than requiring specialized messages for each interface.

5.3 Traditional silos versus cooperative approaches

Once the architecture is developed, each interface is designed by its own group of experts to meet the defined needs. However, this division of effort tends to produce "silos" of thought that can often result in four major problems:

1) Competing protocol selection: Different silo efforts are likely to select different approaches to exchanging data. There are many off-the-shelf protocols that can be extended to support most ITS data exchange needs and some experts can wish to develop their own protocols to optimize performance in certain cases. While each decision can be reasonable in isolation, each protocol adopted by the ITS industry has costs associated with stakeholders learning the technology, implementers programming with the technology, testers verifying conformance to the technology, and maintenance issues with maintaining backwards compatibility, as well as memory and processing issues within devices that need to support multiple technologies. Ideally, the ITS community as a whole can attempt to identify a suite of preferred protocols that meet industry needs so that the variability in systems is minimized.

2) Competing data definitions: Different silo efforts are likely to produce different data definitions to describe the same real-world conditions. This greatly complicates data sharing, increases potential translation errors, and increases integration costs. Ideally, all ITS data definitions can be developed in a cooperative fashion.

3) Limited scope and lack of forwards compatibility: Engineers within the silo teams often attempt to "optimize" their design; however, without a complete knowledge of how data can be used, it is impossible to know if a design is truly optimal or not. This can partially be overcome by ensuring that the reference architecture is developed with as broad a scope as practical, but since innovations occur over time, no effort can be omniscient about how the data will be used; one can only attempt to consider as much data as possible.

4) Competing efforts: A final challenge facing any development team is that there are often different competing and/or overlapping efforts across the world. Once standards are developed, it is often difficult and expensive to harmonize the results at a later point.

This document attempts to address the first issue by identifying different protocols that have been suggested for use within the ITS industry, comparing their respective characteristics, and suggesting a preferred set of protocols for future use.

6 Summary of needs and considerations

6.1 General

In order to evaluate specific technologies, the analysis described in this document began by identifying the key stakeholder needs and considerations for data distribution. Recognizing that the needs of ITS vary based on the specific information flow considered, an analysis of the Harmonized Architecture Reference for Technical Standards (HARTS; see Reference [9]) and ARC-IT v8.3 (see Reference [6]) was performed to identify the types of information flows throughout ITS and to determine the possible characteristics driving the selection of data exchange technology.

6.2 Types of information flows

6.2.1 General

Information flows were characterized by general pattern of information exchange. Those types were then assessed for whether they might be appropriate for satisfaction through a data distribution system. Note that these categorizations attempt to cover 80 % or more of the information flows in ARC-IT, but do not pretend to be an exhaustive and complete assessment.

6.2.2 Non-emergency information sharing

The first type of flow is information of a non-emergency nature, typically shared by central stations. This information can be personalized and can be consumed by any other kind of station (personal, vehicular etc.). Examples are traveller information (centre-field/personal/vehicle), centre-centre traffic, environmental and maintenance information. These flows are probably well-suited for DDF.

6.2.3 Emergency information sharing

The second type of flow is information of an emergency nature, typically shared by central stations. This is generally of two types:

— alerts to other stations, but particularly motorists to alert them of a widespread issue to which they can react, for example, disaster information, severe weather, Amber or Silver alerts. These exchanges are likely to be suitable for data distribution.

— alerts and exchanges with other central stations and supporting field stations to manage emergencies and incidents. These exchanges generally require acknowledgement and can be dialogue-driven. They are unlikely to be suitable for data distribution.

6.2.4 Control flows

A third type of flow is information provided from one station to another to direct a given action by the receiver. Typically, this is a central-field flow with a central station controlling the field station. It can or can not require acknowledgement, but it does require authorization and authentication. This type of flow is generally not suitable for data distribution technologies, but it is possible if one is able to define operations that would commonly be enacted identically for a group of field devices.

6.2.5 Interrogatives

Another type of flow is a dialogue where one party sends a request followed by the other party sending a response. These flows are not suitable for data distribution.

6.2.6 Local exchanges

Local exchanges are information exchanges where the source and destination lie in close proximity. These are typically local exchanges (usually using a short-range wireless medium) because there is a performance requirement that dictates low latency. Data distribution functionality is only viable for these scenarios if the DDF implementation technology meets that low latency and message size requirements.

6.3 Characteristics

Information flows are characterized according to the following characteristics, many of which indicate requirements on potentially suitable data distribution technologies:

— Confidentiality: Many information flows require encryption meeting the requirements of FIPS 140-2, level 3. In some cases, it is necessary to maintain the confidentiality to the application entity (i.e. meaning the data distribution technology is unable to decrypt the information). Additionally, some cases require intrusion detection and mitigation functions that can inspect data distribution messages.

— Integrity: Virtually all flows are likely to need integrity meeting the requirements of FIPS 140-2, level 3, that ensures that received information is authenticated and authorized. In some cases, intrusion detection and mitigation functions that can inspect data distribution messages are also required.

— Availability: Some information flows require support for multiple communication technologies to allow communications when the primary communication channels are unavailable.

— Latency: While most information flows within the architecture allow for up to 2 s of delay between production of the data and its consumption; there are a relatively small number of flows where ultra-low latency (100 ms) is required. Support for the ultra-low latency requirement can be handled by other means if necessary.

— Throughput: For most flows, the data distribution technology often delivers at least 10 kb/s of aggregate data subscriptions from a single ITS-SU source. In a few cases (5-10 % of information exchanges), the data distribution technology needs to be able to deliver up to 500 kb/s of aggregate data subscriptions from a single ITS-SU source.

— Pseudonymity: For most information flows, pseudonymity is not required (i.e. it is acceptable or even desired for the receiver to be able to identify the source).

— Quality of service: The data distribution technology can provide at least a high level of assurance that the data throughput expectations will be met under all conditions, and in some cases, it is necessary to be able to guarantee this.

— Continuity: The data distribution technology can provide a level of assurance that the data exchange capability can be maintained for the defined quality of service for a period of time.

— Non-repudiation: Most information flows require some form of non-repudiation; in many cases an explicit acknowledgement is required. As noted above, flows requiring explicit acknowledgement are probably not suitable for fulfilment by data distribution technology. For flows that do not require such acknowledgement, the data distribution technology can support non-repudiation services such that the sender of a message is not able to successfully claim that it did not send it.

6.4 Solution characteristics

6.4.1 General

Each solution is also characterized by several other factors as follows:

— Communication technology: The data distribution technology can be readily deployed using a range of IP-based communication technologies.

— Misbehaviour reporting: The data distribution technology can report any misbehaving actors to the appropriate systems to ensure that all systems can be properly prepared.

— Geofencing: In several environments it is useful to restrict publications to a specific geofenced area and/or to restrict publication content to information related to a geofenced area.

— Flow filters: In many cases, it is desirable to allow a subscriber to request topic publications to be filtered to better meet its needs based on frequency, accuracy and other parameters. For example, while a source can provide once-per-second updates, a subscriber might only need and want once-per-minute updates.

— Efficient repackaging: In some cases, it can be advantageous if the data distribution technology is able to combine topics from multiple updates from one or more sources and package them into a single publication to meet the needs of each user.

— Aggregation: In some cases, data is commonly used in aggregate; it would be useful if the data distribution performed this aggregation rather than depending on the endpoint to do it.

6.4.2 Architectural topology

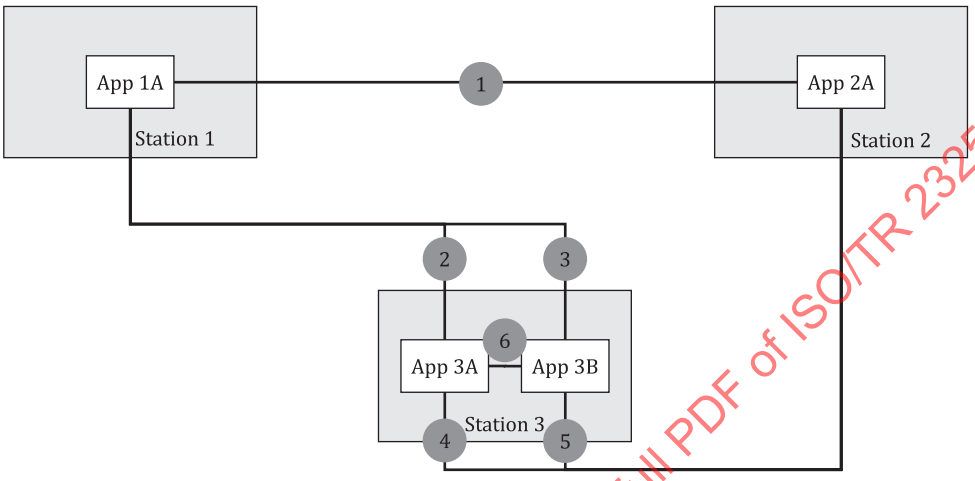
6.4.2.1 General

Part of the goal in sharing data among systems is to minimize the complexity associated with maintaining connections between the various components. Each mechanism for information sharing is

based on an architectural topology that can generally be grouped into one of four styles as described in the following subclauses.

6.4.2.2 Mesh topology

Within a mesh topology, every application entity is required to establish a connection with every other application entity with which it wants to communicate. Once a connection is established, the two applications can subscribe for information and provide publications as necessary. The mesh topology is depicted in Figure 1.



NOTE 1 Connections are logical; all inter-station communications can exist on a single physical medium (e.g. over the air). Each interface is designed for a specific need.

NOTE 2 The number of connections can be determined by the following formula:

$$c_a = c_s = a \times (a - 1) / 2$$

where

c_a application connections

c_s inter-station connections

a number of applications

Figure 1 — Mesh topology

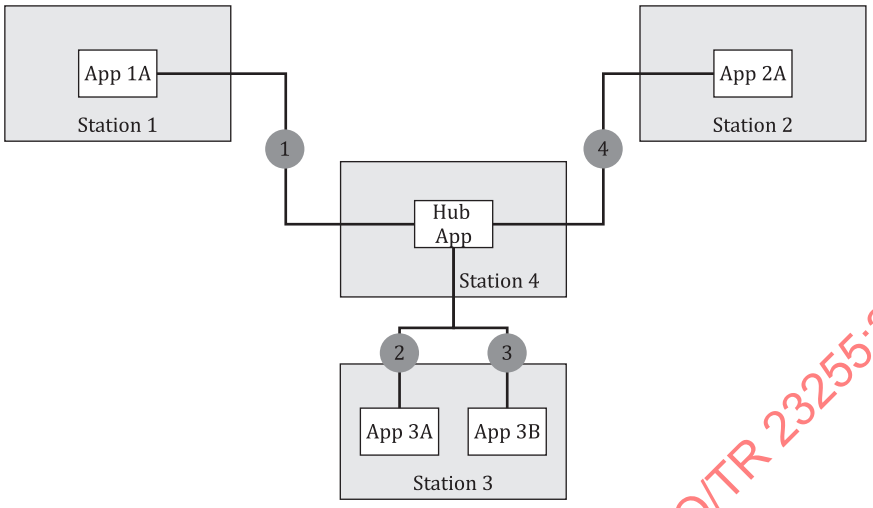
The mesh topology has the advantage that an application providing data can ensure that the application requesting the data is authorized to receive it. However, this also means that each application has to spend resources managing connections and authorizing requests. This can be especially challenging in a cooperative environment where requesters are not necessarily part of a pre-defined list and the number of connections is not necessarily constrained.

While this type of network shares information, it does not meet the definition of a data distribution technology because it does not include generalized layer 5-7 services to handle the data distribution. It is mentioned in this document to contrast data distribution technologies against this more traditional information exchange design.

6.4.2.3 Hub-and-spoke topology

In a hub-and-spoke topology every spoke application entity is required to establish a connection with a hub application. The spoke can then subscribe for information or publish information to the hub. The

hub then has the responsibility of forwarding the publications to all applications that have subscribed for the data. The hub-and-spoke topology is depicted in Figure 2.



NOTE 1 In theory, a Hub app can exist in any station.

NOTE 2 The Hub App can become a single point of failure.

NOTE 3 Each interface represents a superset of all needs.

NOTE 4 The number of connections can be determined by the following formula:

$c_a = c_s = a$

where

c_a application connections

c_s inter-station connections

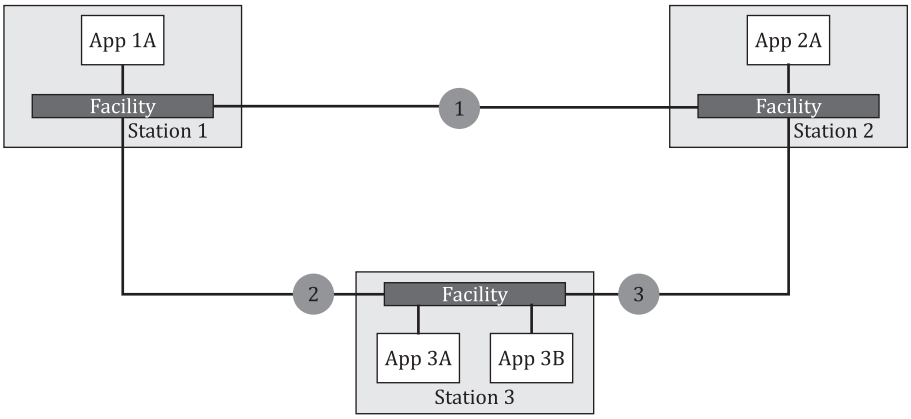
a number of applications

Figure 2 — Hub-and-spoke topology

The hub-and-spoke topology has the advantage that an application providing data can focus on providing its core service while managing a single connection. However, it delegates the authorization task to a remote hub application, which potentially raises issues in a C-ITS environment where the hub application is a separate system (i.e. owned and/or operated by a different legal entity and therefore increasing the number of legal entities with theoretical access to the data). The design also presents challenges for a constantly changing network where devices are mobile and are constantly connecting and disconnecting.

6.4.2.4 Databus service topology

Within a peer-to-peer data distribution topology, every device supports its own service that acts in a manner like a hub. Each application within each device connects to its local hub service. The hub service then connects to the hub services in other devices. Applications publish information to their local hub. The hub service then takes care of distributing the information to other local entities and remote hub services that are authorized. The peer-to-peer topology is depicted in Figure 3.



NOTE 1 Each interface represents a superset of some needs.

NOTE 2 The number of connections can be determined by the following formulae:

$c_a = a$

$c_s = s \times (s - 1) / 2$

where

c_a application connections

c_s inter-station connections

s number of stations

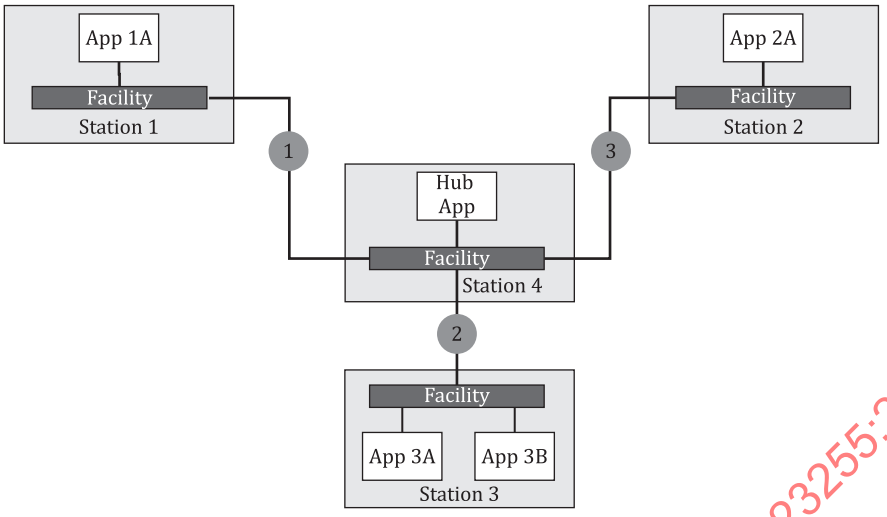
a number of apps

Figure 3 — Peer-to-peer topology

The peer-to-peer topology has the advantage that an application providing data can focus on providing its core service while managing a single connection; further, the authorization task is still largely controlled by a local service within the same system. While a portion of the authorization task is the responsibility of the remote hub service, data will only be sent to the remote service if the remote service (and hence that system) has authorization. The biggest challenge for this design is maintaining connections in the mobile devices. However, this is less of a problem than in some other designs since there are fewer connections to maintain and the management of these connections are concentrated in dedicated software.

6.4.2.5 Hierarchical hub topology

The third data distribution technology is an evolution of the first two. A hierarchical hub topology combines the concepts of the hub-and-spoke and peer-to-peer topology. Every device supports its own service acting in a manner like a hub. The various devices also connect via a device hub. Each application within each device connects to its local hub service. The hub service then connects to a central device hub. Applications publish information to their local hub. The local hub service then takes care of distributing information to the device hub, if authorized. The device hub then distributes to end applications via their own local hubs. The hierarchical hub topology is depicted in Figure 4.



NOTE 1 Each interface represents a superset of all needs.

NOTE 2 The number of connections can be determined by the following formula:

$c_a = c_s = a$

where

c_a application connections

c_s inter-station connections

a number of applications

Figure 4 — Hierarchical hub topology

6.4.2.6 Summary of topologies

The various advantages and disadvantages of each topology are summarized in Table 1.

Table 1 — Summary of topologies

Characteristic	Topology			
	Mesh	Hub-and-spoke	Peer-to-peer	Hierarchical hub
Connections	Application-to-application	Application-to-hub	Facility-to-facility	Facility-to-hub
Code size	High	High	Low	Low
Processing	High	Low	Moderate	Low
Single point of failure	No	Yes	No	Yes

6.4.3 Technology maturity and deployment characteristics

Each technology is characterized by the following maturity characteristics:

— Standardization status: indication of the standard organization(s), standard number, date approved (or current draft status).

— Time in marketplace: time since the first version of the specification was deployed.

— Suppliers: identification of specific products that can be used to develop an ITS implementation that conforms to the standard.

— ICT deployments: list of example uses of the technology outside of ITS.

— ITS deployments: list of specific uses of the technology within ITS.

— Notes: any other relevant information.

Protocols can be further characterized by their dependencies on other technology, the impact that use of the protocol would have on development and deployment, and performance characteristics, as noted in the following list:

Dependencies:

— Network: indication of whether the protocol requires a connection-oriented stack (e.g. TCP) or whether it can work connectionless (e.g. UDP).

— OS: list the target OS supported by the current products.

— Languages: list of the programming languages with which the current products interface.

— Firewalls: indication of the level of effort required to set up communications through firewalls.

Impacts:

— Device deployment: indication of the requirements that are assumed on system devices, for example, whether or not all devices have to support this technology; if gateways can be used; etc.

— Infrastructure topology: indication of the implications on network topology, for example, does a hub have to be present and if so, can this be overcome in areas where there is no infrastructure hub available.

— Data definitions: indication of language(s) used to describe the data (e.g. ASN.1, XML, IDL, UML) and if there are automated ways to convert among them.

— Scalability: indication of whether or not the protocol can be stripped down to its basics so that it can be easily implemented on a simple device.

Functionality and performance:

— Reference: provision of information on the processing and communications load for live operations.

— Registration and discovery: indication of whether or not the technology automatically discovers available systems and information and registers itself with others.

— Multicast: indication of whether or not the technology supports notifying multiple subscribers of data through a single publication and whether this is achieved via true multicast or broadcast.

— Filtering: indication of whether or not the protocol supports filtered subscriptions.

— Aggregation: indication of whether or not the protocol can perform aggregation and manage subscriptions to aggregated data.

— Support for ITS security mechanisms: indication of whether or not the technology natively supports cooperative ITS security (e.g. IEEE 1609.2). If not, indication of whether existing products can be readily configured to support this feature in an interoperable fashion, and what would be required to support this level of security.

— Data authenticity: authenticability at the data level, being able to say that the data that was provided was provided by a trusted source.

— Access Control over data: who can see each item of data that is published.

— Configurability: ability to manually adjust parameters to adjust memory usage, processing usage; indication of the advantages, the level of difficulty, etc.

— System management: indication of the information and tools available to manage performance of the network.

6.5 Objective analysis

6.5.1 General

A survey of existing literature found few examples of analysis and comparison between protocols satisfying various publish-subscribe architecture patterns. Some testing has been carried out comparing broker-based protocols relevant to web services or IoT deployments. However, no objective comparison between OMG DDS and other protocols was found. The ITS community has discussed OMG DDS for many years but does not yet have any deployment experience. There is limited experience in the US with Apache Kafka (in the Wyoming connected vehicle pilot) and OASIS MQTT (both in the Wyoming and New York City connected vehicle pilots) and OASIS AMQP has been employed in the NordicWay C-ITS deployment. But, no quantitative assessments, objective lessons-learned or related documents are available.

Consequently, an independently funded and non-partisan software team was instructed to construct a test environment suitable for evaluating the scalability of protocols that could fit the publish-subscribe data dissemination pattern.

Separately, tests were designed to reflect the current understanding of ITS deployment patterns by leveraging the content of existing ITS reference architectures.

Tests were conducted, varying one parameter at a time (message size, number of publishers, etc.). Data was collected and analyses were performed. The following subclauses describe the test results and early quantitative conclusions. They also note more subjective lessons learned from development and implementation. Details of the test environment are described in Annex A.

6.5.2 Protocols tested

6.5.2.1 Advanced message queuing protocol (AMQP)

OASIS AMQP (also published as ISO/IEC 19464) is a symmetric peer-to-peer protocol operating over TCP/IP that enables the development of clients, brokers and routers. A given node can operate as client and broker, or simply as a router. In this way AMQP can support hub-spoke, peer-peer and hierarchical hub-spoke architectures. AMQP works with transport layer security (TLS), and enables publish-subscribe behaviour through filters at brokers/routers. In practice, many AMQP implementations view it as a hub-based protocol, but this is related mainly to the way AMQP is implemented and configured, not the protocol itself. AMQP does not seem to include a dynamic method for learning about other AMQP devices, so building a peer-to-peer network requires substantial configuration work.

6.5.2.2 OMG data distribution service (DDS)

OMG DDS is a publish-subscribe protocol that does not use a centralized hub at all. Instead, each data producer is also a data publisher. Architecturally, OMG DDS follows the peer-to-peer model. OMG DDS requirements on the data provider/publisher communicate directly with subscribers, necessitating a discovery phase, where OMG DDS-enabled entities establish communications. This discovery phase is contained within the base specification but is often extended by vendors. OMG DDS provides a discovery standard promoting interoperability. However, the standard is extensible, supporting vendor-specific extensions to discovery. One open-source implementation was available at the time of testing.

6.5.2.3 Kafka

Apache Kafka is a distributed streaming platform with three key capabilities:

— publish and subscribe to streams of records, similar to a message queue or enterprise messaging system;

— store streams of records in a fault-tolerant durable way; and

— process streams of records as they occur.

Kafka and the supporting service Zookeeper serve as a hub-spoke publish-subscribe engine.

6.5.2.4 Message queuing telemetry transport (MQTT)

OASIS MQTT (also published as ISO/IEC 20922) is a publish-subscribe protocol designed for the hub-spoke architecture pattern, where data providers tend to be low-power, low-capacity devices. Compared to the other protocols it is relatively lightweight and simple to implement.

6.5.3 Protocols considered and not analysed

6.5.3.1 Constrained application protocol (CoAP)

CoAP is an internet standard protocol designed to deliver small messages between low-power devices in bandwidth-constrained environments. It defines a request/response communications model between participating devices, using UDP for transport. It does not support publish-subscribe and is not suitable for handling ITS data distribution use cases. CoAP sees use in low-data rate sensor environments using low power networks, such as utility metering, but does not have an architectural paradigm that suggests its use for data distribution.

6.5.3.2 JMS

Java message service (JMS) provides message exchange functionality between systems. It provides a Java API to delivery loosely coupled message exchanges at the facilities layer. It enables mesh and pub-sub architectures and handles all of the message distribution but offers few additional features. JMS is well-suited to custom, distributed enterprise applications, but offers no advantages over more specialized data distribution protocols when it comes to moving large amounts of data in the ITS context.

6.5.3.3 Simple network management protocol (SNMP)

SNMP is an internet standard protocol that enables the acquisition, organization and configuration of information. It is well suited to and used for device control, for communications devices (routers, switches etc.). In NTS, SNMP forms the basis for the National Transportation Communications for ITS Protocol (NTCIP) standards used to control ITS field devices. SNMP does not provide one-to-many pub-sub functionality, and thus is not a viable option for widespread data distribution use cases.

6.5.3.4 Simple object access protocol (SOAP)

SOAP is a messaging protocol that depends on other facilities protocols (typically HTTP) to link it to transport. SOAP defines message exchange patterns suited to machine interpretation. However, it requires the use of XML and does not support publish-subscribe, so is not applicable to ITS data distribution use cases. SOAP has seen use in ITS, but the community has moved away from it for a variety of reasons and there is no compelling case to challenge that trend.

6.5.3.5 Simple text orientated messaging protocol (STOMP)

STOMP is a text-orientated wire protocol. It purports to be extremely simple for client development and is stable as of 2012. It is supported by several server products, notably ActiveMQ (below) and RabbitMQ

(a well-known broker primarily focused on MQTT), and many client implementations exist. STOMP was not tested due to its focus on text-based messaging.

6.5.3.6 Representational state transfer (REST)

REST was developed to address some of the difficulties with SOAP. In particular, it aimed to simplify development and use. REST requires HTTP, but can deliver content in several forms (CSV, JSON, RSS). Like SOAP, Websockets and the other protocols in this category, REST supports only the mesh network paradigm, which limits its use to custom applications. Its simplicity and flexibility suggest that REST coupled with well-known output formats such as JSON might be viable for some applications, but nonetheless does not support the primary data distribution use cases.

6.5.3.7 WebSocket

Websocket is a communications protocol designed to provide full-duplex communications to the web environment (HTTP is half-duplex, requiring polling operations). It has no concept of publish-subscribe so is not a viable option for data distribution use cases. Centre-to-centre applications could utilize Websocket to build custom ITS applications, however.

6.5.4 Protocols considered and investigated but not tested

6.5.4.1 ActiveMQ, OpenWire and HornetQ

ActiveMQ is an Apache open source, Java-based messaging server. It provides the central hub that supports highly reliable, scalable messaging that can be customized by a developer using its Java API. ActiveMQ supports several wire protocols, i.e. the protocol between data producer and the hub, or data subscriber. Its native protocol is termed 'Core', but it also supports AMQP (1.0), OpenWire, MQTT (3.1) and STOMP (1.0, 1.1, 1.2). Apache OpenWire allows the developer to access ActiveMQ from C, C++, or C# server-side. STOMP can work with ActiveMQ through Ruby, Perl, Python or PHP. Other protocols are limited to Java.

ActiveMQ was discovered after testing was underway; future performance testing could consider ActiveMQ and its Core protocol. It might still be a valid option despite being omitted in the current document.

HornetQ is another Apache middleware product that pre-dates ActiveMQ; all of its code was contributed to the ActiveMQ project and HornetQ is no longer under active development.

6.5.4.2 Extensible messaging and presence protocol (XMPP)

XMPP is a text-based messaging protocol originally developed to provide chat services, and is an open specification. Based on XML but advertised as performance-competitive with JSON, XMPP includes pub-sub as an extension. Architecturally, XMPP looks a lot like some of the other protocols of interest, however its focus on text, XML, and the lack of known high performance server software discounts it for ITS applications.

6.5.5 Summary

Testing focused on the architectural patterns currently seen in ITS, these being peer-to-peer and hub-spoke. No attempt was made to leverage AMQP's flexibility or to construct hierarchical hubs. Future testing might consider these aspects. Future testing might also consider ActiveMQ, and the in-development open-source OMG DDS implementation from the Eclipse Foundation (open source but not available when testing was initiated).

7 Summary of analysis results

7.1 General

The following analyses are reflective of various tests performed on the protocol implementation selected for testing. Tests are described in some detail; additional information about test configurations is located in Annex A. The analyses are naturally limited to the protocol implementations under test. There can be other implementations, or even other protocols that provide better results for some ITS use cases.

7.2 Quantitative results

7.2.1 General

The graphs shown in this clause use the following conventions and abbreviated terms:

— In most cases, latency is plotted along the vertical axis. Latency is a measure from the time of data creation at the publisher to the time of fulfilment of data subscription at the subscriber. Reported latencies are averages across all publisher-subscriber combinations.

— Most plots use a logarithmic scale.

— Avg = average

— AMQP = freely available AMQP used for data publication and subscription

— MQTT = freely available MQTT used for data publication and subscription

— Kafka = freely available Kafka broker used for data publication and subscription handling

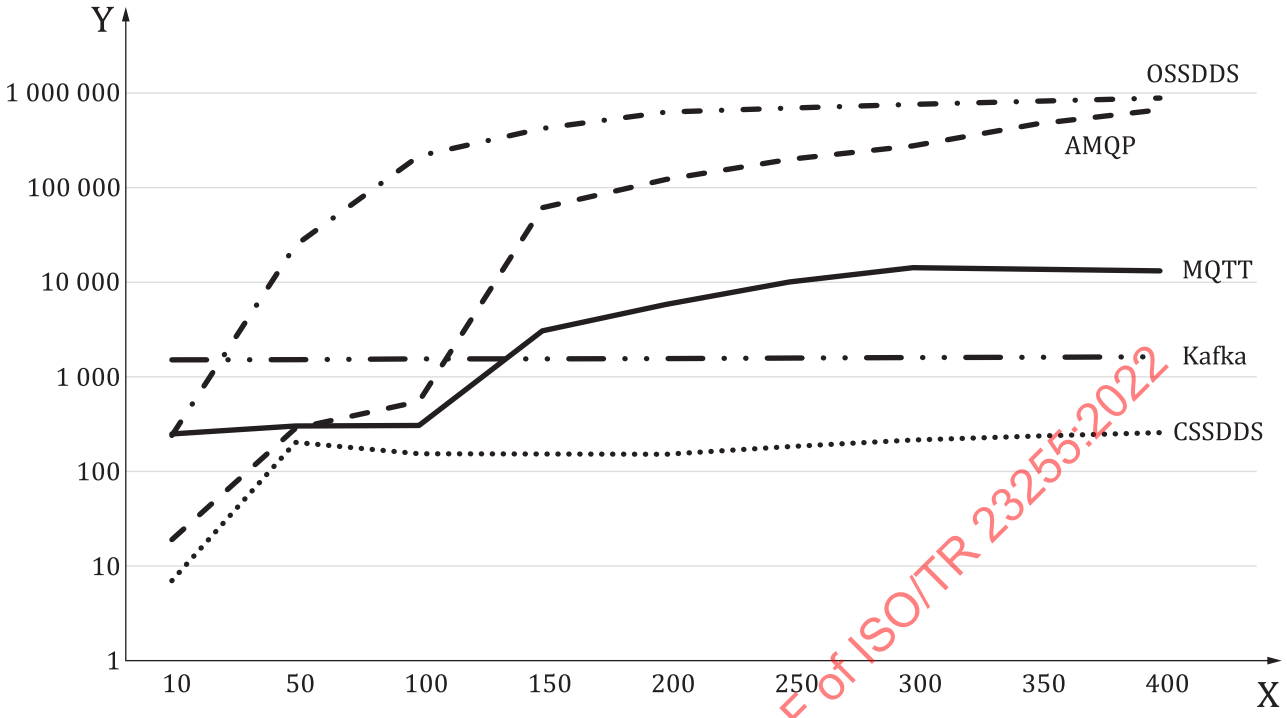
— OSSDDS = freely available OMG DDS middleware used for data publication and subscription

— CSSDDS = commercially available OMG DDS middleware used for data publication and subscription

The first three protocols are commonly distributed as freely available software; brokers and commercial installations have costs, which do not apply to this evaluation. For OMG DDS, there is one available open-source implementation that was used initially as a learning tool, but it has well-advertised performance limitations. A commercial version was used for all tests, but only after the necessary knowledge had been gained from the open-source version (see notes in the following subclauses related to OMG DDS knowledge capture and development).

7.2.2 Many2One

This test emulated the collection of data from a large number of sources, as in the collection of vehicle situation data by a transportation information centre or traffic management centre. It varied the number of publishers and provided data to one subscriber. Figure 5 shows the average time it took for a packet of data produced by a publisher to arrive at the subscriber.



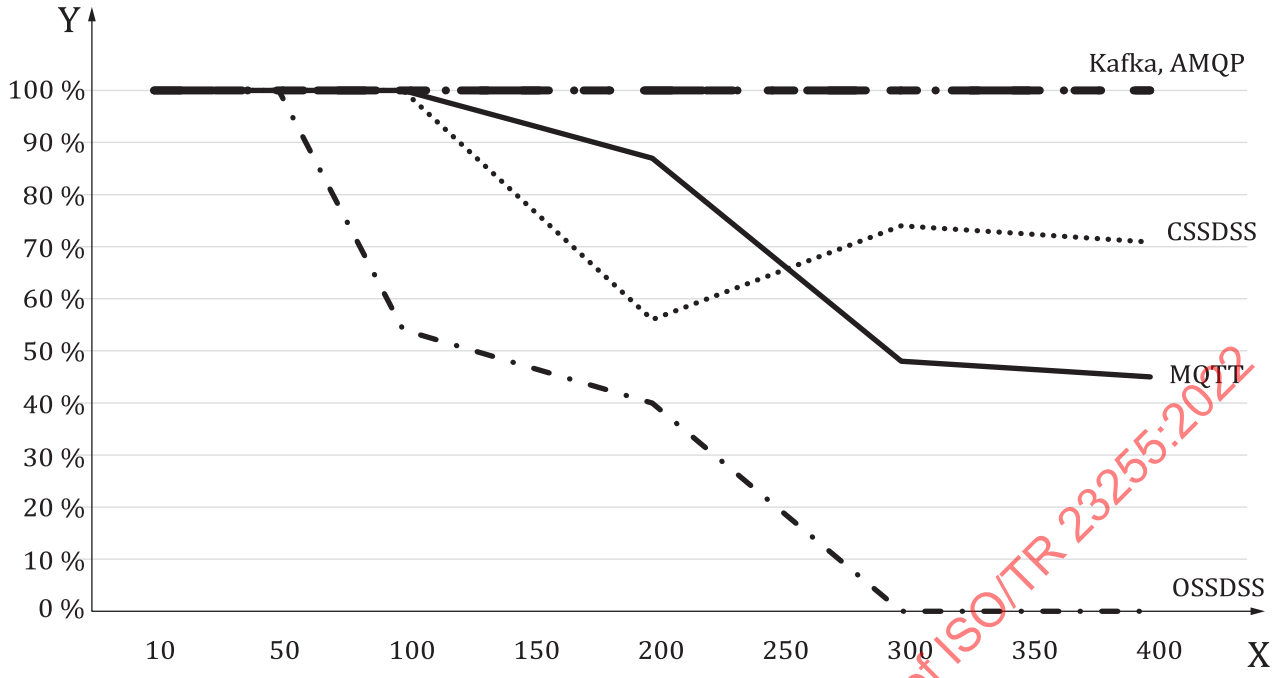
Key

X number of publishers

Y average latency in ms

Figure 5 — Latency with one subscriber

Figure 6 includes the percent satisfaction, that is, the amount of times that the subscription that was completed.



Key

X number of publishers

Y success rate in %

Figure 6 — Success with one subscriber

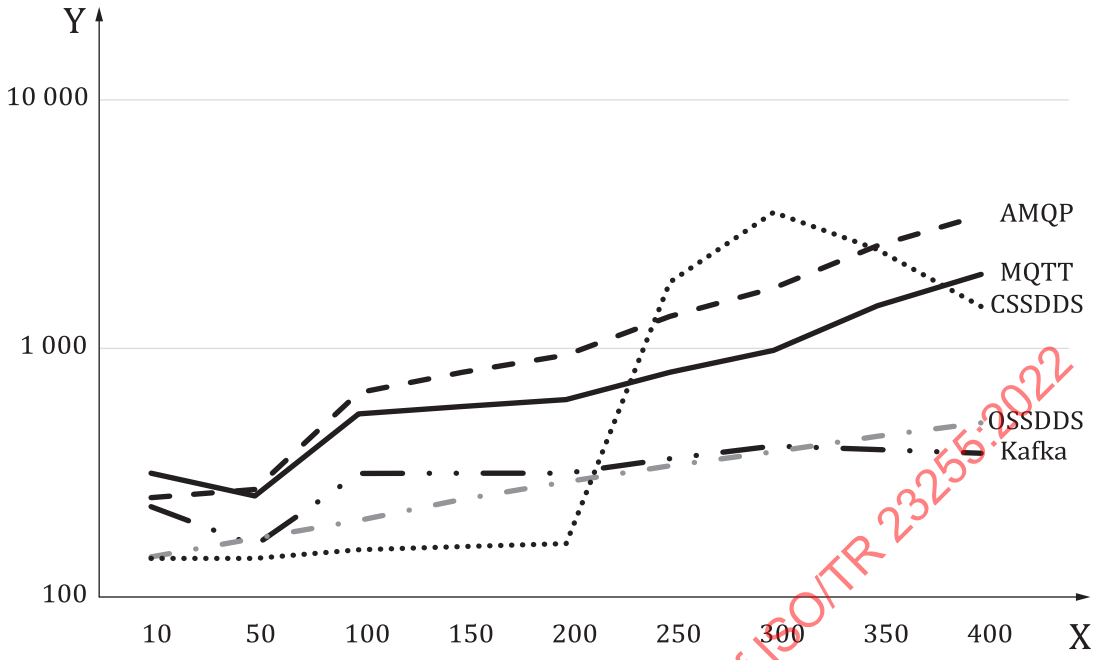
The protocols using brokers, while ostensibly of similar architecture, have significantly different behaviour as the number of publishers is scaled. AMQP is the most efficient of the brokers at low numbers of publishers, but the worst at high numbers. Kafka is consistent, while MQTT is in between. The commercial OMG DDS solution consistently shows the highest performance in this environment, though from 50-100 publishers the difference between AMQP, MQTT and CSSDDS might not be significant.

In many instances once the number of publishers increases, some subscriptions are not satisfied. AMQP starts failing above 400 publishers. MQTT starts failing at 200 publishers but maintains some level of success scaling to 1 000. The commercial OMG DDS also started failing at 200 publishers, and above approximately 400 publishers failed to complete at all. The open-source OMG DDS failed to complete with large numbers of publishers, which explains the gap in the OSSDDSAvg line.

A subsequent test with increased network bandwidth for the 400-publisher case showed similar results, indicating that scalability issues are not network related. It is unclear if loading on the publisher virtual machine was an issue; this might be the subject of any future analysis and potential deployers are advised to consider this loading issue.

7.2.3 One2Many

This test emulated the widespread distribution of a traveller information flow, or a centre configuring a multitude of field devices. It used one publisher and varied the number of subscribers. The average time it took to deliver a packet is shown in Figure 7.



Key

X number of subscribers

Y average latency in ms

Figure 7 — Latency with one publisher

All protocols held 100 % success except for CSSDDSAvg in the 400-subscriber case, which possibly explains the apparent increase in performance. Latency dropped for the CSSDDS subscriptions that were satisfied (84 %), but 16 % of subscriptions were unfulfilled.

7.2.4 10 to Many

This test expanded on the 'One2Many' test by increasing the number of publishers, to identify if the increased publication load affected delivery success and/or latency. As Figure 8 and Figure 9 illustrate, latency followed the pattern established with the One2Many tests, but that latency increased by a substantial amount:

— AMQP and MQTT showed a tolerance for no more than 10s latency, at which point they began dropping messages.

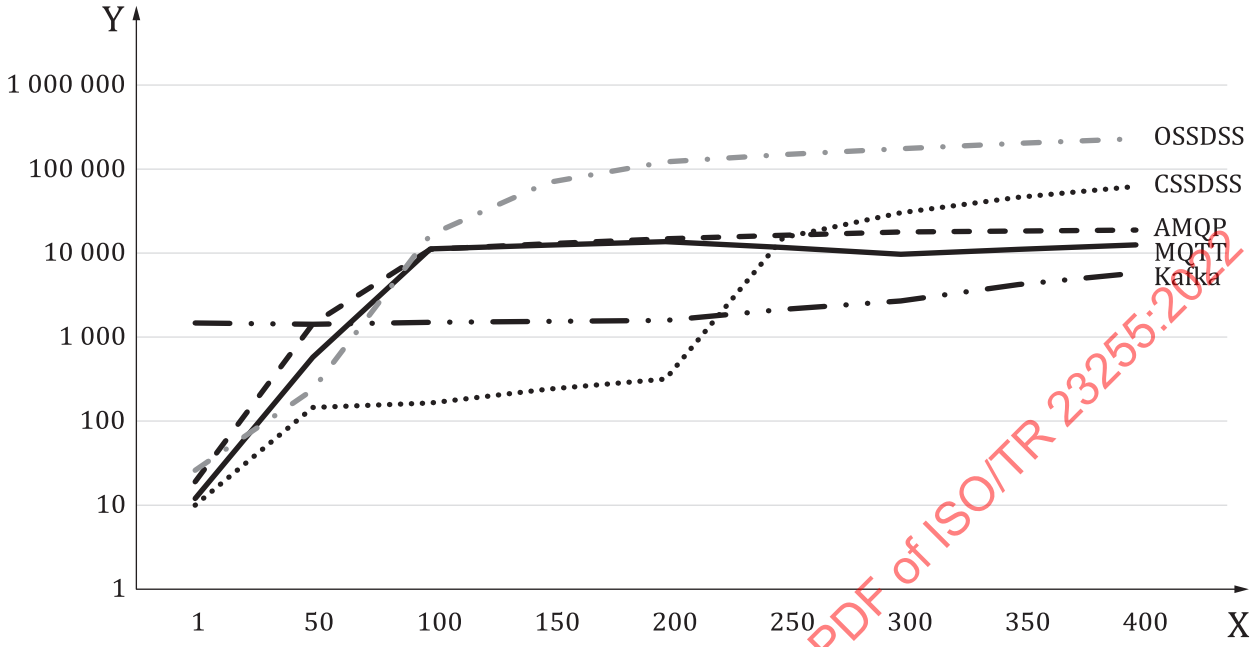
— Kafka's base latency started just under one order of magnitude higher (just over 1s from approximately 300 ms, with 10x the message traffic), but then held just as steady under the 10-publisher case as under the 1-publisher case. In every case Kafka delivered 100 % of messages.

— The open-source OMG DDS implementation, which performed with comparatively low latency and high satisfaction in the One2Many case, started to fail to satisfy subscriptions once the number of subscribers exceeded 100 in the 10-publisher case (exactly where is unclear due to limitations in the number of tests; this change occurred somewhere between 100 and 200 subscribers).

— Open-source OMG DDS exhibited vastly increased latency: two orders of magnitude at 100 subscribers and three orders at 200 subscribers.

— The commercial OMG DDS solution held onto its low-latency performance advantage through 200 subscribers; beyond that point it showed increased latency compared to Kafka and also failed to satisfy all subscriptions.

Overall, 10 publishers with 200 subscribers seemed like an inflection point for the OMG DDS implementations, while AMQP and MQTT showed such behaviour even earlier, at 100 subscribers. Kafka demonstrated the most consistent performance of all protocols examined in this test.

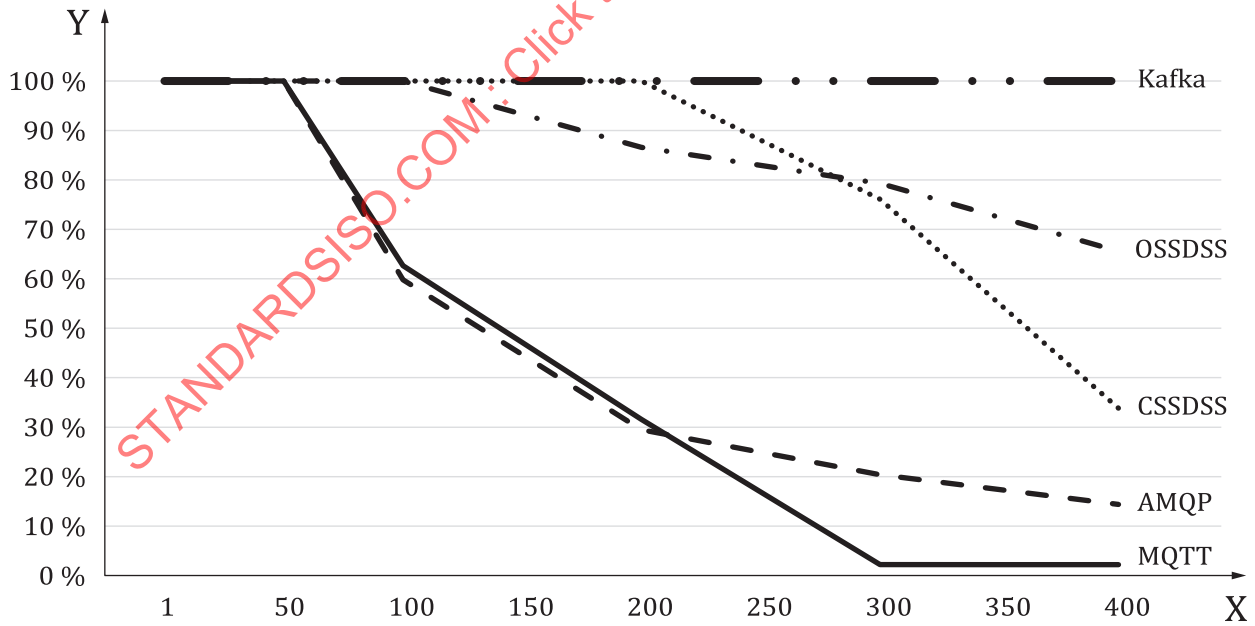


Key

X number of subscribers

Y average latency in ms

Figure 8 — Latency with 10 publishers



Key

X number of subscribers

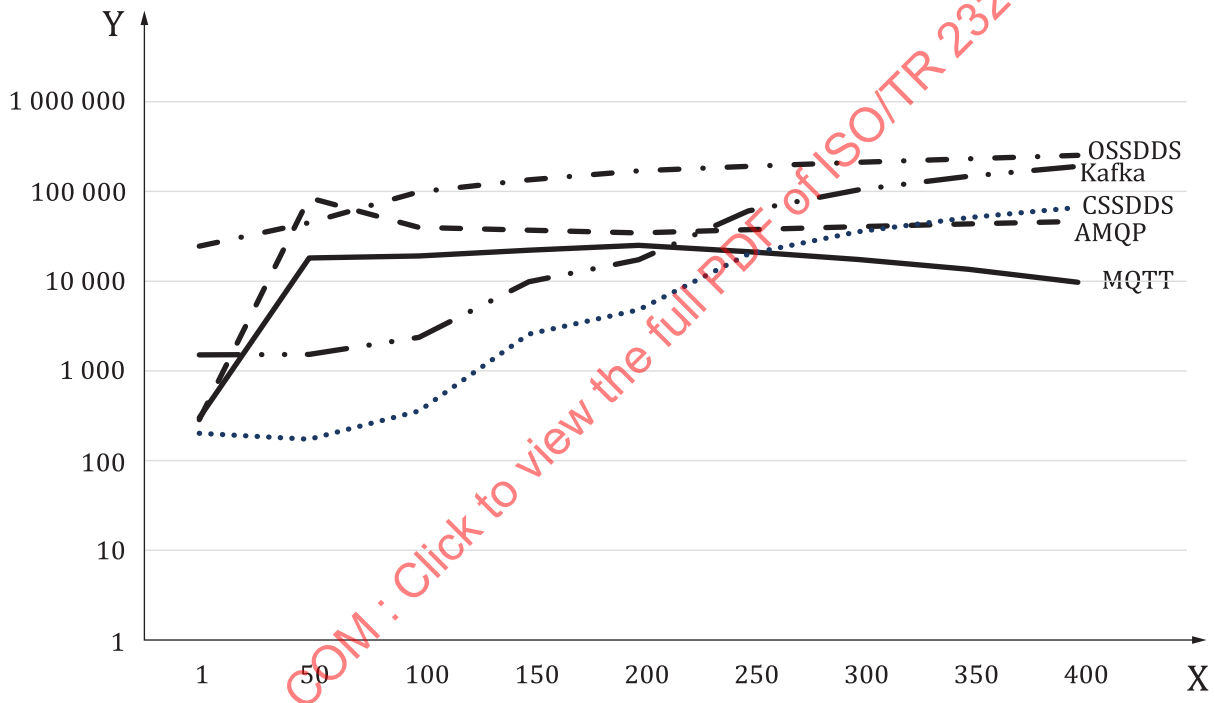
Y success rate in %

Figure 9 — Success with 10 publishers

7.2.5 50 to Many

This test built on the previous test, expanding the number of publishers to 50 and scaling subscribers. This is analogous to a densely interconnected connected-vehicle environment where many entities communicate with one another. While many of these information exchanges are considered candidates for broadcast mode communications, substantial performance gains could be made if pub-sub technologies could provide sufficient performance in the local environment, thus reducing the load on the wireless network. The results are shown in Figure 10 and Figure 11.

Unsurprisingly, performance measures reflected trends from the previous two sets of tests. All protocols saw average latency and success % fall precipitously from desirable values, with the exception of Kafka's success percentage, which never wavered. Unlike previous tests however, Kafka did exhibit significant latency increases once the number of subscribers exceeded 100. Success percentage scaled so poorly that it became clear this would the practical limit of scalability testing. No more than 100 subscribers are really practical with 50 publishers in the test environment's network configuration.

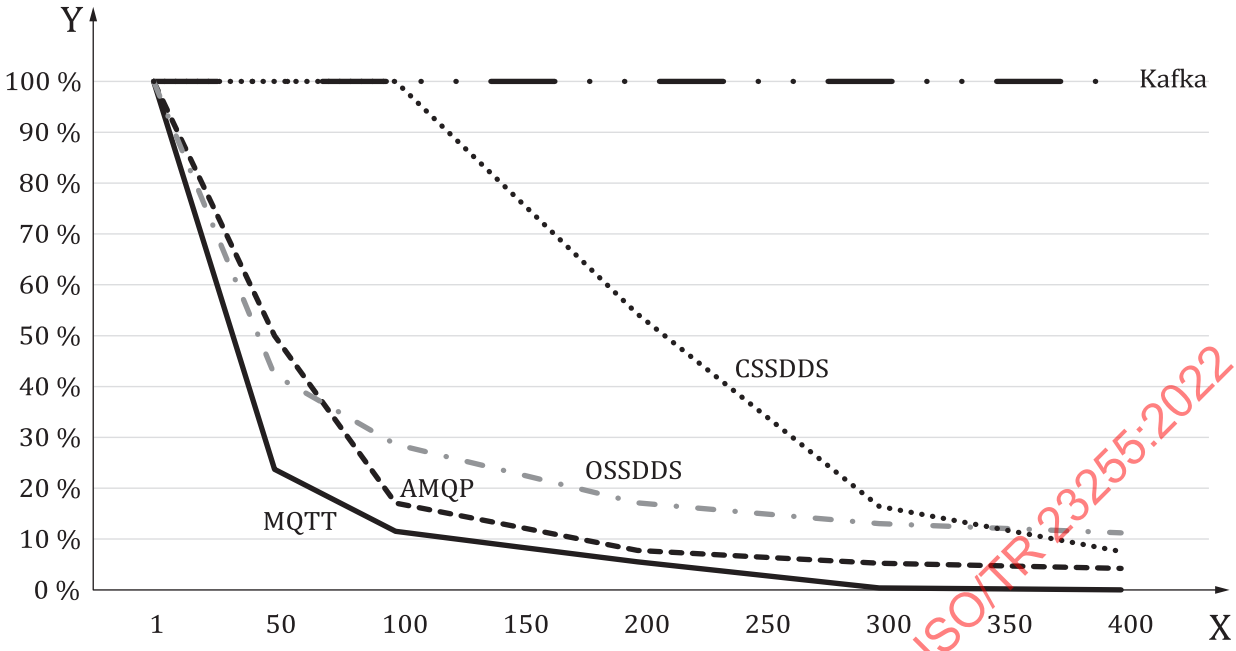


Key

X number of subscribers

Y average latency in ms

Figure 10 — Latency with 50 publishers



Key

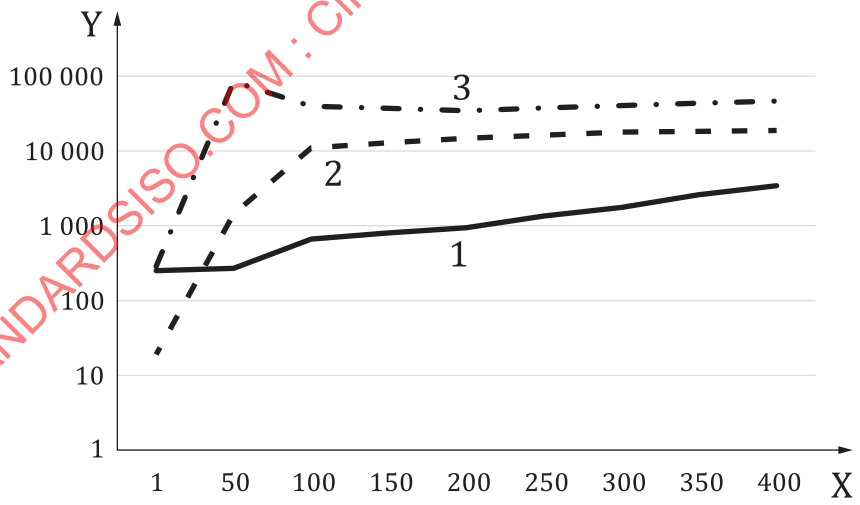
X number of subscribers

Y success rate in %

Figure 11 — Success with 50 publishers

7.2.6 N to N

Figure 12 through Figure 20 summarize some of the previously presented tests in different ways to facilitate visualization of protocol performance.



Key

X number of subscribers

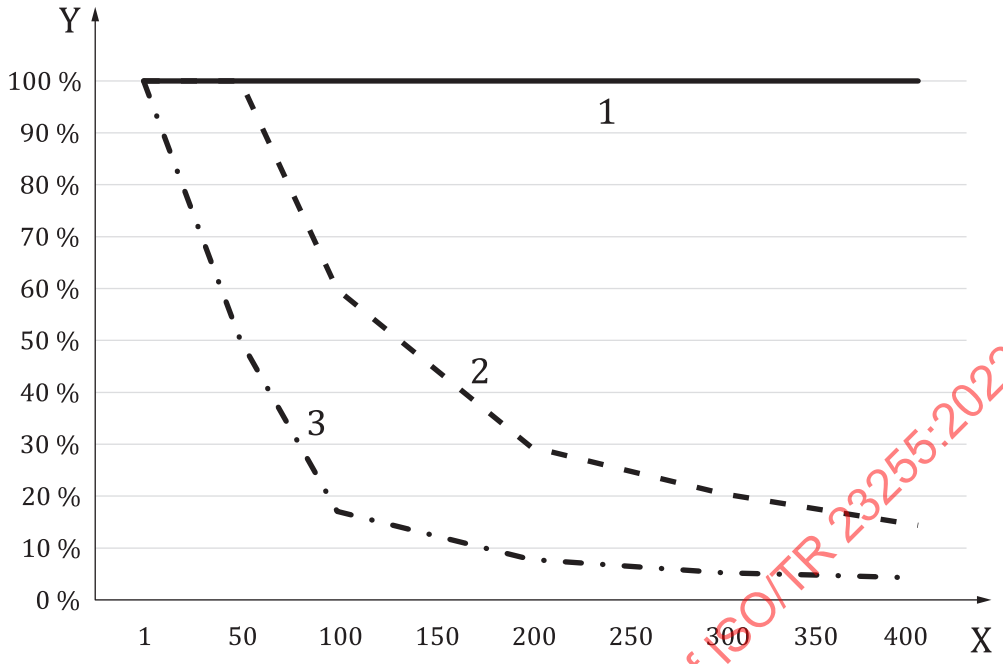
Y average latency in ms

1 1 publisher

2 10 publishers

3 50 publishers

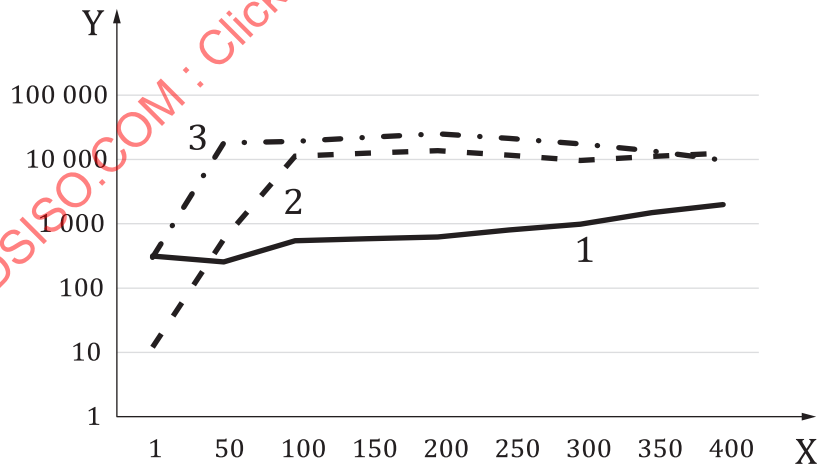
Figure 12 — AMQP latency



Key

X	number of subscribers
Y	satisfaction in %
1	1 publisher
2	10 publishers
3	50 publishers

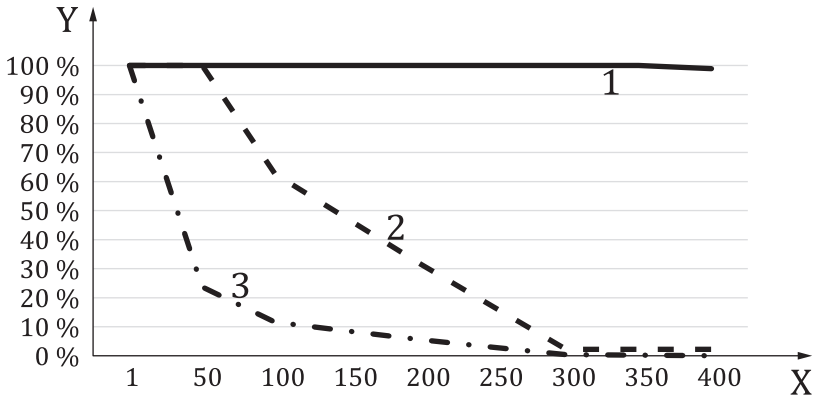
Figure 13 — AMQP success



Key

X	number of subscribers
Y	latency in ms
1	1 publisher
2	10 publishers
3	50 publishers

Figure 14 — MQTT latency



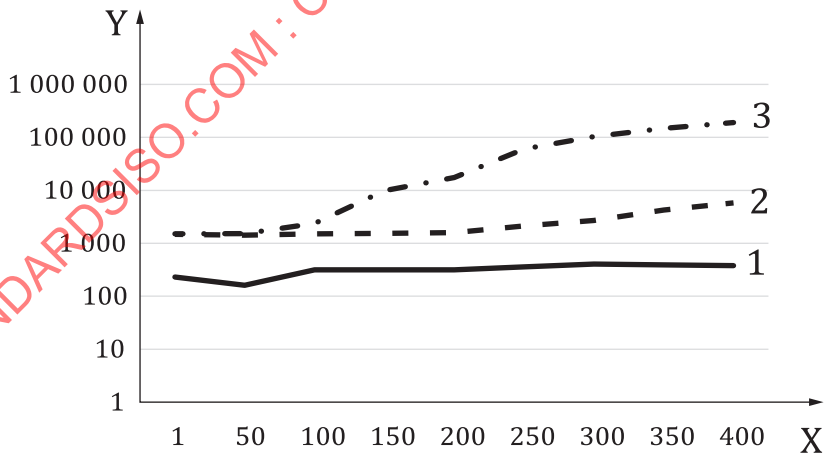
Key

X	number of subscribers
Y	satisfaction in %
1	1 publisher
2	10 publishers
3	50 publishers

Figure 15 — MQTT success

AMQP and MQTT scale in similar fashion, with a flattened latency curve as the number of subscribers increases, for all cases. For both protocols, latency rises to a maximum 10s. In the cases with more than one publisher however, the flattening in latency corresponded to a collapse in percent satisfaction.

Kafka behaved differently. At one publisher, Kafka exhibits slightly more latency at low numbers of subscribers but retains that latency value all the way through the tests at 400 subscribers, at which point it offers better latency performance then either MQTT or AMQP. The 10-publisher case is similar, except that unlike MQTT or AMQP, Kafka never fails to satisfy subscriptions. For the 50-publisher case, where both bandwidth and processing are being taxed, Kafka still holds up and never drops a message.

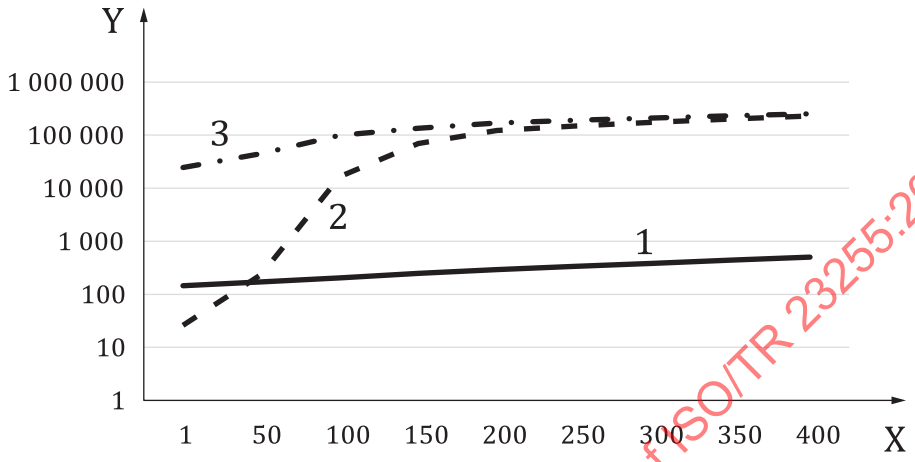


Key

X	number of subscribers
Y	latency in ms
1	1 publisher
2	10 publishers
3	50 publishers

Figure 16 — Kafka performance

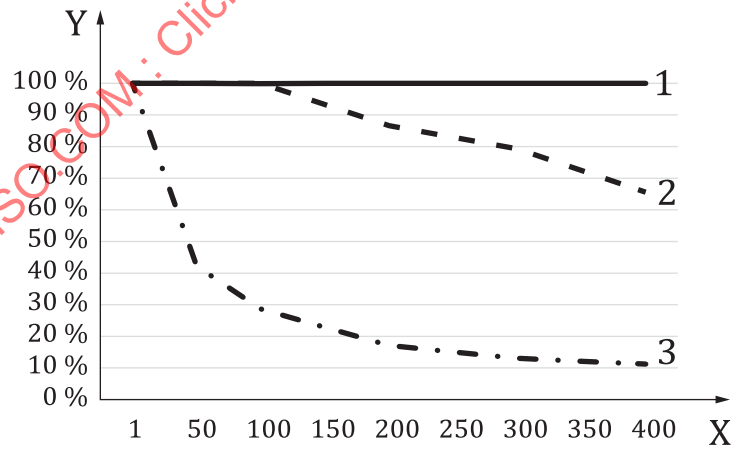
The databus protocol was also different. Both the open source and commercial software tended to scale better than the hub-based lightweight protocols, though both failed to satisfy all publications as the number of publishers and subscribers taxed the infrastructure. The commercial solution performed notably better, offering lower latencies and higher rates of subscription satisfaction at most data points, and especially as the number of publishers climbed. Oddly, the commercial solution falls off more quickly at extremely high loads.



Key

X	number of subscribers
Y	latency in ms
1	1 publisher
2	10 publishers
3	50 publishers

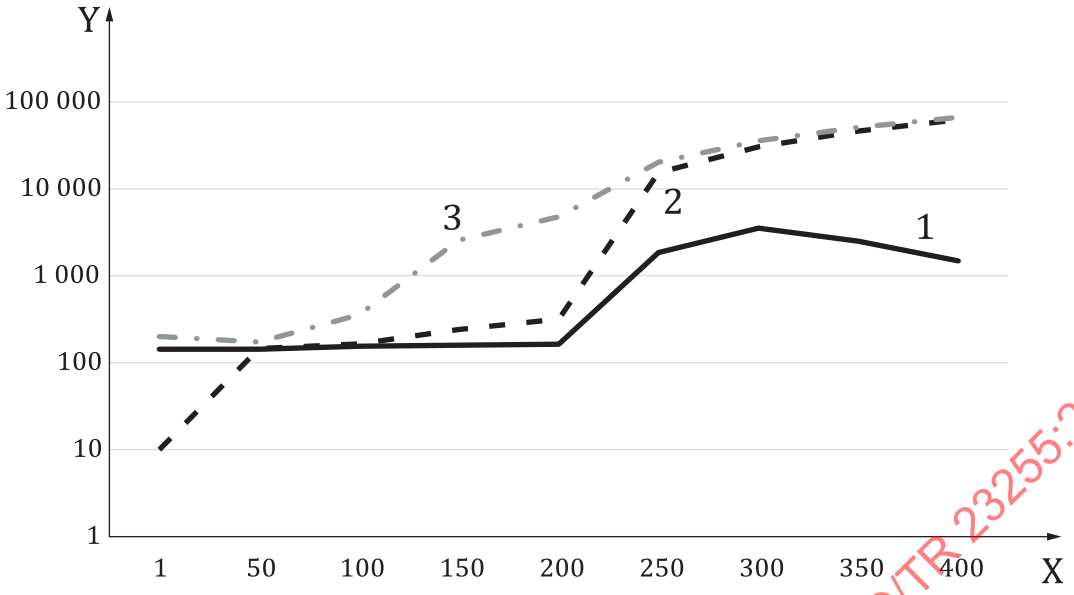
Figure 17 — OSSDDS latency



Key

X	number of subscribers
Y	satisfaction in %
1	1 publisher
2	10 publishers
3	50 publishers

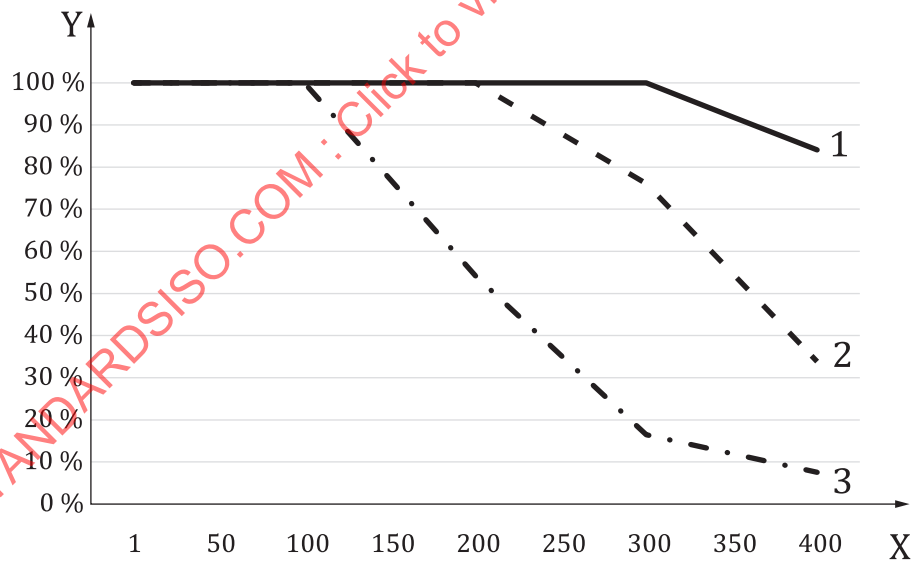
Figure 18 — OSSDDS success



Key

X	number of subscribers
Y	latency in ms
1	1 publisher
2	10 publishers
3	50 publishers

Figure 19 — CSSDDS latency



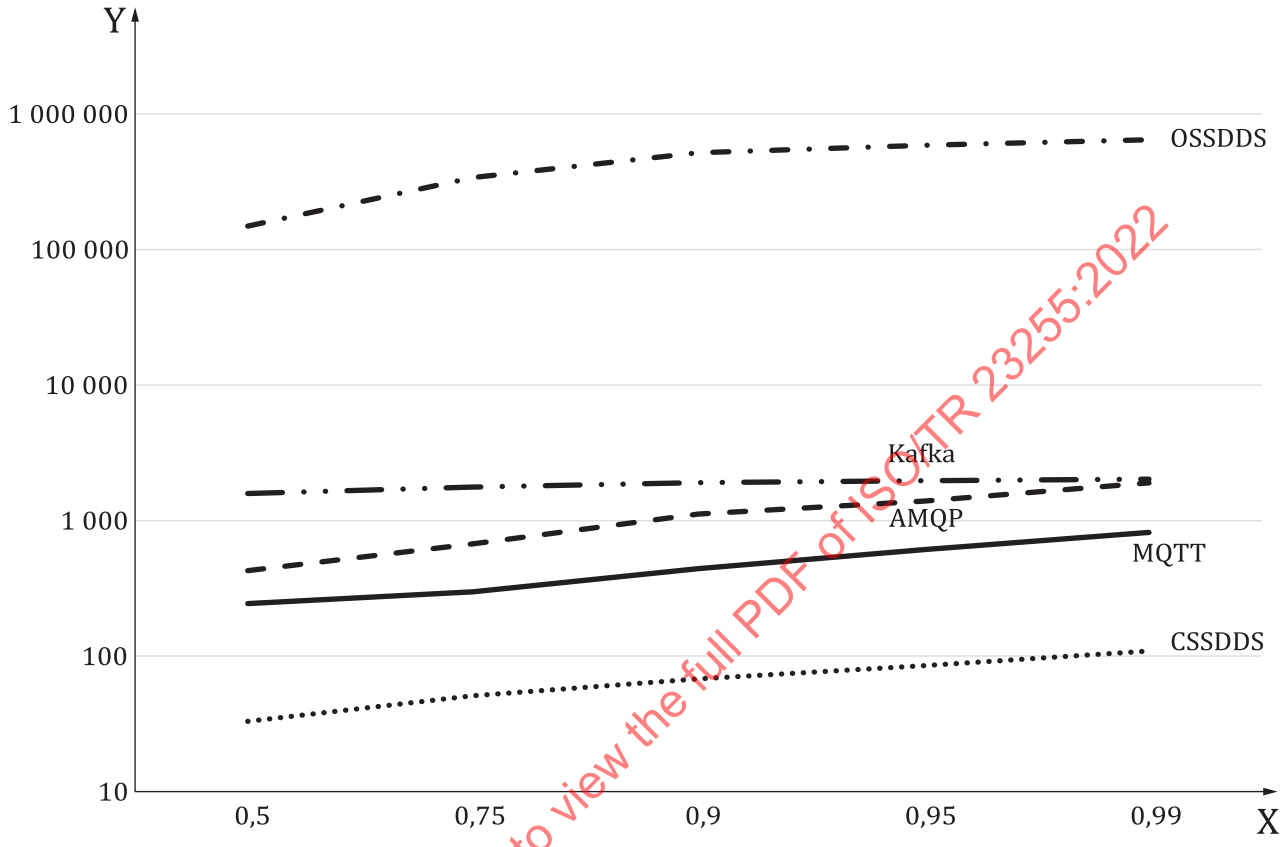
Key

X	number of subscribers
Y	satisfaction in %
1	1 publisher
2	10 publishers
3	50 publishers

Figure 20 — CSSDDS success

7.2.7 Latency as a function of completion percentage

Subscription fulfilment times can stretch under load. Some of this has already been illustrated. Figure 21 illustrates this behaviour for the 100-publisher version of the Many2One case.



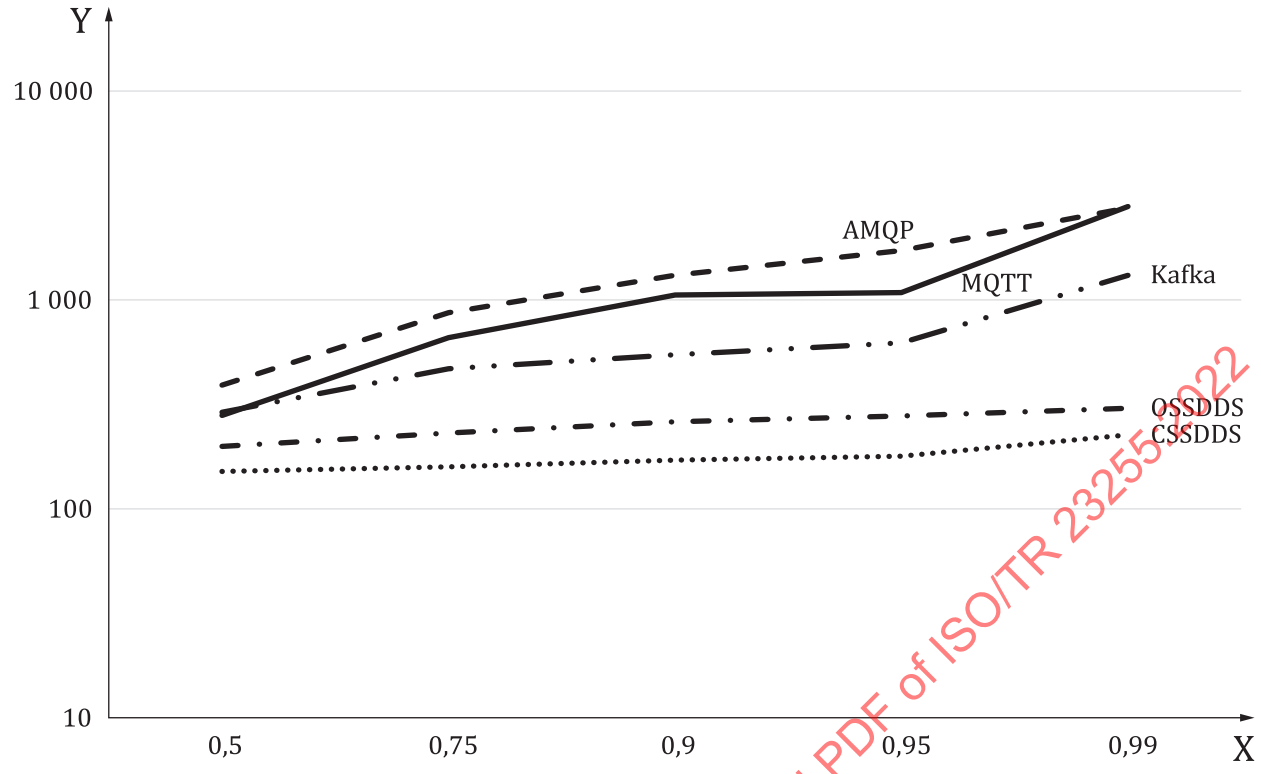
Key

X subscription completion in %

Y average latency in ms

Figure 21 — Latency at completion with 100 publishers

Similar trends are exhibited for the Many2One 100 subscriber case, as shown in Figure 22.



Key

X subscription completion in %

Y average latency in ms

Figure 22 — Latency at completion with 100 subscribers

It is noteworthy that the OMG DDS implementations, even the OSS version, hold up much better than the broker protocols.

7.2.8 Other tests

Some other tests were run to examine boundaries and edge cases:

— The above tests were run with a 1 Hz publication rate. Some of these tests were repeated with slower publication rates to determine if protocols that previously failed might satisfy all subscriptions at lower rates. This was borne out: all protocols performed extremely well when the publication rate was reduced to 0,1 Hz (1 message/10 seconds).

— Considering the distribution of large binary blobs (as might be used for a software update for example), other tests were run where the goal was to distribute a small number (20) of messages to each subscriber, but that 'message' might be large, in this case 100 Kbytes. The three hub-based protocols satisfied all subscriptions up to the 100-subscriber case. At 200 subscribers, AMQP began to fail to satisfy all subscriptions, and at 300 subscribers MQTT began to fail as well. Neither of the OMG DDS protocols could be configured to operate this test. This test is outside the typical use case for these protocols but was run to determine if there might be a cause for using them in this way. The end result suggests 'maybe', at least for Kafka.

7.3 Qualitative lessons learned

While the goal of this testing was primarily to objectively quantify performance differences between protocols, the development team learned a great deal about implementation differences and challenges during construction and operation of the test environment. These are summarized here:

1) Time synchronization between publisher and subscriber is critical to be able to measure performance; this will likely extend to real-world use.

2) AMQP retry logic is not built in; developers will need to build their own retry logic in case the initial connection to the hub fails.

3) AMQP channel connection needs to be reset by the developer after a failure; the connection is not automatically reset.

4) MQTT under heavy loading, does not always gracefully respond to attempts to close the communications channel.

5) The broker-based protocols under test (Kafka, MQTT and AMQP) support TCP only. OMG DDS supports TCP and UDP.

6) More success experienced with OMG DDS when using TCP for discovery but UDP for data exchange; this is likely related to protocol configuration complexity and developer experience. It would be preferable to have UDP for both, but in some network configurations the UDP discovery processes were unreliable.

7) The broker protocols isolate publisher and subscriber: one does not know the IP address of the other. The opposite is true for OMG DDS: publishers and subscribers know one another's IP addresses.

8) OMG DDS discovery process needs to be completed before successful subscription fulfilment; this takes a non-trivial amount of time, such that in a low latency environment several messages will be lost (or the opportunity to send them missed) by the time discovery is complete.

9) OMG DDS is highly configurable, and substantially more complex than the broker-based protocols. Implementers will typically need more time to become comfortable with and develop robust configurations using OMG DDS rather than the broker protocols.

10) Kafka's message queue can hold messages in between subscriber connections, so that if the connection is interrupted, initial subscription satisfaction will first include (relatively) old messages.

11) Data specifications are handled quite differently in OMG DDS vs the broker-based protocols. When specifying data to be exchanged, the broker-based protocols require little or no additional work within the broker configuration. The publisher and subscriber jointly agree upon the names and types of data to be provided. The publisher sends that data through the middleware. On the subscription end, data is discovered and retrieved. Within the application, only the code that creates the message (the publisher) and that interprets data (the subscriber) needs to change.

    With OMG DDS on the other hand, the specification of data is typically configured via a message structure (or IDL) definition file. This file is compiled using tools supplied by the vendor, and then used by the application and middleware code. Effectively, the data to be published or subscribed to needs to be pre-defined and embedded in the middleware.

    OMG DDS does include a mechanism for dynamic data typing that might eliminate the need for compile-time decisions. However, this was beyond the scope of the report and was not evaluated.

8 Summary of protocol characteristics and applicability to ITS

Tables 2-5 provide key characteristics of the protocols that were analyzed within this document.

Table 2 — Advanced message queuing protocol (AMQP) characteristics

Topologies supported		Hub and spoke, hierarchical hub, peer-to-peer[a].
Maturity and deployments	Standardization status	V1.0 standardized as ISO/IEC 19464; specification publicly available at https://www.oasis-open.org/standards#amqpv1.0.
	Time in marketplace	Since 2012.
	Suppliers	Apache ArtemisMQ, VerneMQ (with AMQP plugin), Microsoft.
	ICT deployments	Integrated to Microsoft Azure Service Bus.
	ITS deployments	Nordicway (https://www.nordicway.net/interchange).
	Notes	
Dependencies	Network support	TCP/IP, TLS.
	OS support	Linux and other Java-supporting OS.
	Language support	Implementation-dependent, commonly Java and Go.
	Firewall complexity	Simple: AMQP has IANA-assigned TCP port numbers.
Impacts	Device deployment	Commercially available client code requires a Java runtime engine or OS with Go support.
	Topology restrictions	
	Data definitions	Extensive options for data primitives. Data definitions at topic level are application-specific.
	Scalability	Based on objective testing, scales reasonably given sufficient bandwidth and host processing capability.
Functionality and Performance	Processing load	Not observed to be a challenge.
	Registration and discovery	Needs to be pre-configured.
	Multicast	No
	Filtering	Yes, at the broker. Subscribers can subscribe to filtered queues. Hierarchy and wildcard support not required by protocol, but commonly implemented in brokers.
	Aggregation	Not natively, but can be implemented through code at the broker or provider.
	ITS security support	TLS
	Configurability	Highly configurable; dynamic typing, hosts might function as sources, brokers, routers or sinks.
	System management	Broker management is implementation dependent. A client management specification is under development.

[a] The AMQP Specification indicates that it is a peer-to-peer protocol. However the commercially available solutions currently seem to support broker-based implementations only.