FFI-RAPPORT

20/00480

Experimenting with a big data infrastructure for multimodal stream processing

Audun Stolpe
Bjørn Jervell Hansen
Jonas Halvorsen
Eirik Jensen Opland

Experimenting with a big data infrastructure for multimodal stream processing

Audun Stolpe
Bjørn Jervell Hansen
Jonas Halvorsen
Eirik Jensen Opland

Norwegian Defence Research Establishment (FFI) 16 January 2020

Keywords
Stordata
Databehandling
Informasjonsinfrastruktur
Informasjonsintegrasjon

FFI report
20/00480

Project number
1430

Electronic ISBN
978-82-464-3254-0

Approvers
Jan Erik Voldhaug, *Director of Research*
Ole-Erik Hedenstad, *Research Manager*

The document is electronically approved and therefore has no handwritten signature.

Summary

It is an important part of the Armed Forces' activities to monitor Norwegian land areas, the airspace, the sea and cyberspace. This surveillance supports both traditional tasks such as defending sovereignty or crisis and conflict management, as well as civil-military tasks such as rescue services and environmental preparedness. The overall response time of the Armed Forces, as well as the quality of its operational decisions, depend on the ability to perceive a situation, interpret it and understand it, that is, on the level of situational awareness.

New communication technologies and the ever-increasing availability of computing power today make it possible to utilize data of a variety and volume that can significantly enrich situational awareness in the future.

From a computational point of view, progress in this area depends on whether we have computational models that are able to translate data into relevant real time intelligence, and whether we are able to coordinate machine clusters that, working together, are capable of adapting to potentially very large spikes in the quantity or complexity of available information (complexity being understood as the amount of processing power it takes to convert data into actionable intelligence).

In this report we take a closer look at some general properties such a machine cluster could reasonably be expected to have, as well as the matching characteristics a surveillance algorithm must have in order to exploit it efficiently. Although it is not reasonable to assume that all types of surveillance tasks and information needs can be served with identical system support, the working hypothesis in this report is that some general systemic properties will be sufficient for most cases. Examples include, loose coupling, scalability, fault tolerance and parallelizability.

We do not claim to have proved or refuted this hypothesis (i.e. that some general systemic properties will be sufficient for most cases), but will be content for now to adduce some experimental evidence in support of it. In other words, the method we adopt is empirical. More specifically, we do an experimental case study of high velocity stream reasoning supported by a scalable coordinated machine cluster running a variety of software components and algorithms. The various components and algorithms will be described in more detail as we go. By stream reasoning, we shall mean the operation of turning continuously incoming data into actionable intelligence in real time. The case study, more specifically, attempts to detect illegal, unreported, and unregulated fishing from a stream of AIS reports, supplemented with geographic information as well as with additional on- and offline information about ships, landing bills and more.

The experiment was designed to see to what extent standard software components can be utilised to build a stream processing infrastructure meeting the identified requirements. The outcome of the experiment was a confirmation that the chosen core components essentially provided a streaming infrastructure with the desired properties, mainly due to the characteristics of the core component Apache Kafka. The main deviation was the infrastructure's fault tolerance ability: During the experiment, further study of Kafka's handling of network partitioning casted doubt over its ability to handle such situations. As this experiment was conducted on a robust network, the infrastructure's tolerance for network partitioning was not tested. This is, however, an interesting avenue for further work, as network partitioning is a characteristic of tactical networks.

Sammendrag

Det er en viktig del av Forsvarets virksomhet å overvåke norske landområder, luftrommet, havområder og det digitale rom. Overvåkningen skal støtte både tradisjonelle forsvarsoppgaver slik som suverenitets- og myndighetsutøvelse eller krise- og konflikthåndtering, såvel som sivil-militære oppgaver slik som redningstjeneste og miljøberedskap. Responstiden til Forsvaret, samt kvaliteten på Forsvarets operative beslutninger, avhenger av situasjonsforståelsen, det vil si av evnen til å oppfatte, tolke og forstå en situasjon.

Nye kommunikasjonsteknologier og den stadig økende tilgangen på regnekraft gjør det mulig i dag å utnytte data av en variasjon og et volum som kan berike situasjonsforståelse i fremtiden betydelig. Det springende punkt er hvorvidt vi har beregningsmodeller som evner å omsette data til relevant etterretning i sann tid, og hvorvidt vi har sammenkoblede maskinklynger som tillater at datavolumet øker avhengig av informasjonsbehov og tilgjengelighet.

I denne rapporten studerer vi hva slags systemstøtte og algoritmikk en slik løsning krever. Selv om det ikke er rimelig å anta at alle typer overvåkingsoppgaver og informasjonsbehov kan betjenes med identisk systemstøtte, er arbeidshypotesen her at enkelte overordnede systemegenskaper vil være tilstrekkelig for de fleste tilfeller. Her tenker vi på slike ting som parallelliserbarhet (overvåkingsoppgaven må være av en slik karakter at relevante beregninger kan spres over flere maskiner som arbeider på problemet samtidig), løs sammenkopling (maskiner og kilder må kunne tilkobles uten rekonfigurering og nedetid), skalerbarhet (systemet må kunne vokse med prosessorkraft og minne når det er nødvendig) og feiltoleranse (systemet må tåle at en viss prosent av maskinene feiler eller tas ut av angripere).

Vi benytter en empirisk metode for å undersøke arbeidshypotesen, der vi tar utgangspunkt i en samtidig strøm av AIS-meldinger fra Kystverket, som formidler meldinger fra fartøy innenfor et dekningsområde som omfatter norsk økonomisk sone og vernesonene, og forsterker disse dataene med informasjon fra Fiskeridirektoratets Landings- og sluttseddelregister. Hensikten er å identifisere skip som ikke rapporterer at de fisker, selv om bevegelsesmønsteret og informasjon fra sluttseddelregisteret indikerer det motsatte.

Deteksjon av uregulert fiske allerede er et velstudert problem i forsvarssammenheng. Det som gjør denne rapporten annerledes er sanntidighetsfokuset. Vi ønsker at systemet skal flagge og følge et fartøy så lenge bevegelsene til fartøyet og bakgrunnsinformasjonen om fartøyet gir grunn til det, og ikke lenger. På den måten ønsker vi å kunne overvåke samtlige ca 3500 fartøy som i løpet av en normal virkedag rapporterer på denne AIS-strømmen i sann tid. Teknisk innebærer dette to ting: (1) Vi utvikler en parallelliserbar algoritme vi kaller *tortuosity* (*kurvethet*) for å analysere bevegelsesmønsteret til et skip i sann tid, og (2) vi definerer og konfigurerer en stordatarkitektur designet spesifikt for strømmende data med redundans, skalerbarhet og feiltoleranse bygget inn både i dataflyt og beregninger. Disse to punktene eksemplifiserer de overordnede systemegenskaper som ble nevnt over, og er derfor egnet til å underbygge eller utfordre arbeidshypotesen.

Eksperimentet ble designet for å se i hvilken grad standard programvare kan brukes til å bygge en strømprosesseringsinfrastruktur som oppfyller de identifiserte kravene. Utfallet av eksperimentet var en bekreftelse på at de valgte kjernekomponentene i det vesentlige gir en streaminginfrastruktur med de ønskede egenskapene, hovedsakelig på grunn av egenskapene til kjernekomponenten Apache Kafka.

Contents

Summary 3

Sammendrag 4

1 Introduction 7

2 An infrastructure for stream processing 9
2.1 Requirements 9
2.2 The concept of multimodal stream processing 10

3 Experiment use case: detecting illegal, unreported, and unregulated fishing in real-time 12
3.1 A description of the experiment 13
3.1.1 Analyzing movement patterns 13
3.1.2 Noise reduction with geo-fences 13
3.1.3 Mixing in static background data about the vessel 13
3.2 A disclaimer 14

4 A selection of infrastructure components 15
4.1 Overview of the infrastructure 16
4.1.1 Core components 16
4.1.2 Producers and Consumers 20

5 Dataflow, algorithms and a detected event 24
5.1 Step 1: Data ingestion 24
5.2 Step 2: Trajectory analysis and geo-fencing 25
5.2.1 Analysing the tortuosity of a trajectory 25
5.2.2 Flagging and tracking suspicious vessels 27
5.3 Step 3: Inspecting a detected event 28

6 Conclusion and further work 31

Appendix

References 33

1 Introduction

It is an important part of the Norwegian Armed Forces' activities to monitor Norwegian land areas, the airspace, the sea and the digital space. This surveillance supports both traditional tasks such as defending sovereignty or crisis and conflict management, as well as civil-military tasks such as rescue services and environmental preparedness. The overall response time of the Norwegian Armed Forces, as well as the quality of its operational decisions, depend on the ability to perceive a situation, interpret it and understand it, that is, on the level of situational awareness.

New communication technologies and the ever-increasing availability of computing power today make it possible to utilize data of a variety and volume that can significantly enrich situational awareness in the future.

From a computational point of view, progress in this area depends on whether we have computational models that are able to translate data into relevant real time intelligence, and whether we are able to coordinate machine clusters that, working together, are capable of adapting to potentially very large spikes in the quantity or complexity of available information (complexity being understood as the amount of processing power it takes to convert data into actionable intelligence).

In this report we take a closer look at some general properties such a machine cluster could reasonably be expected to have, as well as the matching characteristics a surveillance algorithm must have in order to exploit it efficiently. We shall often refer, rather loosely, to the conjectured machine cluster + algorithms as 'the system', relying on context to disambiguate.

Although it is not reasonable to assume that all types of surveillance tasks and information needs can be served with identical system support, the working hypothesis in this report is that some general systemic properties will be sufficient for most cases. Examples include, loose coupling, scalability, fault tolerance and parallelizability–properties that will be explained in more detail in Chapter 2.

We do not claim to have proved or refuted this hypothesis (i.e. that some general systemic properties will be sufficient for most cases), but will be content for now to adduce some experimental evidence in support of it.

In other words, the method we adopt is empirical. More specifically, we do an experimental case study of high velocity *stream reasoning* supported by a scalable coordinated machine cluster running a variety of software components and algorithms. The various components and algorithms will be described in more detail as we go. By stream reasoning, we shall mean *the operation of turning continuously incoming data into actionable intelligence in real time.* It is to be distinguished from the related concept of *stream processing*, by which we shall mean the technology and software that relays a continuous data stream.

As data for the experiment, the principal source we use is the live stream of AIS (Automatic Identification System) messages from the Norwegian Coastal Administration. This stream transmits messages from vessels within a coverage area that includes the Norwegian economic zone and the protection zones around Svalbard and Jan Mayen, excepting fishing vessels smaller than 15 meters and recreational vehicles smaller than 45 meters.

AIS is an automatic tracking system that uses transponders on ships and is used by vessel traffic

services. A dynamic or voyage-related AIS message always includes a unique identification of the vessel, its position, course, and speed, and may include equipment information, ship type, current activity, and more. It is usually a relatively information-rich object, that is transmitted every few minutes by every vessel that is under a duty to report, and that as such generates a large enough data stream to test the feasibility of real time stream reasoning for surveillance purposes.

Now, to demonstrate the utility of a loosely coupled architecture that allows new sources to be piped into the data flow at run-time, we amplify the reasoning process with data from the Norwegian Directorate of Fisheries' Landing- and Bill of Lading Register (Fiskeridirektoratets Landings- og sluttseddelregister), as well as with geodata describing no-take zones, that is, marine protected areas that do not allow fishing.

These sources are integrated and exploited conjointly, in order to detect ships that with a certain degree of probability are fishing illegally. The reasoning that is automated in the process goes approximately as follows: if a vessel does not report that it is engaged in fishing, but the movement pattern, its locality and possibly the information from the Landing- and Bill of Lading Register indicates the opposite, then it is flagged as suspect. In addition we set up geo-fences around protected zones, where a geo-fence is understood as a virtual perimeter for a real-world geographic area. An additional alert is triggered if a vessel breaks a perimeter, in which case it is venturing into a no-take zone.

Admittedly, detection of unregulated fishing is already a well-studied problem in defense. What is different about the case study described above its real time focus. Essentially, we want the resulting system to flag and follow a vessels as long as the movement of that vessel and the background information justifies it, and no longer. This is an approach that is designed to use memory and compute cycles sparingly and to enable us to monitor all the approximately 3500 vessels, by our own counting, that report to the stream on a normal working day. Technically, this involves two things:

1. We develop a parallelizable algorithm called tortuosity to analyze the movement pattern of a ship in real time. This algorithm has three notable features:
   (a) it is parallelizable and therefore fast,
   (b) it does not, unlike machine learning algorithms, require training, and therefore
   (c) does not need to draw on historical data which means that it can run in real time.
2. We define and configure a big data architecture designed specifically for streaming data with redundancy, scalability, and fault tolerance built into both data flow and metrics.

These two points exemplify the overall system properties that were listed above and will be further elaborated upon in Chapter 2, and are therefore suitable for assessing the plausibility of the aforementioned working hypothesis.

The report is structured as follows: A set of reasonable requirements for a stream reasoning infrastructure for the military domain is discussed in Chapter 2. The IUU (Illegal, Unreported and Unregulated) fishing case study is described in Chapter 3, while the components of the supporting system are detailed in Chapter 4. Chapter 5 offers a walkthrough of a typical run of the detection algorithm, while Chapter 6 draws some conclusion and outlines possibly lines of future research.

2 An infrastructure for stream processing

2.1 Requirements

In this chapter we extrapolate a set of requirements on a stream reasoning infrastructure from a discussion of some considerations involved in compiling a situational picture. Stated abstractly, these requirements can be summarized by the dimensions of flexibility, reliability, accountability and timeliness.

For concreteness, and in order to vary the theme a bit, imagine surveillance of a battlefield rather than marine traffic. Picture a military analyst who is tasked with monitoring the battlefield in order to assess the most timely and efficient manner of moving medical personnel to injured soldiers or of evacuating soldiers from the field. This task involves keeping an eye on planned evacuation flights in case a projected landing site comes under threat by enemy activity. If it does, then the analyst would be interested in knowing if there are any friendly units nearby that are equipped to counter the threat.

In general, planning of medical evacuation is a complex and important process in military operations. One of the most challenging aspects is making all necessary information available to the decision makers. This is a non-trivial task as different kinds of information, medical information vs. ORBAT information, say, is naturally distributed across different disconnected systems.

On the assumption that this kind of scenario is a typical example of the more general problem of maintaining situational awareness, we extract the following requirements from it:

Timeliness: Intelligence in general is actionable only if it is current or up-to-date. Ensuring the timeliness of information requires the ability to collect, transfer, process, and present the stream of data in real time. Directing friendly units to engage with an adversary that is no longer there is obviously not useful. Thus, as the value of data may deteriorate over time rather rapidly, a stream reasoner needs to perform all the calculations and communication on the fly with the data that has newly arrived, and needs to be able to continue to do so as traffic spikes.

Scalability: It follows from timeliness that a stream reasoner needs to scale automatically if the quantity of available information grows, adding more memory and processing power to the underlying machine-cluster as needed.

Parallelizability: Scalability in turn constrains the kind of algorithms that can be considered suitable for real time data-intensive surveillance tasks: if it is to to produce live intelligence without setting a fixed upper bound on the allowable quantity of incoming data, the relevant algorithms should be parallelizable, meaning that the required calculations are of such a nature that they can be spread across multiple machines working on different parts of the problem simultaneously.

Loose coupling: One reason the quantity of available information could suddenly spike is due to the influx of information from novel sources joining the coordinated cluster of machines that makes up the system. Referring to the example above, this could happen for instance when new coalition partners join the operation, adding their own ISR (Intelligence, Surveillance

and Reconnaissance) to the data flow. A stream reasoner that turns live data into actionable intelligence, should therefore be based on some sort of message bus or publish/subscribe (henceforth pub/sub) architecture that decouples information from software, thus enabling new information consumers and -producers to be added on the fly, that is, without downtime or resource-intensive reconfiguration of parts of or the whole of the machine cluster. [1]

Accountability: actionable intelligence ought to be verifiable and retraceable to the extent possible. It ought to be be possible to give an account in retrospect of the reasoning behind a flagged event and of how it was derived from the information that was available at that snapshot in time. In the scenario described above, the reason for the success or failure of a an evacuation should be open to analysis and evaluation. Applied to the hypothesized live stream reasoning system under description, it follows that his system needs to have a way of recording its distributed state at any given moment in time. Viewed as a pub/sub system this entails, more concretely, that it should be possible to retrieve the status of all incoming and outgoing messages on all topics or channels.

Fault tolerance: fault tolerance is the property that enables a system to continue operating properly in the event of the failure of some of its components. Fault tolerance is particularly important in high-availability or real-time systems since the data in such a system is perishable and cannot necessarily be recovered after a reboot. In a military setting fault-tolerance should also be taken to mean that there is no single point of failure to be targeted by an adversary. As a general rule, fault tolerance is tantamount to redundancy. The machine cluster underlying the hypothesized stream reasoner should have multiple copies of data stored on different machines.

Reliability: the information relayed by the system should be diachronically correct in the sense that the system preserves the temporal ordering of messages. For instance, once they are processed messages should never be interpolated into a more recent substream of data, which, if the stream processor relays a message more than once, can easily happen. It is a very real possibility in a fault tolerant system where there exists multiple copies of every message. Needless to say, distortion of the time sequence can have very adverse effects. Imagine for instance that a friendly unit is reported to have arrived at vacant location when in fact the enemy units got there first. Therefore the stream processor upon which the reasoner is implemented must offer guarantees that every broadcast will happen *at most once*. In addition to this, the relayed information should also be required to be complete *in the specific sense* that the stream processor is able to guarantee that each message is processed *at least once.*

We use these requirements in the next chapter to select a message bus that fits most, if not all of the bill, and we indicate where it deviates.

2.2 The concept of multimodal stream processing

Turning data into actionable intelligence may require a broad range of analytical techniques which will vary from case to case. In the general case, data must be assumed to flow through a data

[1] In a pub/sub system, senders of messages do not program the messages to be sent directly to specific receivers but instead broadcast messages to subscribers, whoever they may be. Similarly, subscribers express interest in one or more topics into which messages are grouped and receive broadcast messages, without needing to know the producers or contact them directly.

processing pipeline in which each section represents an algorithm that adds value to the intelligence product. The key ideas here are stacks or pipelines of computations and the maintenance of real time. By the maintenance of real time we simply mean that if each computation in the pipeline is able to keep up with the flow of data, then the entire pipeline is, and consequently the stream reasoner as such never falls behind real time. A stream processor that is scalable enough to support qualitatively different computations, aka. *modes* of computation, stacked on top of a high velocity data stream without falling behind real time, is what we shall call a *multimodal stream reasoner*. Obviously, multimodality is not an all or nothing affair, but comes in degrees and depends upon the computations applied. This concept is illustrated in Figure 2.1.

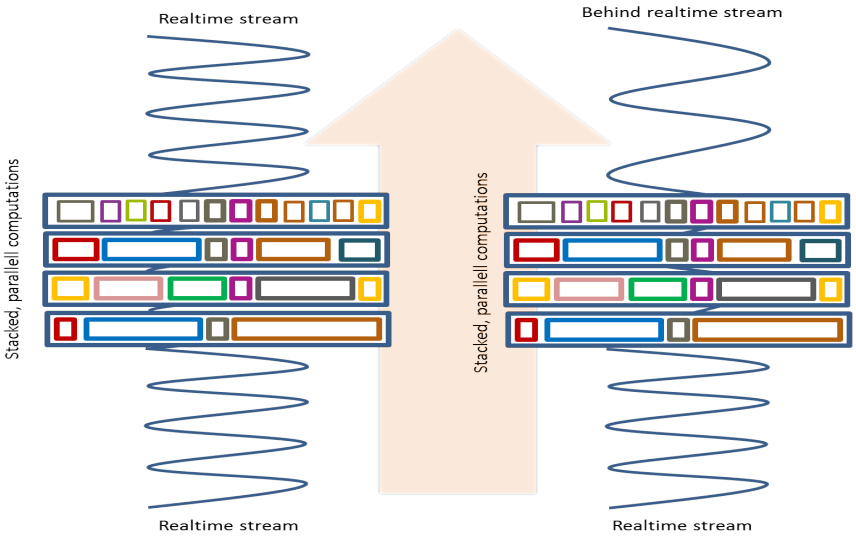


Figure 2.1 Multimodal versus non-multimodal stream reasoning.

Each layer in the respective stack is meant to illustrate a computation, and the partition of it is meant to illustrate the parallel distributed nature of these computations. The velocity of a data stream is indicated by the wavelength of the curve that represents it. A multimodal stream reasoner, then, is one in which the wavelength of the curve is the same when the stream is reassembled after the successive concurrently computed fragments of each layer as it was going in.

3 Experiment use case: detecting illegal, unreported, and unregulated fishing in real-time

Needless to say, a stream reasoning infrastructure needs to be tested and evaluated. In general this may involve anything from inspection of the application runtime environment, measuring average consumption of main memory or calculating the likelihood of network partitions in a particular use case.

For the purposes of the present report, we shall be content with scalability and throughput testing. What we are mainly interested in for now is to find a configuration of software components that scales horizontally, distributes computations, and maintains real time through each qualitatively different computational step that is added to the front of the pipeline.

Clearly, testing cannot be performed in the abstract, we need a test case. For this purpose, we chose to work with the widely recognized problem of detecting illegal, unreported, and unregulated (IUU) fishing from a live stream of position reports, specifically the maritime Automatic Identification System (AIS). The reasons for this fall into two categories: technical convenience and intrinsic interest.

Starting with the latter, there is scientific consensus that IUU fishing significantly undermines the sustainability of the world's oceans.

According to the latest report from the Food and Agriculture Organization of the United Nations (FAO) on the topic, only 59.9 percent of the major commercial fish species that FAO monitors are presently being fished at biologically sustainable levels, while 33.1 percent are being fished at biologically unsustainable levels[2]—a situation that is described as "worrying". In contrast, just 40 years ago, 90 percent of FAO-monitored fisheries were being utilized at biologically sustainable levels, and just 10 percent were being fished unsustainably. Ensuant to this, FAO's Sustainable Development Goal 14, calls on the international community to effectively regulate fish harvesting end overfishing, illegal fishing, and destructive fishing practices, and to implement science-based management plans aimed at restoring stocks (Food and Agriculture Organization of the United Nations 2018).

From a technical convenience point of view, AIS data has the virtue of being easily accessible in real time, well studied, plentiful and algorithmically interesting. For the present project we chose to use the open data stream published by The Norwegian Coastal Administration (Kystverket). This open part of the coastal administrations network of AIS receivers relays messages in real time from all ships of more than 45 meters length within a coverage area of 12 nautical miles from the coast. The data is free of charge and does not require user registration. The user is, however, obligated to comply with the terms stipulated in the Norwegian License for Open Data (NLOD).

As we, at the end of the day, are interested in stream reasoning infrastructures for the military domain, ideally the test data should have been military data. However, as unclassified military data is hard to come by, and as AIS data is used also in military picture compilation, we feel that this fishing use case is sufficient to reach conclusions that can reasonably be expected to also hold for the military domain.

[2]The remaining 7 percent are underfished.

3.1 A description of the experiment

In our experiment the high level goal is to detect IUU fishing by analyzing the movement pattern of a vessel, amplified by link analysis and purpose-defined geo-fences. Link analysis is a data analysis technique from network theory which is used to determine the type of relationships between nodes in a network . In our experiment we use link analysis tools to analyze sales documents from the Directorate of Fisheries' Landing- and Bill of Lading Register. In particular we check whether an identified vessel is listed as a seafood supplier, which is treated as proof that the vessel in question is a fishing vessel.

The overall process then, is one in which the raw AIS data is refined the following three stages:

3.1.1 Analyzing movement patterns

Our analysis of the trajectory of a vessel involves two substeps:

1. first we ingest the binary AIS stream and convert the data on-the-fly to a format that is suitable for spatio-temporal analysis
2. the output goes into a stream reasoning component that classifies movement patterns based on the concept of the *tortuosity* of a trajectory. The tortuosity concept will be further explained in Chapter 5.2.1.

3.1.2 Noise reduction with geo-fences

Tortuosity measures are not a good indicator of the behaviour of a vessel that is too close to land, since seemingly erratic behaviour may simply be due to maneuvering into or out of port. Conversely behaviour that is too regular, such as that typically displayed by ro-ro ferries, may also sometimes trigger a detected event. Both types of events are excluded by drawing a geo-fence that pushes the event detection algorithm sufficiently far away from land.[3]

3.1.3 Mixing in static background data about the vessel

Whilst the trajectory, speed and location of a vessel can yield a good indication of its current activity, the accuracy of such an analysis will not be a hundred percent, and false positives are bound to occur.

In order to zoom in on vessels of interest with more precision, we therefore mix in static background data that enriches the information in the stream. By static data we simply mean data that changes relatively infrequently, and that is externally published or provided by a batch-based system.

In our experiment we use a composite set of data that contains catch data from the Directorate of Fisheries' landing and closing bills register, and vessel data from the Directorate of Fisheries'

[3] In addition to this use of geo-fences to filter out false positives from the event stream, we also use geo-fences to monitor protected areas, though we will not mention it further in this report.

ship register. The stream of detected fishing events is joined in real time with this background information and piped to a *link analysis tool* for the twofold purpose of

1. assessing the regularity and conformity of past behaviour , i.e. *trust*
2. map potentially significant business networks

The working hypothesis is that the sum total of information in this enriched data stream, gives a reasonably good indication of whether a particular vessel warrants a closer look.

3.2 A disclaimer

A disclaimer is probably in order at this point. We do not aim to advance the state of the art in maritime surveillance per se. We are not domain experts, and must defer to those who are to judge the intrinsic interest for that particular purpose of the proposed approach. Rather the experiment is designed to test the capabilities and throughput under high velocity data streams, and in particular, to assess the feasibility of selecting a suite of tools that is capable of maintaining real-time through a multiplicity of refining computations. In other words, the IUU case is meant to serve merely as an indicator for the general utility of the techniques and procedures involved, and, perhaps most importantly, to focus attention on the importance of the specific concept of multimodal stream reasoning, as described in Chapter 2.2, for situational awareness.

4 A selection of infrastructure components

We have described the concept of real time stream reasoning as a computation that translates data into actionable intelligence. We have moreover, stipulated that a sufficiently flexible and extensible stream reasoner will be implemented on top of a distributed stream *processing* system understood as a coordinated machine cluster capable of adapting to potentially very large spikes in the quantity or complexity of available information, and fulfilling the other requirements listed in Chapter 2.1 as well. Finally, we have isolated the matching characteristics a surveillance algorithm should have in order to be able to realize the scalability of the underlying stream processing system, thereby deriving a general infrastructure concept consisting of system architecture + algorithms. In this chapter we make all this more precise by showcasing one concrete implementation of this concept.

The concept of an information infrastructure is not entirely crisp, though, and needs to be handled with a bit of caution as it is used in different contexts to denote different things. These things often differ in detail as well as in scope. According to (Johnson 2012) an "information infrastructure consists of the classifications, standards, protocols, and algorithms that organize information and communication", thus emphasizing software and standards. In (Pironti 2006), in contrast, an information infrastructure is defined as all of the people, processes, procedures, tools, facilities, and technology which supports the creation, use, transport, storage, and destruction of information. Whereas the former limits the scope of the definition to standards and software, the latter includes people and abstract business processes as well.

Although adherence to document standards, file formats and protocols is certainly an integral part of the experimental infrastructure that is described in this chapter, the standards themselves will not be described as such. The following list gives a list of the most important standards that are involved:

- The ISO 6709 standard representation of geographic point location by coordinates.
- The TCP/IP communication protocol.
- The GeoJSON format for representing geographical features.
- The AIS message format for vessel tracking.
- The Resource Description Format (RDF) for representing background information from the Norwegian Directorate of Fisheries' Landing- and Bill of Lading Register.

Putting standards to one side for now, when we use the term 'information infrastructure' in the following, what we have in mind is a suite of software components that together make up a streaming platform. According to the conclusion from (Stolpe et al. 2018), where it is postulated that it is not possible to build an all-purpose big data infrastructure that will have the desired characteristics for all use cases, we further narrow this concept down to selections of software components that jointly fulfill the requirements formulated in Chapter 2. Further, the composition of the components yields a total rate of production that is sufficiently high, and scalable, to maintain real time in the face of realistic increases in workloads. This latter property is epitomized by our proposed concept of multimodal stream reasoning, which denotes a set of qualitatively different computations that are stackable in the sense that no computation in the stack will make the rate of production fall behind real time. Such computations are typically distributed and hence require a big data programming framework.

4.1 Overview of the infrastructure

4.1.1 Core components

The infrastructure essentially consists of a core parallel and distributed message bus surrounded by an arbitrary and variable set of satellite consumers and producers. The consumers and producers interact and communicate through the message bus by way of a mediation application that chains, transforms and routes streams to and from the message bus. A visual representation of this configuration is presented in Figure 4.1.

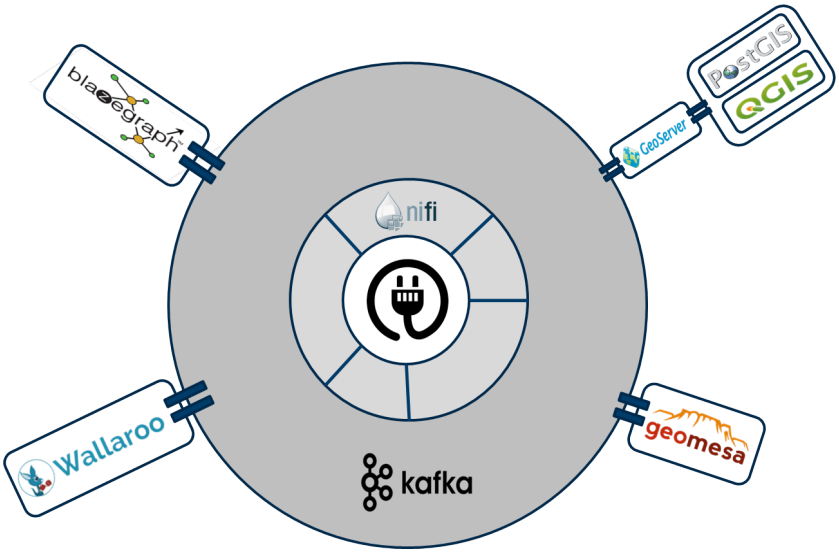


Figure 4.1 The infrastructure in outline.

The individual components in this particular realization of the infrastructure will be explained in due course. It is convenient to make a rough distinction between *core components*, on the one hand, and *producers and consumers* on the other.

The consumers and producers are specialized applications that process data streams for a particular purpose using custom algorithms, e.g. for spatio-temporal indexing or event detection. They output the results of computation back into the core as a refined data stream so that other specialized applications can stack their own computations on top of the already refined stream. These applications do not need to be aware of each other and may come and go during run-time.

The core component's principal responsibility is first of all to ingest and broadcast the raw data streams to the producers and consumers. Other responsibilities include combining, composing, storing and indexing the various streams as well as taking snapshots of the distributed state of the system for rollback and fault-tolerance purposes.

The selected components will be described in the following text, but it is worth notating at this stage that all components that handle dynamic data are in-memory solutions that do not involve reading and writing to disk. This is a key feature in order to ensure the near real time processing required for the analysis task at hand.

Moving on, we now turn to describing the two core components in Figure 4.1, namely Apache Kafka and Apache NiFi.

4.1.1.1 Data distribution with Kafka

Apache Kafka is a software platform originally developed by LinkedIn but donated to the Apache Software Foundation and open sourced in 2011.[4] In a birds-eye perspective, it is a distributed event streaming platform capable of handling high velocity, high throughput data in real time and of scaling up as workloads increase.

On a technical level, Kakfa is essentially a massively scalable pub/sub message queue designed as a distributed transaction log. More specifically, Kafka distributes and stores sequences of messages, aka. streams, that come from arbitrarily many processes called producers. The messages in a stream are strictly ordered by their offsets (the position of a message in a sequence of messages), and indexed and stored together with a timestamp. This makes it possible to replay streams from any point, by setting a cursor to a particular position in the stream. Streams themselves are indexed by topics and external consumer applications connect to Kafka by subscribing to one or more of these topics.

Kafka runs on a cluster of one or more servers (called brokers), and the partitions of all topics are distributed across the cluster nodes. Additionally, partitions are replicated to multiple brokers.

Taking stock, Kafka has three key capabilities:

- publish and subscribe to streams of records in a manner similar to a message queue or enterprise messaging system.
- store arbitrarily large streams of records in a fault-tolerant persistent way
- search inside streams, play from offset, play from beginning
- process streams of records maintaining real time

Kafka is generally used for two broad classes of applications:

- building real time streaming data pipelines that reliably get data between systems or applications
- building real time streaming applications that transform or react to the streams of data based on detected events

A summary of the most interesting properties of Kafka, relative to our requirements from Chapter 2, goes as follows:

Loose coupling: As a pub/sub message bus which offers the topic as the main abstraction to connect, a variable number of producers and consumers can interact flexibly without configuration prior to deployment.

Scalability: The distributed nature of Kafka means that it scales horizontally, i.e. that it avails itself of accessible machine resources, memory and CPUs, when necessary. Its capacity to handle high velocity data streams and to store big data sets is therefore limited only by the

[4] https://en.wikipedia.org/wiki/Apache_Kafka

size of the cluster that is available to it.

Accountability: Depending on the size of the available machine cluster, Kafka is capable of storing, indexing, and effectively managing a petabyte size backlogs of data. It also implements a snapshotting algorithm called the Chandy-Lamport algorithm that is used in distributed systems for recording a consistent global state of an asynchronous system. The former supports traceability and provenance, the second makes it possible to say at any given moment what the system knew at the point in time, thus opening up the basis for decisions made for scrutiny.

Reliability: As with any distributed system, the computers that that make up a distributed publish-subscribe messaging system can fail independently of one another. In the case of Kafka, an individual broker can crash, or a network failure can happen while the producer is sending a message to a topic. The key, when recovering a failed instance, is to resume processing in exactly the same state as before the crash, not missing or duplicating any messages. This is called exactly once semantics and is implemented as an optional configuration in Kafka.[5]

As regards fault tolerance, and in particular tolerance for faults that are due to network partitioning, the situation is less clear.

Kafka was engineered specifically to be used within centralized data centers, where network partitioning rarely happens. Based on this assumption, the design prioritizes consistency and availability over partition tolerance, and as a consequence certain network partitioning conditions can be unrecoverable. Specifically, under certain worst-case settings, network partitioning can result in data loss (see Kingsbury (2013) and Apache Kafka (2019)).

Unlike many other distributed systems, the leader election process (necessitated when a current leader falls out) is not done by a simple majority vote. Rather, only topic replicas that are fully in-sync with the leader (referred to as an in-sync replica, ISR) are candidates. In most situations this is unproblematic, but in the border-case where all ISRs are unreachable, a choice between maintaining consistency versus availability must be taken with regards to how the system should act.

That is, in these situations Kafka defaults to prioritizing consistency by selecting the first reachable ISR as the new topic leader. However, it is possible to instruct Kafka to prioritize availability instead by instructing it to select the first reachable topic replica, which is is not necessarily an ISR, as the new leader. In both cases, however, data loss can occur.

Therefore, though Kafka meets many of our requirements, it is not a perfect match. A plausible alternative would be to use Kafka as a core infrastructure for storage, processing and distribution, and to use some system that implements the MQTT[6] protocol at the edge of the network for IoT devices. The MQTT protocol is a publish-subscribe-based messaging protocol that is specifically designed for connections where a "small code footprint" is required or the network bandwidth is limited. Some popular implementations are:

- RabbitMQ[7]
- Mosquitto[8]

[5] https://www.confluent.io/blog/exactly-once-semantics-are-possible-heres-how-apache-kafka-does-it/
[6] http://mqtt.org/
[7] https://www.rabbitmq.com/
[8] https://mosquitto.org/

- ActiveMQ[9]

All of these are open source and free, and at least the first two are suitable for deployment on servers as well as on low power single board devices such as a Raspberry Pi.

4.1.1.2 Plugging in with NiFi

Apache NiFi is a software project from the Apache Software Foundation designed to automate the flow of data between software systems. It is based on the "NiagaraFiles" software previously developed by the US National Security Agency (NSA). It was open sourced as a part of NSA's technology transfer program in 2014.[10]

NiFi is essentially a visual application development environment managing adapters to different data sources, and for assembling pipelines for data routing, transformation and system mediation logic.

NiFi is designed to operate within machine clusters using TLS encryption for secure communication. It is highly scalable, fault-tolerant and resilient. NiFi also guarantees message delivery even at very high scale, which means that it it fits well with Kafka's exactly once semantics. As NiFi is designed to scale out, or scale horizontally, by spreading its computations over a machine cluster, the rate of throughput is only limited by the number of available machines. Importantly for our purposes, given a sufficiently large cluster of machines NiFi will be able to maintain real time, and thus to support multimodal stream reasoning.

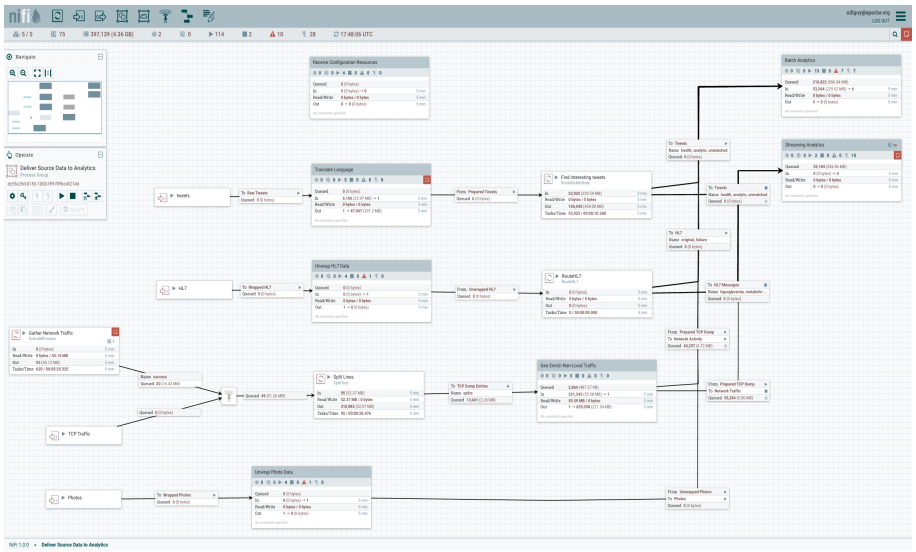


Figure 4.2 Apache NiFi user interface.

In our infrastructure as visualized in Figure 4.1 we are using NiFi as a dock for raw data streams to come in and be preprocessed before they flow into Kafka. These preprocessing steps may be simple or complex, which in the latter case means that the preprocessor in question is a NiFi pipeline

[9] https://activemq.apache.org/
[10] https://en.wikipedia.org/wiki/Apache_NiFi

consisting of more than one step. As an example, in our test case we use NiFi to combine the GPSd binary decoder for AIS messages with GeoMesa-NiFi, a prepackaged adapter that provides spatio-temporal indexing of data objects in kafka streams. In outline the process is as follows: binary AIS data flows continuously into a NiFi processor that decodes the AIS into JSON. These JSON objects are piped into GeoMesa-NiFi which indexes the objects spatio-temporally in the form of geohashes, essentially string encodings based on the contained features (points, geometries, timestamp, etc.) which can be efficiently utilized when performing geospatial operations such as analysis and querying, before the objects are published to a Kakfa stream for consumption by geospatial software at the other end. More about this other end shortly.

4.1.2 Producers and Consumers

In our infrastructure, the producers and consumers, that process data streams for our particular purpose, are loosely coupled in line with the requirements in Chapter 2. In particular, given that any discrepancies or kinks in formatting are ironed out in NiFi, the producers and consumers should not be assumed to be aware of each other. The rationale behind this was explained in the requirements chapter.

4.1.2.1 Event detection in Wallaroo

Our test case, as previously mentioned, is the problem of detecting illegal, unreported, and unregulated fishing from a live stream of AIS reports published by the Norwegian Coastal Administration. In essence this is an event recognition problem, and as such it involves inferring complex events from simpler events that can be derived more immediately from the sensor stream. Specifically for our case, it involves inferring probable fishing behaviour from simpler events such as turning, loitering, speeding up etc.

In order to be consistent with our design goals, an event processor had to allow

1. complex event logic to be expressed succinctly and with ease, and
2. parallel, scalable computation in order to maintain real time.

In our case parallel computation is particularly challenging since compiling vessel trajectories is an inherently *stateful* computation. That means that it is not simply a matter of filtering out from a stream elements that satisfy a condition. Rather events have to be compiled *from memory* using previous reports emitted by the same vessel. However, memory cannot be *local* memory, or we would create another bottleneck and eventually fall behind real time. In other words, we need to parallelise and distribute state.

One system that fits this bill nicely, we found, is Wallaroo Python[11] – succinctness and ease of expression arguably being a feature of Python itself. As a big data processing framework, Wallaroo is designed for writing distributed data processing applications, especially demanding high-throughput and low-latency. What made Wallaroo particularly useful for us, though, is that it is built precisely for distributed state management, i.e. the unit of parallelisation is a state object that retains past

[11] https://www.wallaroolabs.com/

events for as long as its definition demands. We shall explain the importance of this in more detail later.

4.1.2.2 Querying high-velocity, spatio-temporal data using GeoMesa

In many cases, the streaming data will contain spatio-temporal information which one wishes to utilize for searching and filtering. The AIS case presented in this report is very much an example of such. Specifically, we would like to filter on AIS messages within user-specified geospatial regions and within a certain time frame of history.

Filtering on geographic and/or temporal features are expensive, yet often required, operations. Typically, systems have built-in features catering for these two categories of data, based on utilizing index structures specially constructed for the task (e.g. R-trees[12]). Postgres[13], a much used relational database, supports such functionality through the widely used PostGIS extension.

Maintaining spatio-temporal indexes is expensive, and since existing solutions such as PostGIS are made for centralized storage of static data, they do not fit the distributed, streaming data assumption outlined in the above-mentioned case.

One candidate system for supporting spatio-temporal filtering in a streaming, big data setting, is GeoMesa[14]. GeoMesa is a suite of open source tools that provides geospatial indexing, querying and analytics over well-known big data infrastructure components by utilizing an efficient geohashing algorithm. It supports long-term storage on top of distributed databases (e.g. Accumulo, HBase, and Cassandra) as well as real time indexing and processing of streaming spatio-temporal data on top of Apache Kafka.

In our case, GeoMesa was used to ingest decrypted AIS messages in real time (JSON objects), index the data spatio-temporally and hence make the stream queryable. More concretely, GeoMesa stored geohash-indexed data in Kafka streams, where the geohashes can be used used to provide efficient querying based on spatio-temporal features. We set up the NiFi-pipeline to produce two GeoMesa-indexed streams; “Latest”, which maintained an updated list of the last known position of known ships, and “Historic” which maintained a 24-hour history over all AIS messages received. For the latter stream, stale information (AIS messages older than 24 hours) was continuously dropped from the underlying Kafka stream.

As for querying of the GeoMesa-indexed streams, this functionality was exposed using well-known interfaces and formats for geospatial data by utilizing GeoServer as middleware between GeoMesa and the user application QGIS[15]. Specifically, by utilizing a plugable GeoMesa-extension to GeoServer, GeoServer could easily be set up to utilize GeoMesa-indexed Kafka streams directly as geospatial data stores.

[12] https://en.wikipedia.org/wiki/R-tree
[13] https://www.postgresql.org/
[14] http://geomesa.org
[15] http://www.qgis.org

4.1.2.3 *Exposing data through common geo-interfaces using GeoServer*

As noted in the previous chapter, the functionality for querying GeoMesa was provided through GeoServer[16], which is a middleware component for sharing geospatial data. This facilitates the use of common geo standards and interfaces such as WMS, WFS, GeoJSON, etc. for querying and retrieving data, which further allows us to utilize a large range of off-the-shelf systems and libraries for processing and presenting the data to the user.

In addition to exposing the GeoMesa-indexed streams for querying, GeoServer was also used to expose a PostGIS database that contained static coastline data, used for geofencing purposes when identifying illegal fishing. In summary, GeoServer acts as a middleware bridge between systems responsible storing, indexing and querying geospatial data, such as GeoMesa and PostGIS, and user interfaces such as QGIS, through the use of standardized interfaces.

4.1.2.4 *Plots and and track selection in QGIS*

The data stored and processed in the underlying components, including the identification of unreported fishing, must of course be presented to the user in a visual way. More concretely, we needed to represent AIS plots, and identified unreported fishing, in terms of bounding boxes, visually for the user.

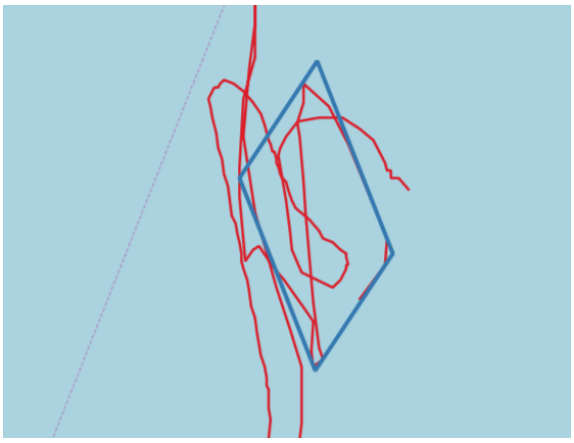


Figure 4.3 Bounding box representing identified unreported fishing.

For this case, we utilize QGIS, an open source desktop GIS application that supports map rendering, plotting and scripting for visual analytics of geospatial information. It can utilize common standards and interfaces for geospatial data, such as maps from Open Geospatial Consortium's Web Map Service (WMS)[17] , features from Web Feature Service (WFS)[18] and GeoJSON, and allows point-and-click filtering on geospatial data provided. Furthermore, QGIS allows writing Python-scripts for extending functionality with user-defined operations. For example, we added functionality for rendering unreported fishing, point-and-click lookup of ship info, as well as fetching and drawing

[16] http://geoserver.org
[17] https://www.opengeospatial.org/standards/wms
[18] https://www.opengeospatial.org/standards/wfs

historic track plots (shown in Figure 4.3).

4.1.2.5 Enriching the data stream with Blazegraph and LodLive

As explained in Chapter 3.1.3, we enrich the event stream generated from the stream of AIS messages with data from the Norwegian Directorate of Fisheries' landing and closing bills register, and vessel message from the Norwegian Directorate of Fisheries' ship register with the purpose of assessing trust and mapping business networks.

There are two components in the infrastructure dedicated to this process and similar ones, namely Blazegraph[19] and LodLive[20].

Blazegraph, is a GPU-accelerated high performance database for large graphs. The database supports multi-tenancy and can be deployed as an embedded database, a standalone server, and as a replication cluster.

An important reason behind our choice of Blazegraph is that it represents data in The Resource Description Framework (RDF), RDF is a family of World Wide Web Consortium (W3C) specifications that is designed as a general method for modeling and linking data across the Web independently of where it is physically stored. Using an RDF store for static mix-ins makes the infrastructure *open-textured* in the sense that further enrichment of the event stream with other kinds of open source data is straightforward, presupposing RDF-formatting.

Lodlive, on the other hand, is the tool we use for visual link analysis of enriched events. This step should be considered merely a proof-of-concept as LodLive is an experimental project at this stage.

[19] https://www.blazegraph.com/
[20] http://en.lodlive.it/

5 Dataflow, algorithms and a detected event

This chapter outlines a run of the system as it is applied to the aforementioned example case of interpreting the AIS stream from the Norwegian Directorate of Fisheries. This following material serves two expositional purposes: first it illustrates the flow of data and messages through the distributed message bus, in our case through Kafka. Second, it exemplifies how qualitatively different algorithms can be layered onto the stream without incurring a cost that makes the output fall behind real time. These algorithms are of three different kinds:

- Algorithms for analysing vessels' movement patterns.
- Geodesic algorithms for drawing geo-fences and calculating distance from shore.
- Algorithms for evaluating queries against static background data for enriching the event stream.

The point of the infrastructure is to have scalability and high throughput in all parts of the system so that qualitatively different algorithms such as these can be stacked on top of each other without slowing down the output.

5.1 Step 1: Data ingestion

The raw data stream is an open AIS-stream as described in Chapter 3. The AIS messages are transmitted at a frequency varying from a few seconds to several minutes, depending on ship type, message type and other things.

Ingestion of the AIS-stream into Kafka is configured in NiFi, which converts the binary AIS messages into JSON objects and splits the incoming stream into two GeoMesa streams each of which is submitted to Kafka.

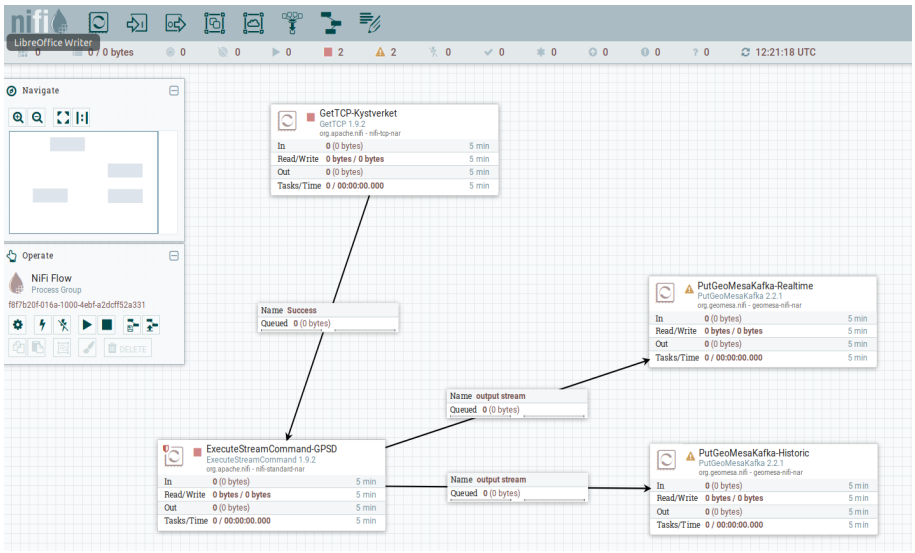


Figure 5.1 A complex preprocessor in NiFi.

Stepping through the process, as illustrated in Figure 5.1, the box labelled GetTCP is a drag-and-drop component in NiFi that connects to an incoming TCP stream, in our case the binary AIS stream. The output from GetTCP is piped to another NiFi processor labelled ExecuteStreamCommand, which converts the binary data to JSON. At this point the stream is split into two by dragging and dropping NiFi's GeoMesa adapter onto the NiFi canvas twice. The NiFi GeoMesa adapter comes prepackaged. It creates GeoMesa objects (features, polygons etc.) from JSON objects, and in our case, outputs them to two different Kafka streams for consumption by geospatial software. The two Kafka streams correspond to the two boxes labelled respectively PutGeoMesaKafka-Historic and PutGeoMesaKafka-Realtime. The streams differ wrt. the way elements in the stream are indexed and stored. The former indexes AIS-messages by the time of their arrival and persists the data into Kafka. This is suitable for analysis of historic data in specific intervals of time, typically trajectory analysis. The second indexes AIS-messages by ship, that is by MMSI number, and is suitable for real-time vessel tracking. Note this dual use of Kafka as both a real-time streaming engine and as a high-volume data storage.

5.2 Step 2: Trajectory analysis and geo-fencing

After the AIS messages have been preprocessed and submitted to Kafka, they are available to all consumers by way of subscription to the corresponding Kafka topic.

The real-time data is consumed by Wallaroo, which is the framework we chose to use for all event detection logic. As explained in Chapter 4.1.2.1 specializes in distributing state and stateful computations.

By our own counting, there are approximately 3500 vessels in the raw AIS stream on a normal working day during normal working hours. Wallaroo creates one state object for each of these, which means that it is in reality administering approximately 3500 separate continuous streams, each tracking one vessel. Wallarroo then runs the event detection logic on each of these simultaneously.

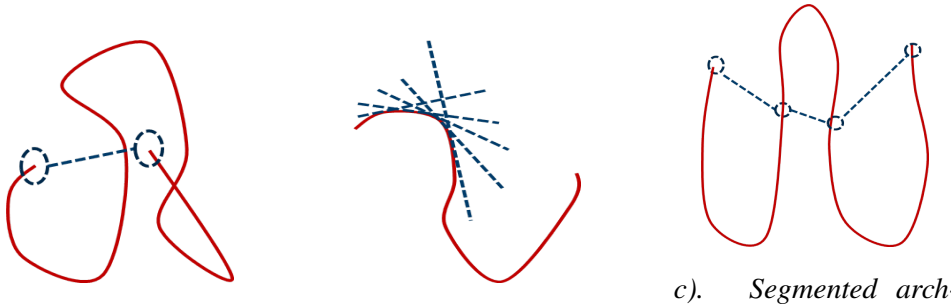
5.2.1 Analysing the tortuosity of a trajectory

Since this case study is focused on throughput and the maintenance of real time, we wanted a fast parallelisable trajectory analysis algorithm that does not require training. This would make for a simple computation that could be distributed and pushed to the relevant data for each individual ship.

We eventually chose to implement various measures of *tortuosity*. In informal terms, tortuosity is simply the property of a curve being twisted or having many turns. Our working hypothesis was simply that as a general rule of thumb, if a vessel is moving at the right speed sufficiently far from shore making many turns over a relatively small area, then it is likely engaged in fishing.

Measures of tortuosity have been formalized independently in several disciplines, for instance in the study of diffusion and flow in fluid mechanics (Epstein 1989) and in optometry (Grisan et al. 2003, Pearson 2003). Studies of typical movement patterns for fishing vessels (e. g. Jiang et al. (2016)) indicate that these measures may be applicable to detection of illegal fishing (Enguehard et al. 2013, Jiang et al. 2016, Storm-Furru 2019). We implemented the following three such

measures, namely *arch-curve ratio*, *average rate of turn* and *segmented arch-curve*, see Figure 5.2.



a) The arch-curve ratio is the ratio of the length of the curve to the distance between the ends of it.

b) The mean rate of turn says how much the bearing changes on average between AIS reports.

c). Segmented arch-curve splits a trajectory into constant turn segments and takes the average of the arch-curve ratio of each.

Figure 5.2 Tortuosity measures.

While tortuosity measures indicate fishing behaviour, there are several sources of false positives that need to be filtered out. For instance a ro-ro ferry crossing back and forth over a fjord will leave a trail with a very high arch-curve ratio although it clearly does not resemble fishing behaviour (see Figure 5.3).

Figure 5.3 Since the distance between the ends of the curve after a back-and-forth crossing is very small, a ro-ro ferry has a high arch-curve ratio.

However, ro-ro ships and other ferries will typically sail at an average speed at approximately 15 knots, which is enough to rule them out, since vessels that are engaged in fishing typically move at much lower speeds.

Some movement patterns cannot be filtered out simply by combining tortuosity and average speed. A typical example is a ship coming into port, see Figure 5.4: as it approaches the port it typically comes in at a relatively high speed and in a straight line. After it has been anchored up, however, speed drops to insignificantly different from zero, whereas the tortuosity rises quickly if the boat is drifting measurably at anchor. These two factors taken together may produce *average* values–the line of approach to the port providing the speed, and the dead drift providing the tortuosity–that are compatible with low speed fishing activity.

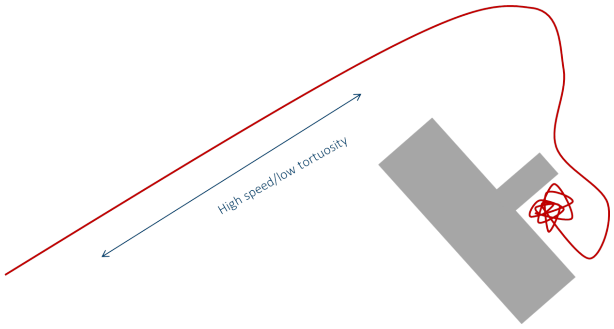


Figure 5.4 A ship coming into port may produce average *values compatible with fishing activity.*

Cutting a long story short, after a good deal of experimentation, we were able to filter out these and a number of other false positives by combining different tortuosity measures with the following kinds of heuristic:

- typical speeds
- typical distances from shore
- continuous motion

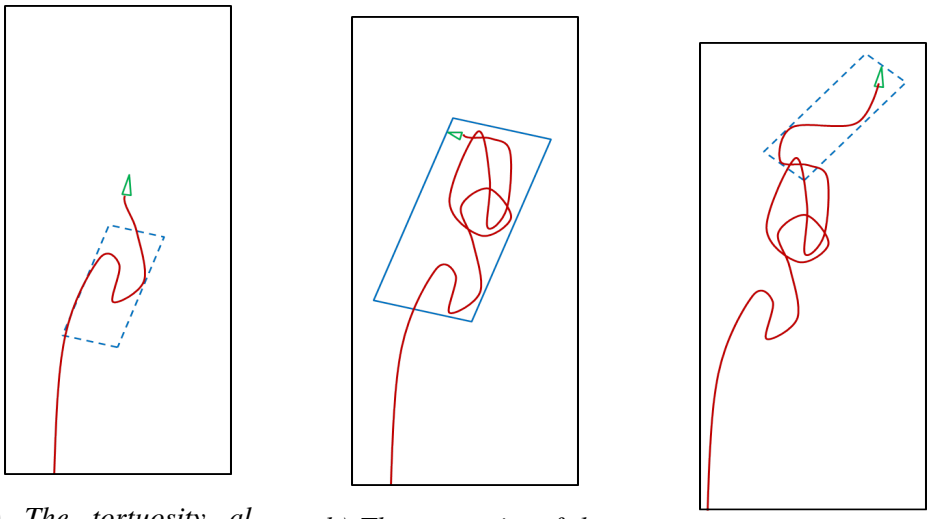
In addition to representing typical distances from shore, we also drew geo-fences around protected marine areas to pick up any activity there.

The resulting flow of data goes as follows: the AIS stream from the Coastal Administration is plugged into Kafka using NiFi to split it into a real-time stream and a persisted history. The real-time stream is consumed by Wallaroo which does live event detection looking for ships that do not report that they are engaged in fishing but act as if they are. Typical speeds are calculated directly by the AIS messages, whereas all distance calculations are performed by querying a PostGIS database containing information about coastlines, through GeoServer.

5.2.2 Flagging and tracking suspicious vessels

The event detection algorithm in Wallaroo uses a sliding window of configurable size to analyse the data. That is, the algorithm has a queue of a specified length and pops off and appends tracks one by one. Each time a track is appended the analysis is run again. This window thus represents the attention span of the algorithm and the extent of its recollection. It is possible to reconnect a vessel to its history though, by using the history persisted in Kafka.

When a likely fishing event is detected, Wallaroo draws a minimum bounding rectangle around the contents of its attention window and publishes the resulting box as a GeoJSON object on a separate Kafka stream. These boxes will typically come in sequences as a vessel usually does not change behaviour abruptly. Stated differently, the event detection algorithm starts outputting bounding rectangles for a vessel as long as the vessel's behaviour indicates that it is fishing. As its behaviour changes, however, the event detection algorithm loses interest and stops sending rectangles. This is illustrated in Figure 5.5.



a) The tortuosity algorithm detects zigzagging within its window of attention.

b) The tortuosity of the trajectory, distance from shore, speed etc. indicate probable fishing behaviour.

c) As straight lines start to dominate window of attention, the algorithm 'loses interest'.

Figure 5.5 Progression of the tortuosity algorithm.

The route the data takes from here is somewhat complicated, but uses only components we have already introduced and explained: the resulting stream of rectangles is published as a separate publicly accessible Kafka stream. As we are using NiFi to mediate between systems, these rectangles are read by a standard generic Kafka listener in the set of pre built NiFi adapters before they are piped to the NiFi-native GeoMesa-Kafka adapter that serializes spatio-temporal features in a searchable, GeoMesa-indexed Kafka stream.

Since the persisted history of AIS tracks that is stored as soon as the AIS stream comes in is indexed in the same way, these rectangles can efficiently be combined with stored trajectories via the GeoServer interface.

In our experiment we exploited this opportunity by having GeoServer generate plottable Web Feature Service (WFS) layers on-the-fly, based on spatio-temporal queries from the client system. In this way, the system produced auto-refreshing layers for real-time ship positions as well as for detected events (i.e. stream of rectangles). These layers are then visually rendered in QGIS by superimposing onto Web Map Service (WMS) maps.

5.3 Step 3: Inspecting a detected event

When a vessel behaves in a manner that indicates that it is fishing, sufficiently far from shore, and at the right speed, it is flagged by the event detection algorithm and a minimum bounding rectangle encasing the triggering trajectory is published via Kafka. After the processing steps described in the previous subchapter this rectangle eventually finds its way to a map layer in QGIS which is updated continually in order to maintain real time. Figure 5.6 shows how this rectangle is displayed in QGIS

to indicate which segment of a vessel trajectory it is that is flagged as indicative of fishing:

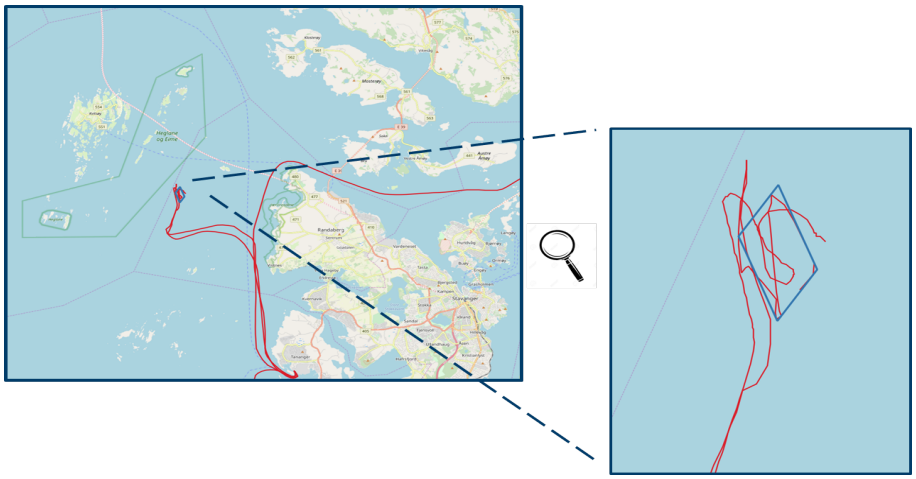


Figure 5.6 A detected event in QGIS.

The QGIS display extracts an AIS track for that vessel that day from the persisted history of AIS tracks in Kafka and combines it with the real-time stream of minimal bounding boxes for the detected event.

In this case a ship is detected very close to the Heglane and Eime nature reserve. It does not report that it is engaged in fishing, but its movement pattern, speed and location indicates that it is.

In order to be able to acquire more information about a ship we added functionality to QGIS that allows the user to interface seamlessly with maritime open-source intelligence online such as MarineTraffic which is a ship tracking and maritime intelligence provider that allows you to track the movements of any ship in the world.

This online information, see Figure 5.7, reveals that the vessel in question is indeed a fishing boat.

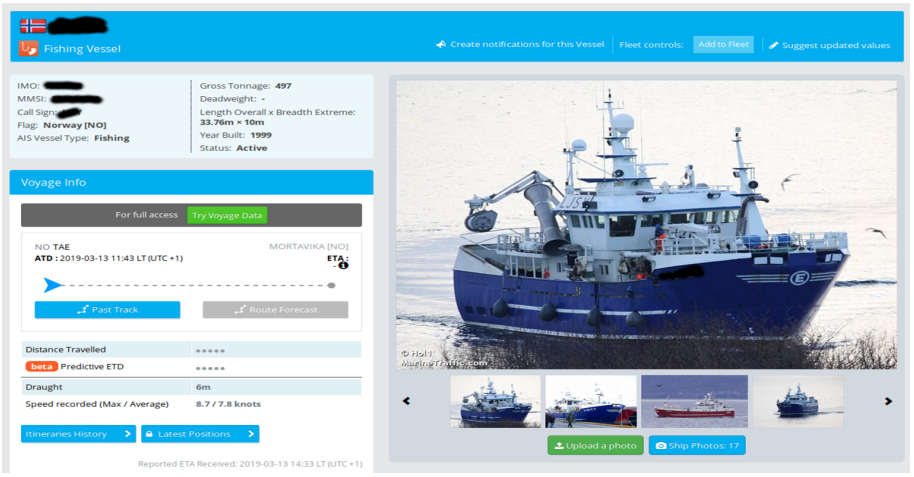


Figure 5.7 The detected vessel (screenshot from `http://marinetraffic.com`).

At this point the user may choose to explore the history and business ties of this vessel further by

enriching the event with data combined from the Norwegian Directorate of Fisheries' landing and closing bills register and its ship register. As with the online lookup, this too is provided to the user as a UI feature triggering a query against the Blazegraph database.

The query result is then displayed in LodLive, which is a visual browser that allows a graph to be traversed and extended by pointing and clicking. Each click triggers a new query against Blazegraph adding more information to the analysis. The reader should note that, since Blazegraph is a parallel distributed and cloud native graph database, it is capable of handling a huge number of such queries in parallel. Since there is nothing in our setup that requires a visual display of the linked data or a human in the loop, this opens the possibility of automating the detection of well known patterns of interest.

The initial knowledge graph that is returned from Blazegraph and displayed in LodLive is presented in Figure 5.8. It gives you among other things the organization number from the Brønnøysund Register identifying the firm behind the fishing boat in question, a short history of previously landed catches and the names and nationality of buyers. What would constitute sufficient grounds for trust or mistrust we must, however, leave to the domain experts.

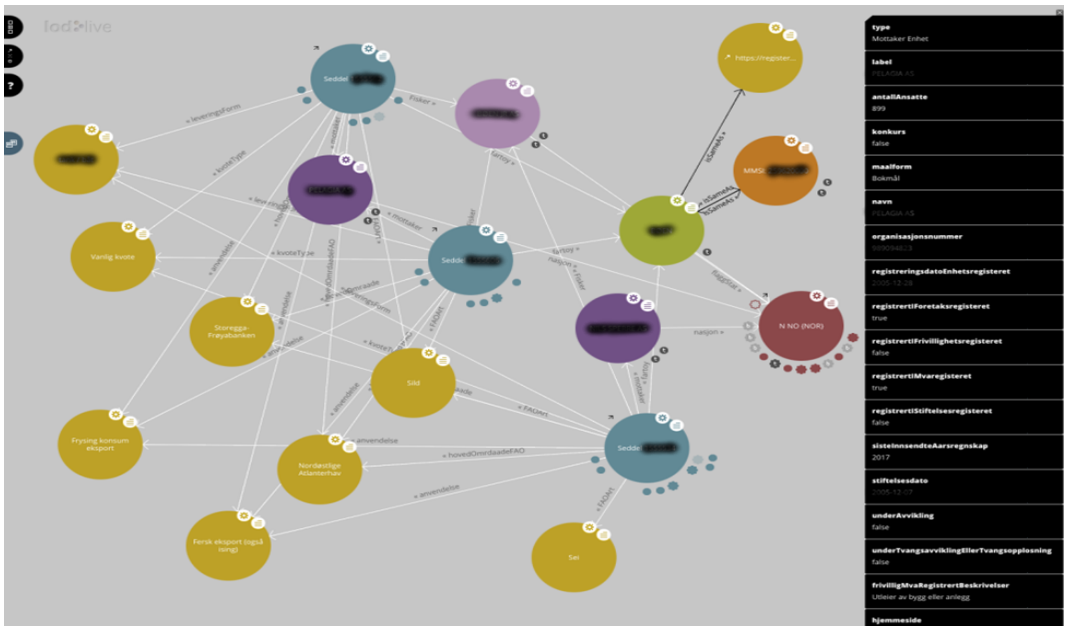


Figure 5.8 Link analysis of the ship.

6 Conclusion and further work

It is an important part of the Norwegian Armed Forces' activities to monitor Norwegian land areas, the airspace, the sea and the digital space. This surveillance supports both traditional tasks such as defending sovereignty or crisis and conflict management, as well as civil-military tasks such as rescue services and environmental preparedness. The overall response time of the Norwegian Armed Forces, as well as the quality of its operational decisions, depend on the ability to perceive a situation, interpret it and understand it, that is, on the level of situational awareness.

New communication technologies and the ever-increasing availability of computing power today make it possible to utilize data of a variety and volume that can significantly enrich situational awareness in the future.

From a computational point of view, progress in this area depends on whether we have computational models that are able to translate data into relevant real time intelligence, and whether we are able to coordinate machine clusters that, working together, are capable of adapting to potentially very large spikes in the quantity or complexity of available information (complexity being understood as the amount of processing power it takes to convert data into actionable intelligence).

In this report we have taken a closer look at some general properties such a machine cluster could reasonably be expected to have, as well as the matching characteristics a surveillance algorithm must have in order to exploit it efficiently. We have conceived of this as *stream reasoning* system or -infrastructure, by which we mean a system that turns data into actionable intelligence in real time.

We have identified a set of requirements that such a stream reasoning infrastructure should satisfy in order to be suitable for the military domain:

Timeliness: Ensuring the timeliness of information requires the ability to collect, transfer, process, and present information in real time. Since the value of data may deteriorate over time rather rapidly, a stream reasoner needs to perform all the calculations and communication on the fly with the data that has newly arrived, and needs to be able to continue to do so as traffic spikes.

Scalability: A stream reasoner needs to scale automatically if the quantity of available information grows, adding more memory and processing power to the underlying machine-cluster as needed.

Parallelizability: If it is to to produce live intelligence without setting a fixed upper bound on the allowable quantity of incoming data, the relevant algorithms should be parallelizable, meaning that the required calculations are of such a nature that they can be spread across multiple machines working on different parts of the problem simultaneously.

Loose coupling: A stream reasoner that turns live data into actionable intelligence, should be based on a publish/subscribe architecture that decouples information from software, thus enabling new information consumers and -producers to be added on the fly.

Accountability: Actionable intelligence should be verifiable and retraceable. It ought to be be possible to give an account in retrospect of the reasoning behind a flagged event and of how it was derived from the information that was available at that snapshot in time.

Fault tolerance: There should not be a single point of failure in the system that could fail or be targeted by an adversary. The machine cluster underlying the stream reasoner should therefore

have redundancy built in, e. g. by having multiple copies of data stored on different machines.

Reliability: The system should preserve the temporal ordering of messages. Older messages should never be interpolated into more recent data, which is to say that each message should be processed *at most once*. Moreover, a stream processor should guarantee that each message is processed *at least once*. Taken together, this two requirements yield an *exactly once* constraint on message passing.

Most of these requirements are met by the Apache Kafka stream processor that figures prominently in our case study. Taking stock

- *Loose coupling* is provided by Kafka's pub/sub mechanism, which enables producers and consumers of information to come and go while Kafka is running.
- Kafka provides *scalability* out of the box, as it can be configured to make scatter data and computations over arbitrarily large machine cluster. In our case study only storage up to the size of hundreds of gigabytes was necessary, thus petabyte level storage, while possible in Kafka, was not tested. Another scalability aspect worth mentioning from the experiment, is the ability of the infrastructure to provide near real-time analysis of streaming data based solely on in-memory data.
- Kafka easily handles support for post hoc analysis of *accountability*, as all data shared through its pub/sub mechanism (in this instance this comprised all data sharing) is available in stored logs.
- Kafka provides *reliability* in terms of correct information being available in the infrastructure as it provides guarantees that messages is processed *exactly once*.

The requirement of parallelizability is absent from this list since it is a feature, not of the stream processor per se, but of the stream *reasoner* and the particular surveillance algorithm employed. Our case study demonstrates how any stream reasoning engine that parallelizes state, we chose to use the Wallaroo framework for Python, may be used to perform interesting cases of event detection running entirely in main memory (and therefore in real time). To show this we developed an algorithm for recognizing fishing behaviour among ships that has three features:

- It is parallelizable and therefore fast.
- It does not, unlike machine learning algorithms, require training.
- It runs entirely in main memory and therefore in real time.

Notably absent from the aforementioned checklist is also the requirement of *fault tolerance*, for which the situation is less clear. Even though Kafka duplicates data over the underlying machine cluster, its mechanisms for recovering from network partitions does not preclude data loss entirely. This problem is connected with a point known as leader election in the underlying consensus protocol that Kafka uses to coordinate different machines in the cluster. [21] It is to be expected that Kafka would be too brittle in this respect for a tactical network, where the likelihood of network partitioning is non-negligible, although this has not been tested.

[21] https://en.wikipedia.org/wiki/Raft_(computer_science)

References

Apache Kafka (2019), 'Apache Kafka 2.2 documentation'.
URL: *https://kafka.apache.org/documentation/*

Enguehard, R., Hoeber, O. & Devillers, R. (2013), 'Interactive exploration of movement data: A case study of geovisual analytics for fishing vessel analysis', *Information Visualization* **12**, 65 – 84.

Epstein, N. (1989), 'On tortuosity and the tortuosity factor in flow and diffusion through porous media', *Chemical Engineering Science* **44**(3), 777 – 779.

Food and Agriculture Organization of the United Nations (2018), *The State of World Fisheries and Aquaculture 2018*.
URL: *https://www.un-ilibrary.org/content/publication/8d6ea4b6-en*

Grisan, E., Foracchia, M. & Ruggeri, A. (2003), A novel method for the automatic evaluation of retinal vessel tortuosity, *in* 'Proceedings of the 25th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (IEEE Cat. No.03CH37439)', Vol. 1, pp. 866–869 Vol.1.

Jiang, X., Silver, D. L., Hu, B., de Souza, E. N. & Matwin, S. (2016), Fishing activity detection from ais data using autoencoders, *in* R. Khoury & C. Drummond, eds, 'Advances in Artificial Intelligence', Springer International Publishing, Cham, pp. 33–39.

Johnson, N. R. (2012), 'Information infrastructure as rhetoric: Tools for analysis', *Poroi* **1**(18).

Kingsbury, K. (2013), 'Jepsen: Kafka'.
URL: *https://aphyr.com/posts/293-jepsen-kafka*

Pearson, R. M. (2003), Optometric grading scales for use in everyday practice clinical.

Pironti, J. P. (2006), 'Key elements of a threat and vulnerability management program', *Information systems control journal : the magazine for IT governance professionals* **3**, 52–56.

Stolpe, A., Hansen, B. J. & Halvorsen, J. (2018), Stordatasystemer og deres egenskaper, FFI-rapport 18/01676, Norwegian Defence Research Establishment (FFI). (In Norwegian).

Storm-Furru, S. (2019), Visual Analytics for Fishing Vessel Operations, Master's thesis, University of Bergen, Norway.

About FFI

The Norwegian Defence Research Establishment (FFI) was founded 11th of April 1946. It is organised as an administrative agency subordinate to the Ministry of Defence.

FFI's MISSION

FFI is the prime institution responsible for defence related research in Norway. Its principal mission is to carry out research and development to meet the requirements of the Armed Forces. FFI has the role of chief adviser to the political and military leadership. In particular, the institute shall focus on aspects of the development in science and technology that can influence our security policy or defence planning.

FFI's VISION

FFI turns knowledge and ideas into an efficient defence.

FFI's CHARACTERISTICS

Creative, daring, broad-minded and responsible.

Om FFI

Forsvarets forskningsinstitutt ble etablert 11. april 1946. Instituttet er organisert som et forvaltningsorgan med særskilte fullmakter underlagt Forsvarsdepartementet.

FFIs FORMÅL

Forsvarets forskningsinstitutt er Forsvarets sentrale forskningsinstitusjon og har som formål å drive forskning og utvikling for Forsvarets behov. Videre er FFI rådgiver overfor Forsvarets strategiske ledelse. Spesielt skal instituttet følge opp trekk ved vitenskapelig og militærteknisk utvikling som kan påvirke forutsetningene for sikkerhetspolitikken eller forsvarsplanleggingen.

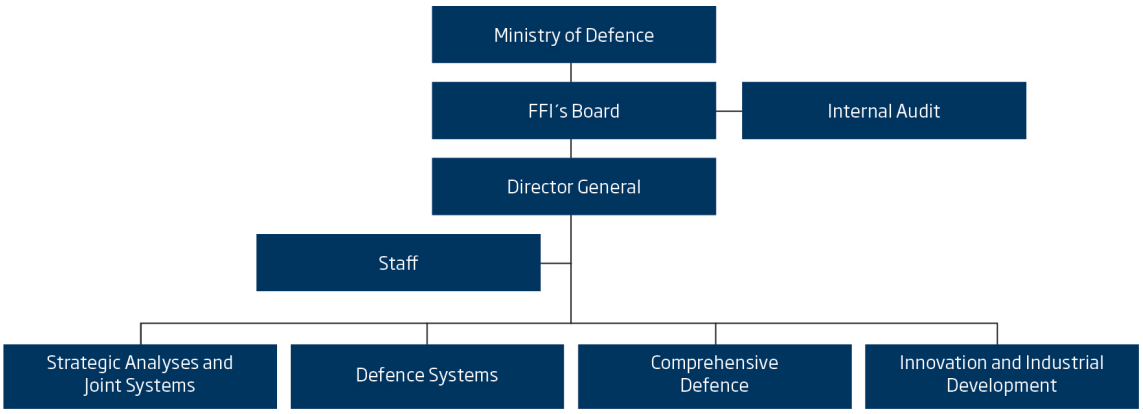
FFIs VISJON

FFI gjør kunnskap og ideer til et effektivt forsvar.

FFIs VERDIER

Skapende, drivende, vidsynt og ansvarlig.

FFI's organisation



Forsvarets forskningsinstitutt
Postboks 25
2027 Kjeller

Besøksadresse:
Instituttveien 20
2007 Kjeller

Telefon: 63 80 70 00
Telefaks: 63 80 71 15
Epost: ffi@ffi.no

Norwegian Defence Research Establishment (FFI)
P.O. Box 25
NO-2027 Kjeller

Office address:
Instituttveien 20
N-2007 Kjeller

Telephone: +47 63 80 70 00
Telefax: +47 63 80 71 15
Email: ffi@ffi.no

FFI Forsvarets forskningsinstitutt
Norwegian Defence Research Establishment