Energy efficient in-memory computing to enable decentralised service workflow composition in support of multi-domain operations

Graham Bent[a], Christopher Simpkin[a], Ian Taylor[a], Abbas Rahimi[b], Geethan Karunaratne[b], Abu Sebastian[b], Declan Millar[c], Andreas Martens[c], and Kaushik Roy[d]

[a]Crime and Security Research Institute, Cardiff University, Cardiff, UK
[b]IBM Research Europe, Zurich, Switzerland.
[c]IBM Research Europe, Hursley, UK
[d]School of Electrical and Computer Engineering, Purdue University, IN, USA

ABSTRACT

Future Multi-Domain Operations (MDO) will require the coordination of hundreds—even thousands—of devices and component services. This will demand the capability to rapidly discover the distributed devices/services and combine them into different workflow configurations, thereby creating the applications necessary to support changing mission needs. To meet these objectives, we envision a distributed Cognitive Computing System (CCS) that consists of humans and software that work together as a 'Distributed Federated Brain'. Motivated by neuromorphic processing models, we present an approach that uses hyperdimensional symbolic semantic vector representations of the services/devices and workflows. We show how these can be used to perform decentralized service/device discovery and workflow composition in the context of a dynamic communications re-planning scenario. In this paper, we describe how emerging analogue AI 'In Memory' and 'Near Memory' computing can be used to efficiently perform some of the required hyperdimensional vector computation (HDC). We present an evaluation of the performance of an energy-efficient phase change memory device (PCM) that can perform the required vector operations and discuss how such devices could be used in energy-critical 'edge of network' tactical MDO operations.

Keywords: Analogue AI, In Memory Computing, services, workflow, Hyperdimensional Computing, Semantic Vectors

1. INTRODUCTION

Future Multi-Domain Operations (MDO) will require the coordination of hundreds—even thousands—of devices and component services. This will demand the capability to rapidly discover the distributed devices/services and combine them into different workflow configurations, thereby creating the applications necessary to support changing mission needs. To meet these objectives, we envision a distributed Cognitive Computing System (CCS) that consists of humans and software that work together as a 'Distributed Federated Brain'.[1] Motivated by neuromorphic processing models, we present an approach that uses hyper-dimensional symbolic semantic vector representations of the services/devices and workflows. We show how these can be used to perform dynamic decentralized service/device discovery and workflow composition.

To illustrate how a 'distributed federated brain' concept might be realized, in this paper we present an example based on a dynamic communications re-planning scenario. In the scenario a commander requires a number of distributed assets and services, initially on different radio channels, to be discovered and to switch to a single communications channel for the duration of the mission. The approach used is particularly applicable to a future military 'Internet of Battlefield Things' (IoBT) where rapid reconfiguration of assets is required to support changing mission needs. In the context of MDO operations, the sensors and services will have been developed and owned by different MDO partners and standard centralized approaches, using formal ontology's

to facilitate service definition and service matching for configuring applications are unlikely to work. The use of semantic vector representations to describe the sensors and services offers an alternative paradigm and in [2–4] it has been shown that service definition, service matching and decentralized service composition can be achieved using an approach based on Vector Symbolic Architectures (VSAs) and that the approach offers significant advantages in environments where communication is limited or unreliable.

In the context of this paper, we argue that in addition to supporting decentralized service composition, that symbolic vector representations can be used to address a number of the key interoperability issues. The first issue relates to the communication of information between MDO partners in a way that minimizes communications bandwidth. We show that using symbolic vectors enables a highly compact semantic representation of the information can be generated and that this can be interpreted by the receiving entities, even if they do not have an exactly matching representation. A second issue relates to the complex challenge of how to perform coordinated missions that require multiple interrelated sub-tasks to be performed by coalition partners, where the partners can perform the required sub-tasks but might do so in a slightly different way to the way their own forces would perform the same task. Thirdly we address the issue of how to perform the required hyperdimensional vector computation (HDC) computations in an energy efficient way by recognising that many of the required vector operations can be performed in a highly parallel fashion using 'In-Memory' and 'Near-Memory' processing devices. We show that using phase change memory memory devices (PCM) how the required vector operations can be performed and how such devices can play a important role in future energy-critical 'edge of network' operations.

The paper is structured as follows: In Section 2 we present a brief background to the use of hyper-dimensional semantic vectors in the context of artificial intelligence and brain inspired computation and we describe why such vectors are particularly well suited to computing paradigms such as in-memory computing; In Section 3 we explain the mathematical concepts that underpin our VSA representation of services and workflows; In Section 4 we describe how the key vector operations can be implemented as an associative memory using a PCM crossbar and, from experimental results, show the typical energy efficiency savings that can be achieved when compared with an alternative CMOS implementations; In Section 5 we present an example scenario based on a dynamic communications re-planning task; In Section 6 we illustrate how vector representations of devices/services and workflows are constructed; In Section 7 we present the logical architecture of a service that implements the VSA operations required and show how a PCM type device can be integrated into the architecture; In Section 8 we present the results obtained from a simulation of the communications re-planning scenario implemented in a representative tactical communications environment based on the NATO Anglova tactical military scenario;[5] Finally in Section 9 we present our conclusions and plans for the future direction of this work.

2. RELATED WORK

VSAs are a family of bio-inspired methods for representing and manipulating concepts and their meanings in a high-dimensional vector space.[6] They are a form of 'brain like' distributed representation that enables large volumes of data to be compressed into a fixed size feature vector such that the semantic meaning of the data and relationships that they represent is preserved. Such vector representations were originally proposed by Hinton[7] who identified that they have recursive binding properties that allow for higher level semantic vector representations to be formulated from, and in the same format as, their lower level semantic vector components. Eliasmith, in his book 'How to Build a Brain',[8] shows how these vector representations can be used to perform 'brain like' neuromorphic cognitive processing. Eliasmith coined the phrase 'semantic pointer' for such a vector since it acts as both a 'semantic' description of the concept, which can be manipulated directly and a 'pointer' to the concept. As such they are said to be semantically self-describing. VSAs are also capable of supporting a large range of cognitive tasks such as: Semantic composition and matching; Representing meaning and order [9,10] ; Analogical mapping[11] ; and Logical reasoning [9,12,13] . Consequentially they have been used in natural language processing[9] and cognitive modelling[8] . One of the important properties of these semantic vector representations is that the vectors can be recursively combined into new vectors, that then represent higher level semantic concepts and that these vectors can in turn be can be unbound to recover the constituent vectors from which they composed.

In [2–4] some of the current authors have shown how devices and software services can be represented as semantic vectors and how these vectors can be combined into vectors that represent service workflows. By exchanging and recursively unbinding the workflow vector it is possible, in a distributed setting, to discover the required components and to construct the required workflow with no centralized control. We refer to this type of application construction as 'Instinctive Analytics' in which the user specifies the workflow required and the appropriate services connect themselves together to perform the required task. The advantage of using symbolic vectors to perform these operations is that the vectors semantically represent the required services and therefore alternative service compositions can be generated. Whilst the work to date has focused on service composition for distributed analytics the concept of distributed workflow is not limited to this relatively narrow domain of service composition and has much wider implications. In the context of military operations we have argued that both information sharing and command and control can be represented as types of workflow .[14] In this paper we present an example command and control application that performs dynamic communications re-planning in the context of a typical MDO tactical command and in formations system (TacCIS) environment.

As discussed in Section 3 our implementation of a VSA requires the manipulation of large hyperdimensional binary vectors which, following Kanerva ,[6] we refer to as hyperdimentional computing (HDC). The inherent robustness of the binary vector representation makes HDC particularly well suited for emerging computing paradigms such as in-memory computing and specifically computational memory based on emerging nanoscale resistive memory or memristive devices .[15–19] In one such work, 3D vertical resistive random access memory (ReRAM) device was used to perform individual operations for HDC .[20,21] In another work, a carbon nanotube field effect transistor-based logic layer was integrated to ReRAMs, improving efficiency further .[22] However, these prototypes have been limited in multiple aspects: a small 32-bit datapath that demands heavy time-multiplexing of hypervectors ;[22] they can store only a fraction of HDC models due to availability of only 256 ReRAM cells ;[20] and they do not allow any reprogrammability as they are restricted to one application ,[20] or a binary classification task .[22] Some of the current authors have shown that since the main computations of the HDC algorithm are performed in-memory with logical and dot product operations it is possible to perform the required operations on memristive devices. Due to the inherent robustness of HDC to errors, it is possible to approximate the mathematical operations associated with HDC to make it suitable for hardware implementation, and to use analog in-memory computing without significantly degrading the output accuracy. Hardware/software experiments using a prototype PCM chip delivered accuracies comparable to software baselines on language and news classification benchmarks with 10;000-dimensional hypervectors, making the work the largest experimental demonstration of HDC with memristive hardware to date .[23] In Section 4 we present relevant details from this work and show in Section 7 how it can be used in the context of our VSA architecture approach to distributed service discovery and workflow orchestration.

3. VECTOR SYMBOLIC ARCHITECTURE

VSAs use hyper-dimensional vector spaces in which the vectors can be real-valued, such as in Plate's Holographic Reduced Representations (HRR),[24] typically having N dimensions ($512 \leq N < 2048$), or they can be large binary vectors, such as Pentti Kanerva's Binary Spatter Codes (BSC),[25] typically having $N \geq 10,000$. For the work here, we have chosen to use Kanerva's BSC large binary vectors, but we note that most of the equations and operations discussed should also be compatible with HRRs.[26]

Typically, when using BSC, a basic set of symbols (e.g., an alphabet) are each assigned a fixed, randomly generated hyper-dimensional binary vector. Due to the high dimensionality of the vectors the basic symbol vectors are uncorrelated to each other with a very high probability. Hence, they are said to be *atomic* vector symbols.[6] Vector *superposition* is then used to build new vectors that represent higher level concepts (e.g., words) and these vectors in turn can be used to recursively build still higher level concepts (e.g., sentences, paragraphs, chapters...). These higher level concept vectors can be compared for similarity using a suitable distance measure such as normalized Hamming Distance (*HDist*).

HDist is defined as the number of bit positions in which two vectors differ, divided by the dimension N of the vector. When setting bits randomly, the probability of any particular bit being set to a 1 or 0 is 0.5; hence, when generating very large random vectors, the result will be, approximately, an equal, 50/50, split of 1s and 0s distributed in a random pattern across the vector. Thus, when comparing any two such randomly generated

vectors, the expected *HDist* will be $HDist \approx 0.5$. Indeed, for 10kbit binary vectors, the probability of two randomly generated vectors having a *HDist* closer than 0.47 (i.e., differing in only 4700 bit positions instead of approximately 5000) is less than 1 in 10^9 [6, page 143]. For the same reason; *atomic* vectors can be generated as needed, on the fly, without fear that the newly generated random vector will be mathematically similar to any existing vector in the vector space. Further, by implication, when using *HDist* on BSC to test for similarity, a threshold of 0.47 or lower implies a match has been detected with a probability of error $\leq 10^{-9}$. A threshold of 0.476 or lower implies a match with a probability of error of $\leq 10^{-6}$. Thus, in our experiments, we used 0.47 as the threshold.

VSAs employ two operations—i.e., *binding* and *superposition*. *Binding* is used to build *role-filler* pairs which allow sub-feature vectors to remain separate and identifiable (although hidden) when bundled into a compound vector via *superposition*. For BSCs, *binding* is a lossless operation, while *superposition*,also referred to as *bundling* or *chunking*, is a lossy process. Kleyko [10, Paper B, page 80] supplies a mathematical analysis of the capacity of a single compound vector such that it can be reliably unbound, i.e., its sub-feature vectors can be reliably detected within the compound vector. This analysis shows that for 10kbit binary vectors the upper limit of superposition is 89 sub-vectors for reliable unbinding. This capacity is sufficient for small workflows with simple service descriptions, but to encode large workflows with more complex service descriptions we require a method for combining more vectors into a single vector whilst maintaining the semantic matching properties. In this paper we use the terms bundling and chunking interchangeably.

Bundling therefore is the recursive method that combines groups of vectors into a single compound vector. The resultant vectors have the same dimension (N) as the original vectors and can be used as the basis for further bundling operations, recursively producing a hierarchical tree structure as shown in Figure 1. Bundling proceeds from the bottom of the hierarchy so that each node in the tree is a compound vector encapsulating the child nodes from the level(s) below. Various methods of recursive *bundling* have been described.[6,10,24,26] However, such methods suffer from limitations when employed for multilevel recursion: some lose their semantic matching ability even if only a single term differs, others cannot maintain separation of sub-features for higher level compound vectors when lower level chunks contain the same vectors [6, page 148] [24, pages 61, 72, 74–para2] [10, Encoding Sequences, page 14]. In order to overcome these issues we use a hierarchical binding scheme first described by Simpkin et al,[2] that uses two methods for permuting binary vectors. The resulting recursive encoding scheme generates vectors at different levels of the hierarchy that capture the semantic information from the levels below.

Recchia and Kanerva point out that for large random vectors, any mapping that permutes the elements can be used as a binding operator, including cyclic-shift.[27] This is because if a high-dimensional vector has no pattern to the distribution of its element values then one or more cyclic rotations will produce a completely different vector as compare to the original value and hence cyclic shift has the same effect of applying a random permutation, i.e., since there was no pattern in the element distribution to start with there will remain no pattern to the element distribution after rotation.

The encoding scheme is described in Equation 1 and employs a combination of both XOR and cyclic-shift binding operators to enable a recursive binding scheme that is capable of encoding many thousands of sub-feature vectors even when there are repetitions and similarities between sub-features:

$$Z_x = \sum_{i=1}^{cx} \left(Z_i^i \otimes \prod_{j=0}^{i-1} p_j^0 \right) + StopVec \otimes \prod_{j=0}^{cx} p_j^0 \tag{1}$$

Omitting $StopVec$ for readability, this expands to,

$$Z_x = p_0^0 \otimes Z_1^1 + p_0^0 \otimes p_1^0 \otimes Z_2^2 + p_0^0 \otimes p_1^0 \otimes p_2^0 \otimes Z_3^3 + \ldots \tag{2}$$

Where

- $\otimes$ is defined as the XOR operator;

- $+$ is defined as the Bitwise_Majority_Vote/Add operator;

  The exponent operator in these equations is a cyclic-shift operation —i.e., positive exponents mean C_{shift_right}, negative exponents mean C_{shift_left}. Note that cyclic-shift is key to the recursive binding scheme since it distributes over $+$ (i.e., bitwise majority addition) and $\otimes$ (i.e., XOR) hence it automatically promotes its contents into a new part of the hyper-dimensional space; thus, keeping levels in the hierarchy separate;

- Z_x is the next highest semantic vector containing a *superposition* of x sub-feature vectors. Z_x these vectors can be combined using 1 into still higher level vectors. For example, Z_x might be the superposition of $B1 = \{A1, A2, A3, \ldots\}$ or $C = \{B1, B2, B3, \ldots\}$;

- $\{Z_1, Z_2, Z_3, \ldots Z_n\}$ are the sub-feature vectors being combined for the individual nodes of Figure 1. Each Z_n itself can be a compound vector representing a sub-workflow or a complex vector description for an individual service step, built using the methods described by Simpkin et al;[2]

- $p_0, p_1, p_2, \ldots$ are a set of known *atomic* role vectors used to define the current position or step in the workflow.

- cx is the size of bundled vector Z_x, i.e., the number of sub-feature vectors being combined; and

- $StopVec$ is a role vector owned by each Z_x that enables it to detected when all of the steps in its (sub)workflow have been executed.

The key to understanding the power of Equation 1 as a hierarchical binding scheme results from the fact that the cyclic_shift operator distributes over XOR and we stipulate that cyclic_shift takes precedence over XOR. This means that if a compound vector such has Z_x, as shown in Equation 2, is used as sub-feature vector for a higher level concept then the p vectors associated with each sub-feature vector Z_n are automatically promoted to a new value by the cyclic_shift. This is illustrated in Figure 1 where we have replaced the XOR operator with a 'dot' symbol in order to de-clutter the diagram.

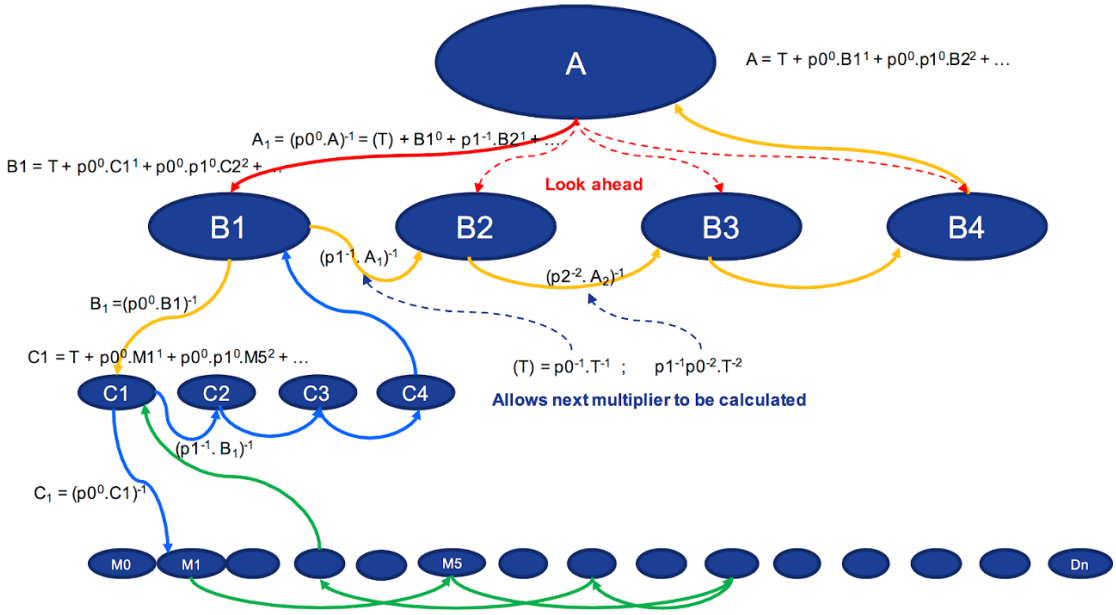

Figure 1. Schematic representation of the hierarchical recursive binding scheme showing how vectors compose into higher level vectors which can then be recursively un-bound from the higher level to lower level vectors.

As we shall see in Section 7, Equation 1 can be used to construct service description vectors and in turn, using the same recursive formulation, these vectors can be combined into service workflow vectors at the next higher semantic level. In this case the vectors, $Z_1 \ldots Z_{cx}$ represent the individual services and service workflow compositions and the vectors $p_o \ldots p_{cx-1}$ are a set of random vectors chosen to represent the position of the component services in the workflow.

To determine the order of the services in the workflow the workflow vector can be recursively unbound by performing an XOR operation using the current position vector and performing a left cyclic shift on the resultant vector as follows:

$$(\, Z_x \otimes p_0^0)^{\,-1} = Z_1 + p_1^{-1} \otimes Z_2^1 + \ldots = Z_1 + noise \tag{3}$$

In this case the resulting vector is a noisy version of the vector Z_1 and to identify that this vector is the requested vector Z_1, it is compared with a list of possible vectors and verified to be Z_1 by virtue of having the lowest *Hdist*. This process is often referred to as a 'clean-up' memory operation.

In the case of distributed services that we consider in Sections 5 to 8, the workflow vector is multicast in the communications network (as a single communications packet) and VSA enabled Services listen to all transmissions and compare the current state of the workflow vector with their own local list of vectors that represent their service capabilities. This can be envisaged as a distributed 'clean-up' memory. Service that match the current unbound service vector offer themselves as candidates that can perform the requested service and one of them is selected. This process is described in detail in Section 7. When a workflow vector is passed between services in this way it is necessary for the receiving service to be able to determine how many times the vector has been unbound in order to perform the appropriate position vector unbinding operation. To do this an additional universally agreed tag vector T is added to Z_x such that each time the vector is unbound the T vector becomes permuted in a predictable way. Once the currently active service has completed its own workflow step it uses the current permutation of the T vector to calculate its position within the received workflow vector. The service can therefore unbind the received workflow vector and re-transmit. This operation can also be implemented as a 'clean-up' memory operation by pre-storing the possible permutations of the T vector. The process continues with the recursively unbound workflow vector being passed between recruited services until the workflow represented by the vector Z_x is complete.

3.1 Vector Binding Capacity and Dynamic Control of Vector Length

The number of component binary vectors that can be combined into a single compound vector whilst retaining the capability to unbind and recover each vector with a high probability (i.e. an error of $\leq 10^{-9}$) depends on the number bits used to represent the vectors and the degree of similarity with other vectors that represent other concepts.This is illustrated in Figure2 over a range of 0 to 50 bundled vectors. If the number of vectors that are bundled into the single compound vector are known, together with an estimate of the similarity of the component vectors, then it is possible to dynamically reduce the size of the compound vector and still be able to recover the individual component vectors. For example, in the case of 20 component vectors, if the component vectors are not similar (i.e., orthogonal) then only 2000 bits are required for the compound vector. A property of our VSA binary vectors is that the compound vector is a distributed representation of the component vectors and therefore a 10Kbit compound vector with only 20 component vectors can be simply truncated to 2K bits and the component vectors can still be recovered with high probability. This truncation can be done dynamically and used to reduce the bandwidth requirement when transmitting vectors around the communications network. Further details of this approach can be found in Simpkin et al .[4]

4. IMPLEMENTING VSA OPERATIONS USING 'IN MEMORY' PROCESSING

Implementing the required VSA operations in standard computing hardware is relatively straight forward but because of the large vector sizes involved this requires the manipulation of large vector arrays and is relatively computationally expensive. As an example, we have seen in Section 3 that to perform 'clean-up' memory operations requires a search between the hypervectors representing the service capability S_i and the noisy unbound workflow hypervector Z, using a suitable similarity metric, such as *Hdist* to determine if there is a match. In

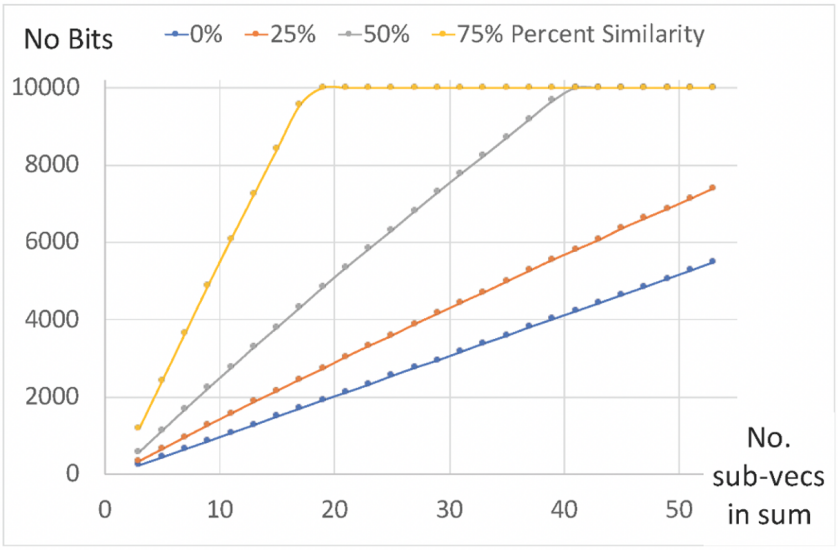

Figure 2. The number of bits required to ensure successful unbinding of a bundled vector depends on the number of vectors summed into the bundle and the similarity of the vectors in the clean-up memory against which the query vector is being compared

this section we describe how we can perform the operations in parallel and with much greater energy efficiently using Phase Change Memory (PCM) devices, a type of memristive devices that rely on the rapid and reversible phase transition of phase-change materials such as $Ge_2Sb_2Te_5$.

We begin by noting that because of the associativity of addition operations, the XOR operations used in the vector binding can be decomposed into an alternative representation using the addition of two dot product terms as shown in Equation 4 as follows:

$$Class_{pred} = \underset{i\epsilon(1,\ldots,cx)}{\arg\max}\, Z \cdot S_i + \bar{Z} \cdot \bar{S_i} \cong \underset{i\epsilon(1,\ldots,cx)}{\arg\max}\, Z \cdot S_i \tag{4}$$

where $\bar{Z}$ denotes the logical complement of Z. Since the operations associated with HDC ensure that both the hypervectors have an almost equal number of zeros and ones, the dot product can then be used as an alternative viable similarity metric. To indicate that we are using this formulation and to be consistent with previously reported work [23] in the remainder of this section of the paper we will refer to the unbound workflow vector Z as the query vector Q and the service vectors S_i as prototype hypervectors P_i and use the dot product as an equivalent similarity measure to *Hdist*.

This formulation now allows us to implement the clean-up memory function by using two PCM crossbar arrays of c rows and d columns as shown in Figure 3.

The prototype service hypervectors, P_i, are programmed into one of the crossbar arrays as conductance states. Binary '1' elements are programmed as crystalline states and binary '0' elements are programmed as amorphous states. The complementary hypervectors $\overline{P_i}$ are programmed in a similar manner into the second crossbar array. The query hypervector Q and its complement $\overline{Q}$ are applied as voltage values along the wordlines of the respective crossbars. In accordance with the Kirchoff's current law, the total current on the ith bitline will be equal to the dot-product between query hypervector and ith prototype hypervector. The results of this in-memory dot-product operations from the two arrays are added in a pairwise manner using a digital adder circuitry in the periphery and are subsequently input to a winner-take-all (WTA) circuit which outputs a '1' only on the bitline corresponding to the class of maximum similarity value. When dot p similarity metric is

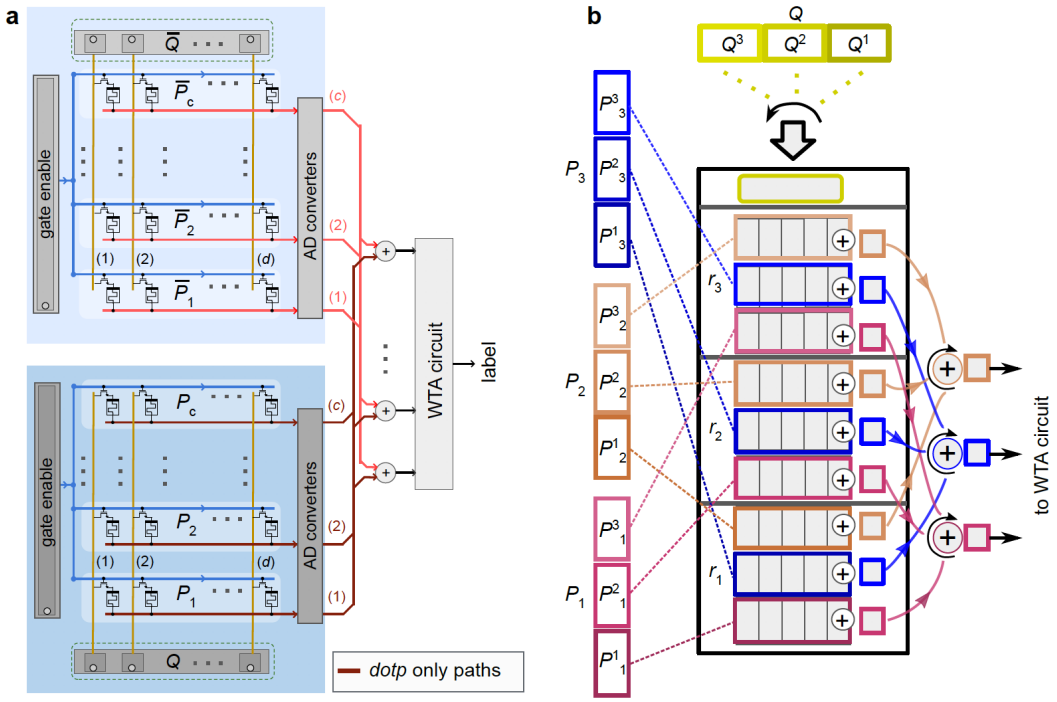

Figure 3. Clean-up memory using processing can be performed using two PCM crossbar arrays and CMOS near memory to perform the dot product summation. The diagram also shows how the service hypervectors are partitioned to improve the matching performance.

considered, only the crossbar encoding P_i is used and the array of adders in the periphery is eliminated, resulting in reduced hardware complexity.

4.1 PCM-based hardware platform

The experimental hardware platform is built around a prototype phase-change memory (PCM) chip that contains PCM cells that are based on doped-Ge2Sb2Te2 (d-GST) and are integrated into the prototype chip in 90 nm CMOS baseline technology. In addition to the PCM cells, the prototype chip integrates the circuitry for cell addressing, on-chip ADC for cell readout, and voltage- or current-mode cell programming. The experimental platform comprises the following main units:

- a high-performance analog-front-end (AFE) board that contains the digital-to-analog converters (DACs) along with discrete electronics, such as power supplies, voltage, and current reference sources;
- an FPGA board that implements the data acquisition and the digital logic to interface with the PCM device under test and with all the electronics of the AFE board;
- a second FPGA board with an embedded processor and Ethernet connection that implements the overall system control and data management as well as the interface with the host computer.

The prototype chip contains 3 million PCM cells, and the CMOS circuitry to address, program and readout any of these 3 million cells. In the PCM devices used for experimentation, two 240 nm-wide access transistors are used in parallel per PCM element. The PCM array is organized as a matrix of 512 word lines (WL) and 2048 bit lines (BL). The PCM cells were integrated into the chip in 90 nm CMOS technology using the key-hole

process. The bottom electrode has a radius of 20 nm and a length of 65 nm. The phase change material is 100 nm thick and extends to the top electrode, whose radius is 100 nm. The selection of one PCM cell is done by serially addressing a WL and a BL. The addresses are decoded and they then drive the WL driver and the BL multiplexer. The single selected cell can be programmed by forcing a current through the BL with a voltage-controlled current source. It can also be read by an 8-bit on-chip ADC. For reading a PCM cell, the selected BL is biased to a constant voltage of 300mV by a voltage regulator via a voltage Vread generated via an off-chip DAC.

The sensed current, Iread, is integrated by a capacitor, and the resulting voltage is then digitized by the on-chip 8-bit cyclic ADC. The total time of one read is 1 ms. For programming a PCM cell, a voltage Vprog generated off-chip is converted on-chip into a programming current, Iprog. This current is then mirrored into the selected BL for the desired duration of the programming pulse. The pulse used to program the PCM to the amorphous state (RESET) is a box-type rectangular pulse with duration of 400 ns and amplitude of 450 mA. The pulse used to program the PCM to the crystalline state (SET) is a ramp-down pulse with total duration of approximately 12 ms. The access-device gate voltage (WL voltage) is kept high at 2.75V during the programming pulses.

4.2 Experimental Results

Experiments were performed using the prototype PCM chip to evaluate the effectiveness of the proposed implementation using Equation 4 for the associative in-memory search. The device was programmed to store 49 10,000 bit prototype hypervectors (approximately one third of the potential capacity). The parallel multiplication operations required to perform the search are performed in the analog domain (by exploiting the Ohm's law) on-chip and the remaining operations (e.g., summation operations) are implemented in software in near-memory. It was found that, when a naive mapping of the prototype hypervectors to the array is used, the chip-level variability associated with the crystalline state detrimentally affects the AM search operation. To address this issue, we employed a coarse grained randomization strategy where the idea is to segment the prototype hypervector and to place the resulting segments spatially distributed across the crossbar array (see Figure 3b). This helps all the components of prototype hypervectors to uniformly mitigate long range variations. The proposed strategy involves dividing the crossbar array into f equal sized partitions (r1,r2,...,r f) and storing a 1= f segment of each of the prototype hypervectors (P1,P2,...,Pc) per partition. Here f is called the 'partition factor' and it controls the granularity associated with the randomization. To match the segments of prototype hypervectors, the query vector is also split into equal sized subvectors Q1,Q2,...,Qf which are input sequentially to the wordline drivers of the crossbar.

A statistical model that captures the spatio-temporal conductivity variations was used to evaluate the effectiveness of the coarse-grained randomized partitioning method. Simulations were carried out for different partition factors. These results indicate that the matching accuracy increases with the number of partitions typically from 82.5 to 96 percent by randomizing with a partition factor of 10 instead of 1 and this was confirmed by experiment.

To benchmark the performance of the system the PCM crossbar sections were also implemented in 65 nm CMOS technology using CMOS distributed standard cell registers with associated multiplier adder tree logic and binding logic respectively to construct a complete CMOS HD processor with the intention of comparing against the figures of merits of the PCM crossbar based architecture. A comparison of the performance between the all-CMOS approach versus the PCM crossbar based approach showed that with an average energy per query of 9.44 nJ in the PCM array, an improvement in total energy efficiency of 117.5:1 compared to the CMOS implementation. Similarly there was a 31.9 x reduction in required chip area.

In Section 7 we discuss how this type of PCM device can be integrated the logical architecture of a VSA enabled service and where the potential energy savings could be realized. These types of device would clearly have a significant impact in energy constrained environments such as IoBT and IoT applications.

5. THE SCENARIO

In this section we describe a communications re-planning scenario to illustrate how the semantic vector representation can be used to perform a complex command and control task. The relevant MDO setting is based in

(TacCIS) environment where agile rapid communications re-planning is required. We show how a communications plan can be represented as a VSA vector and how exchange and recursive unbinding of the vector is used to discover radio's operating under a current communications plan and instruct them to switch to a required target plan.

The scenario is shown in Figure 4 with three communications channels indicated by the three planes (CH1 - CH3). Initially different types of platform and their associated radio's are assigned to the different channels. Some of these radio's are assumed to be on a fixed channel, some can operate on any two channels and a minority can operate on all three channels. Whilst we have chosen three channels to demonstrate the principle, increasing the number of available channels and expanding the capabilities of the radios to use multiple channels is not a limitation. In Figure 4, for illustration, we show a current and target configuration of the radios and their associated platform (e.g., personal radio's are assumed to be associated with an individual user).

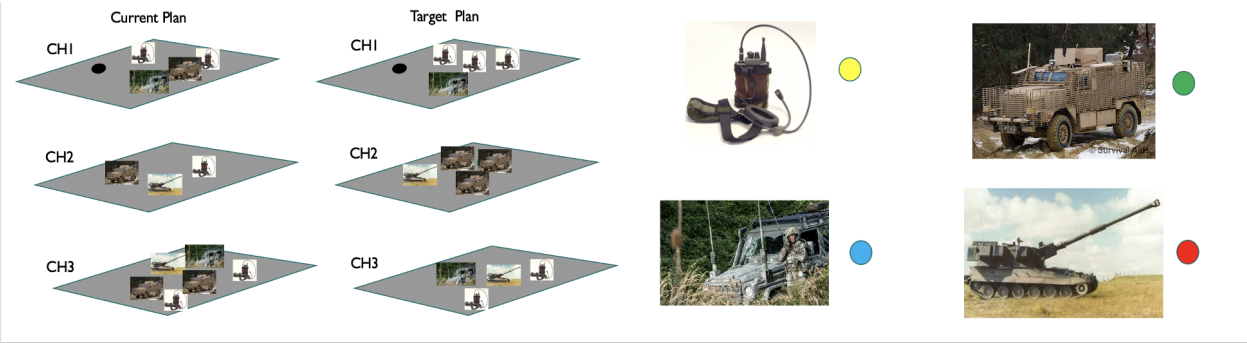

Figure 4. Schematic of the scenario showing the current and target communication plans in three channel layers. The colour coding of the different radio/platforms is used to illustrate the vector unbinding

To aid the explanation, in this section we simplify the description of each radio type by describing it using a colour and the channels that it can operate on. The vector binding scheme described in Section 3, is then used to construct vectors with the requirement for a particular radio using the radio colour and required channel number. For example, if we require a personal radio (yellow node) to be on channel 1 we designate the vector that represents this as vector YC1. We then show how these vectors can again be combined into an actual target plan vector workflow. However since this process results in a single vector representing each radio by colour and channel is sufficient to explain how the scenario unfolds. In Section 6 we explain how a complex semantic vector that represents a detailed description of a radio and its associated platform can be constructed from multiple component vectors that can represent the detailed radio characteristics, the role and the associated platform.

To construct the required workflow we combine the individual radio vectors into a single communications plan vector. Although we are using Equation 1 to do this, we indicate this by a simple addition operation to show how the process works. Our target communications plan as shown in Figure 4 requires three personal radio's (yellow nodes) and one command vehicle (blue node) on channel 1; Three armoured vehicles (green nodes) and an artillery unit (red Node) on channel 2 and a combination of blue, red and two yellow nodes on channel 3. To represent this plan we create a vector Z i.e.,

$$Z = [YC1] + YC1 + YC1 + BC1 + GC2 + RC2 + BC3 + RC3 + YC3 + YC3 + BlC1 \tag{5}$$

where the bracketed vector indicates the current unbound state of the vector i.e. Z = YC1 + noise

The node that makes the re-planning request, in this case the black node on channel 1 (BlC1). If we require that this node is to be informed that the workflow has successfully completed it is included as the last node in the workflow. Importantly, Node BlC1 does not need to know how many radio channels there are, or where the radio's are currently located for the requested task to be completed.

Figure 5 shows some of the sequence of actions that occur as the scenario unfolds. The vector Z is initially transmitted (multicast or broadcast) from BlC1. Using a 10,000 bit vector, this communication can be performed

using a single communications packet. This message is received by all of the nodes currently listening on the same channel. Initially, the unbound vector looks like a noisy version of vector YC1 indicated by the rectangle highlighting the first YC1. All nodes on the channel compare their list of allowable configuration vectors with this vector and the one that best matches the request accepts the vector and takes over control. This is illustrated in Figure 5(a) where a yellow node on Channel 1 responds to the request (highlighted by a black surround) and accepts the unbound vector. How this operation is actually performed is described in detail in Section 7.

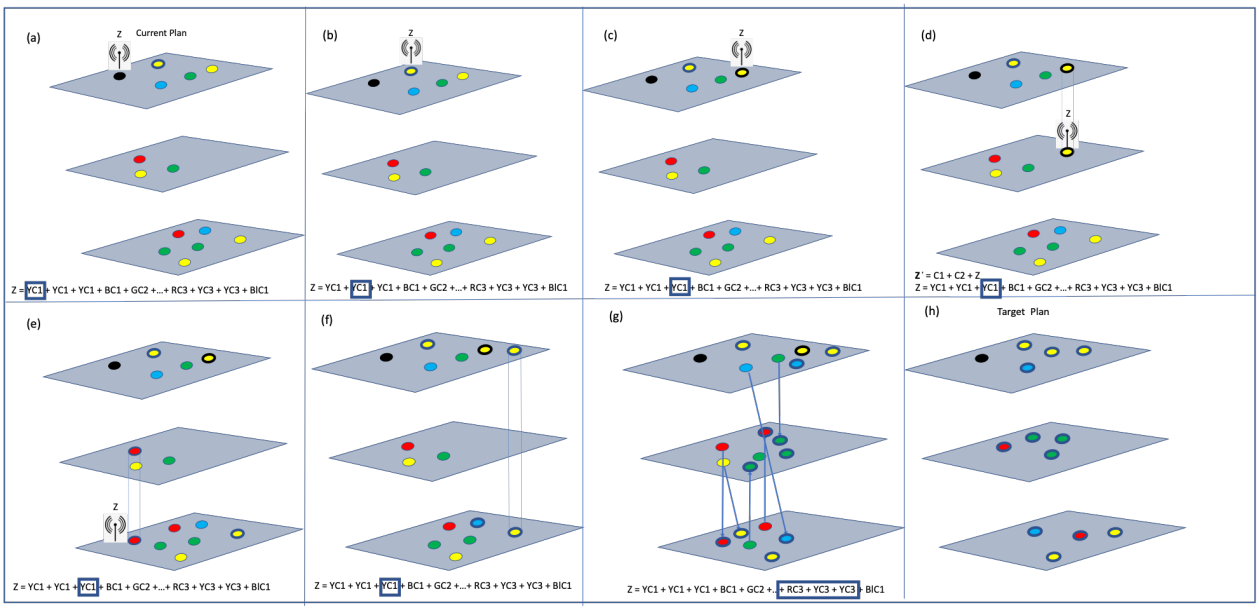

Figure 5. The diagram shows a subset of the sequence of operations (a) to (h) as the scenario unfolds

This response action prevents other yellow nodes listening on Channel 1 from responding to the initial request see. The receiving yellow node now unbinds the vector and re-transmits on Channel 1, see Figure 5(b). The new unbound vector is still a request for another yellow node on channel 1 and so, as shown in Figure 5(c), a second yellow node on Channel 1 matches and takes control and performs the same operations as the previous yellow node.

The vector is therefore again unbound and re-transmitted. This time the unbound workflow vector is a request for another yellow node to be on Channel 1. In this case, there is no response since no such node exists. To solve this challenge, the yellow node currently in control needs to discover a yellow node on another channel and instruct it to switch to Channel 1. In the scenario this particular yellow node can also operate on Channel 2 and so it temporarily switches channels and re-transmits the vector on Channel 2 as shown in Figure 5(d). If a yellow node that could switch to Channel 1 existed on Channel 2 then it in turn would respond and take control leaving the yellow node requester to return to Channel 1.

However, as illustrated in Figure 5(d), in this scenario the only yellow node on Channel 2 can only operate on Channels 2 and 3 and so it does not respond. If the requesting yellow node could switch to Channel 3 then it could try that channel as well but again in this example it cannot. To address this challenge the requesting yellow node needs to seek help from a node that can switch to a channel that has not already been tried. To do this the node constructs a new vector that we term a 'take-over' request vector. This vector is a new vector that binds the current unbound workflow vector with a vector that indicate which channels have already been searched. This is also illustrated in Figure 5(e) as the vector Z' where in this case:

$$Z' = Z + CH1 + CH2 \tag{6}$$

Whilst the creation of the new 'take-over' bound vector adds more noise, nodes receiving the vector Z' will have previously seen and cached the original unbound vector request (i.e. minus the channel information vectors C1 and C2). In response to hearing a 'take-over' vector on their current channel, nodes determine if they can switch to an alternate channel to those in the list and if they have the cached vector Z. If they do then they can respond and take control of the workflow vector. In the scenario the 'take-over' vector results in the discovery of a red node that can switch to channel 3 and it passes control to that node and returns to Channel 1. The red node switches to Channel 3 and transmits its cached version of the vector Z to discover a yellow node on Channel 3 that can switch to Channel 1,see Figure 5 (e). This yellow node accepts the vector causing the red node to return to channel 2 and discovered yellow node to switch to Channel 1. The process continues until all coloured nodes are on the required channels.

If the Z vector includes BlC1 as the final bundled vector then the final action of the process would discover BlC1 which would indicate to the requester that the workflow was completed.

6. REPRESENTING RADIO'S AND COMMUNICATIONS PLANS AS VSA VECTORS

In this section we describe how to represent radio's and their associated platform as VSA vectors and how these vectors can be combined into a new vector that represents a communications plan.

6.1 Representing Radio's as VSA Vectors

To represent radio's as VSA vectors, we use the vector binding scheme described in Section 3. In our scenario, an individual radio is specified by key-value pairs represented in a json description file. The json description defines the various attributes of the radio and its associated roles and platform characteristics. The keys (e.g., member, role) are asscoiated with corresponding values for each (e.g., Vehicle, Gateway). This json structure is used for the purpose of our scenario but any field descriptions and corresponding values can be used. The json file is automatically parsed and converted into binary vectors using the approach described in Reference .[3] The approach uses random binary vectors to describe each letter of the alphabet and any required additional symbols (e.g., punctuation, from which the various keys can be constructed. For example, the key 'company' is constructed from random vectors for each letter of the word as follows:

$$Z_{key-company} = c^0 \otimes o^1 \otimes m^2 \otimes \ldots \otimes y^6 \tag{7}$$

In this case the vector binding produces a unique 'key' vector for each key.

The 'value' vectors are constructed in a similar way but to allow for similar values to be compared the value vectors use both binding and bundling operations. For example the value 'alpha' is constructed as follows:

$$Z_{value-alpha} = a^0 \ + l^1 + p^2 \otimes + \ldots + a^4 \tag{8}$$

We have also been investigating the use of VSA vectors for both role and filler vectors that can be generated from a semantic vector space, for example Word2Vec,[28] via the method of randomised binary projection.[29] This enables semantically similar descriptions of radio's to be created allowing the discovery of similar services from different MDO partners.

The resulting key and value vectors are bound together to construct key value vectors and these vectors are bundled to create a description vector for each radio as follows:

$$Z_{radio-1} = Z_{key-id} \otimes Z_{value-1} + Z_{key-company} \otimes Z_{value-company1} + \ldots + Z_{key-SIDC} \otimes Z_{value-SFGPUCI---A} \tag{9}$$

In our scenario, since we want to instruct a radio to switch to a specific channel, we separately construct vectors for each of the allowable channels on which the associated radio can operate. For example:

$$Z_{UHFChannel1} = Z_{key-UHFChannel} \otimes Z_{value-1} \tag{10}$$

Figure 6. json representation of two radios each with different configurations

and

$$Z_{VHFChannel1} = Z_{key-VHFChannel} \otimes Z_{value-1} \tag{11}$$

And so the first radio in our example would also have vectors for VHF Channel 1 and UHF channels 1 - 3 together with bundled with a vector that described its key-value description pairs.

This approach provides the flexibility for a radio to have a number of accredited roles and channels on which it can operate and each configuration is stored on the radio device as a separate vector.

6.2 Representing Communications Plans as VSA Vectors

To represent a communications plan (comms-plan) as a VSA vector we consider each radio to be the equivalent of a service and the comms-plan itself as a workflow. The workflow needs to record the order in which the radio services are discovered and so we use the hierarchical binding scheme described in Section 3 to achieve this. In this case the service vector vectors are constructed as a vector that combines the required radio description vector and the channel vector that the plan requires that radio to be on, for example:

$$Z_{Comms-Plan} = p_0^0 \otimes Z_{radio-1} \otimes Z_{UHFChannel1} + p_0^0 \otimes p_1^0 \otimes Z_{radio-1} \otimes Z_{UHFChannel1} + \ldots \tag{12}$$

This is the actual vector corresponding to the simplified comms-plan vector (Equation 6) that we used in Section 5.

The construction of a comms-plan vector is shown schematically in Figure 7 where it can be see that the comms-plan vector is a hierarchical vector of vectors which, as a result of the normalisation, is the same dimension (e.g., 10,000 bits) as the original key and value component vectors.

When the comms-plan vector is transmitted the radio's on that channel are listening. In parallel these nodes compare the current unbound workflow vector with their own vector descriptions using a 'clean-up' memory type operation to determine if they have a match. There are a number of ways to perform the comparison but in the demonstration each radio XOR's its current description vector with the received unbound comms-plan vector. It then compares the resultant vector with its allowable channel vectors. If an allowable channel does not match then the hamming distance will be close to 0.5. A significant difference from 0.5 indicates a match. If there is a match then this radio is a candidate for this step of the plan. Details of how the radio is selected and the other required operations is given in Section 7.

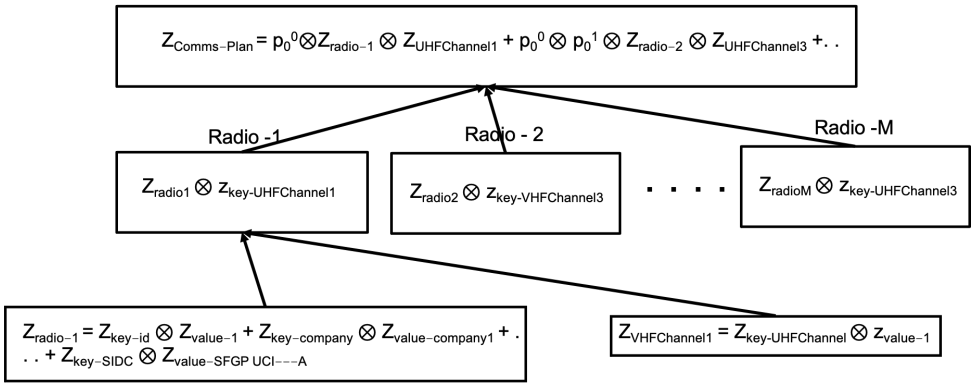

Figure 7. Hierarchical construction of the Comms-Plan vector as a vector of vectors

7. LOGICAL ARCHITECTURE OF A SERVICE

In this section we provide further details of how a service node (in this case a radio) performs the different operations that are required. Figure 8 is a flow chart that shows the logical sequence of operations. Each node has two states. It is either acting in a 'Listening State' or in a 'Requesting State'. In Figure 8 the 'Listening State' workflow is connected by orange flow and is un-shaded. In the 'Requesting State' the flow is connected by blue lines and the actions are shaded. The red dashed lines indicate message transmission/reception.

7.1 Main Actions Performed

The different actions performed are summarised below:

7.1.1 Listening State

In the listening state each node is monitoring its current channel for radio packets that may contain VSA vectors on which it can act. On receipt of a radio packet the node compares the contained message to determine if it is either a VSA 'workflow request' vector or 'take-over' request vector.

7.1.2 Workflow Request

If it is a 'workflow request' vector it caches the vector and then compares this with its own list of permitted configurations using the hamming distance to determine the degree of match. If there is no match with any of its permitted configurations then the process returns to listening for new vectors.

If the 'workflow request' vector does match then based on the quality of match the node waits for a period of time based on a calculated property called fitness. Fitness in this case has two components. The first component of fitness is a time delay that is inversely proportional to the match quality (i.e. a perfect match results in zero delay and a poor match with a longer delay). The second element of fitness is a time period that is inversely proportional to the number of configurations in which the radio can operate. If a node can only operate on one channel for example then we want to select this node with a higher priority than a node with several channels since the latter essentially has more flexibility to match to other parts of the workflow request. During this period the node prepares a 'match vector', which is used to inform the requesting node that this radio-service can perform the service request and can accept control. It also listens on the channel to see if a similar 'match vector' is sent from any other node. If a similar 'match vector' is seen then the node does not offer itself and returns to listening for new vector requests. If no similar 'match vectors' are seen then the match vector is transmitted and the node awaits for confirmation that it has been selected ('Winner-Selected Vector). All nodes that respond to the original request will see the 'Winner-Selected Vector' and based on the content can determine if they are the winner. However, if they don't see the message (e.g., they have gone out of range) a user specified wait time will be exceeded and the node reverts to listening for further requests. The node that is selected as the winner, via the 'Winner Selected' vector, sets its records to the next state that it has been requested to enter (e.g. the channel it is to operate on) and then unbinds the comms-plan workflow vector and enters the 'Requester' state.

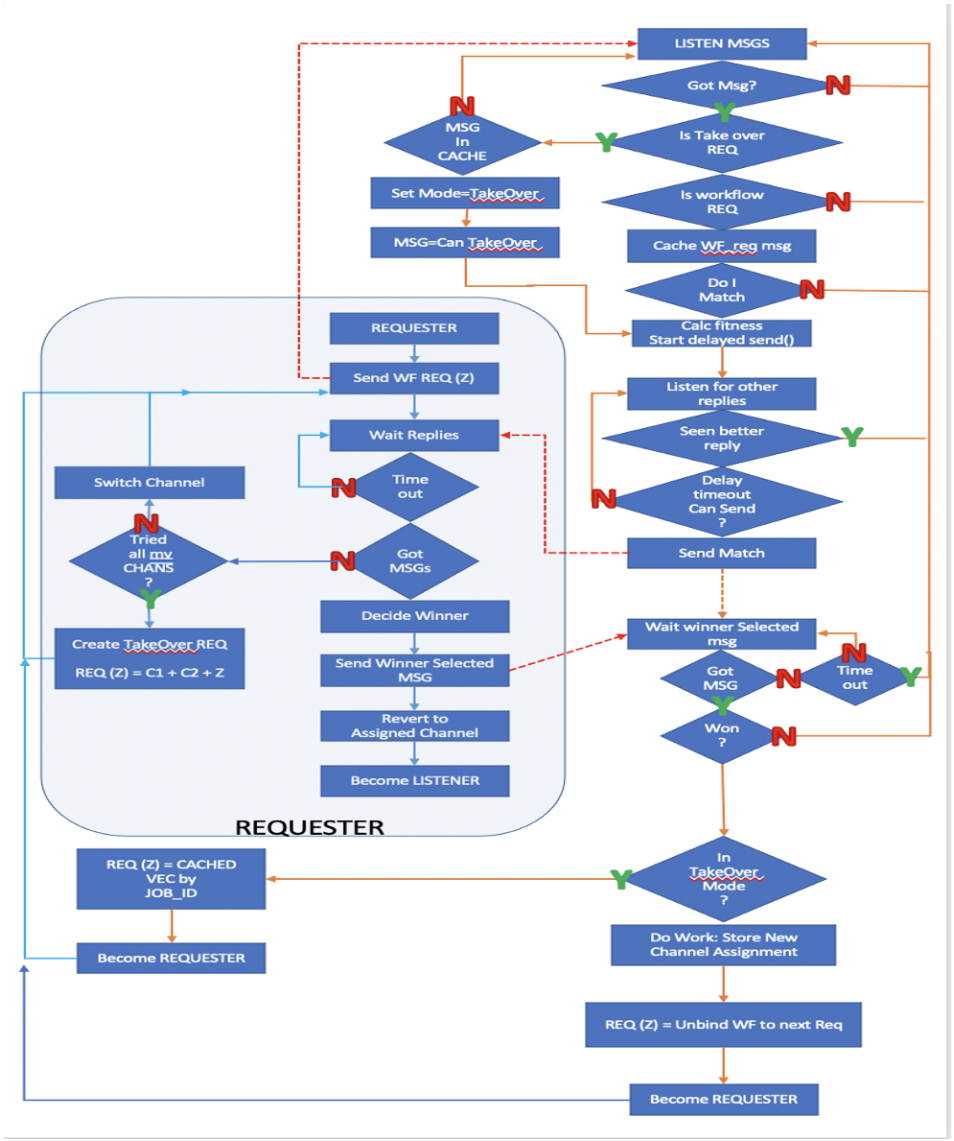

Figure 8. Flow diagram showing the logical sequence of operations that each radio node performs to achieve the required sequence of operations

7.1.3 Take-Over Request

A 'take-over' request is generated when a requesting node does not receive a response on any of its available channels. The 'take-over' request specifies the channels on which the 'workflow request' has been made. Details of this vector are described in the next sub-section. If a node can switch to a channel other than the ones that have been tried it is a candidate to accept the 'take-over' request. This node compares the 'workflow request' job-id with the 'workflow request' vectors in its cache. Since this node should have heard the unsuccessful initial request it will have a 'clean' version of the corresponding 'workflow-request' vector in its cache which it can potentially transmit by proxy on a channel that could not be reached by the original requester. The node registers that it is in the 'take-over' mode and now goes through the process of determining its fitness etc. as in the 'workflow request mode' and if it has not seen a better response from another node it offers to accept the 'take-over' request. If it is successful it now obtains the vector from its cache and enters the 'Requester State'

7.1.4 Requester State

In the requester state the first action is to broadcast/multicast the unbound workflow vector and then listen for replies on the same channel. Whilst responding nodes try to minimise the number of responses to a request using the 'fitness' delay mechanism the possibility exists that a node does not hear a response from another node and replies. The requester may therefore receive more than one response. To handle this possibility the requester waits for a period of time after sending out the 'request vector' and then checks to see how many responses have been received. The response vector includes a measure of the node fitness and so the requester is able to select the best response and encodes the 'winner selected' vector which is multicast to the network. The nodes that sent a response 'hear' the 'winner-selected' vector from which they can determine if they have been selected as winner. The requester then reverts to its assigned channel and enters the 'Listener State'.

If having sent a request no response is received then the requester repeats the process on any other channel that it can switch to. If all channels are exhausted then it creates a 'take-over' request vector. The 'take-over' request vector binds together the current unbound request vector together with vectors that indicate which channels have already been searched for a candidate radio to meet this request. This informs any node that is going to act as a proxy which channels have already been searched. The 'take-over' request is therefore a recursive process which continues until a radio that can fulfil the request is found and at that point the workflow composition proceeds.

7.2 Integration points for the PCM clean-up memory

Using PCM devices to perform some of these operations would result in significant performance and energy efficiency gains. Whilst the current demonstration does not make use of the PCM directly to perform the matching operations, we have shown in Section 4 how the required matching operations can be performed as 'In Memory' operations and the significant energy savings and performance improvements that could be achieved. To maximise the benefit of the 'In Memory' processing requires some re-engineering of current logical flow. In the new model all the different actions that a service can perform are represented as vectors and these are all stored in the PCM associative memory. On receipt of a workflow vector the received vector is compared with all possible configurations in parallel and the best matching vector, if any, indicates that this service is a candidate for the requested task. Since these operations will be performed by all listening service nodes it is important that the similarity measure made by each device can be compared across the different PCM devices and our current work is focused on showing how this can be achieved. The operation to discover the current unbound state of the received vector from the modified tag vector (T) could also be performed using the PCM device and this would significantly improve both the energy efficiency and speed of response of the services. We are currently investigating how the bundling and unbinding operations can be performed using a combination of 'In Memory' and 'Near Memory' processing so that the majority of the required operations can be performed on a single highly compact and energy efficient VSA device.

8. DEMONSTRATION AND EXPERIMENTAL RESULTS

In this section we describe our demonstration simulation environment and the experimental results obtained.

8.1 EMANE Network Emulation Environment

To demonstrate the performance of the VSA approach we have constructed a simulation of a representative TacCIS radio network using the EMANE simulation environment. The main emulated network ("UHF") uses a 1MHz bandwidth for each channel on a 300MHz base frequency. Each channel is modelled using a TDMA radio, allowing us to dynamically vary the radio frequency and time access. This results in a node-to-node bandwidth of approximately 16.5Kbits/sec, average latency of 350ms and 155ms jitter. Each radio, its associated platform and the code required to perform the VSA operations are encapsulated in Linux Containers running EMANE. Making channel changes EMANE doesn't have a pre-built model for a radio with multiple channels, so a close equivalent was needed. One way of looking at channels is to have different frequency bands per channel, and switch between them to change. Only one EMANE mac model allows for changing frequency, which is the TDMA (Time-Division Multiple Access) radio model. Since we had previously worked with this model in work in configuring the VHF radio in the Anglova scenario we had an existing Perl script that could generate TDMA

timing schedules for an arbitrary network plan. We therefore modified this to support multiple frequency bands. The downside of this approach is that a single radio cannot on its own make the decision to switch channels, as it needs a slot in the schedule. In reality this would be performed by sending a message in a "management slot" in the schedule, but for the emulation we have used the approach to send a message on the "management back-channel" to the emulation host to compute a new schedule including the new node, and send this out to all the affected nodes. This achieves the required results in the network layer with the minimum of changes on the emulation environment (the alternative would be to implement new code to the EMANE TDMA model to allow a single radio to effect changes in other radios, which was outside of scope for this piece of work).

The scenario is loosely based around the Angloval tactical military scenario [5] and experimentation environment developed by NATO IST-124 Research Task Group and released into the public domain in order to facilitate experimentation with networking protocols and algorithms by the community at large. Anglova scripts are created to represent the current network plan and these are parsed to automatically create the containers running EMANE and these are controlled using a network bridge. Each container runs an EMANE executable per emulated radio, which can interact with the other EMANE instances using multicast over the control network. Each EMANE instance creates a virtual network interface for user applications to use.

The simulation performed in the IBM Fyre cloud environment using a Ubuntu 20.04 VM, which has been set up with the Anglova network emulation scripts, a Mosquitto MQTT server and the Node-RED orchestration engine.

8.2 Demonstration Configuration

The demonstration scenario comprises sixty radio assets, where each asset is described by a set of characteristics and is initially assigned to a specific radio channel. Each asset is also given a list of the channels on which it is accredited to operate. The initial and target communications plans are shown in Figure 9 for the first 25 radios in the scenario.

id	company	troop	section	member	Battery	Available_Role	Role	State	SIDC	UHF_Channel	UHF_Channels
1	company1	1	1	vehicle	200	Gateway|File|Chat|Email	Gateway	Present	SFGPUCIZ---D	1	1|2|3
2	company1	1	1	charlie-cmd	2			Present	SFGPUCI----A	1	1|2
3	company1	1	1	charlie	2	Backup_Gateway		Present	SFGPUCI----A	1	1|2
4	company1	1	1	charlie	2	Backup_Gateway		Present	SFGPUCI----A	1	1|2
5	company1	1	1	charlie	2	Backup_Gateway		Present	SFGPUCI----A	1	1|2
6	company1	1	1	vehicle	200	Gateway|File|Chat|Email	File	Present	SFGPUCIZ---D	1	1|2|3
7	company1	1	1	delta-cmd	2			Present	SFGPUCI----A	1	1|2
8	company1	1	1	delta	2	Backup_Gateway		Present	SFGPUCI----A	1	1|2
9	company1	1	1	delta	2	Backup_Gateway		Present	SFGPUCI----A	1	1|2
10	company1	1	1	delta	2	Backup_Gateway		Present	SFGPUCI----A	1	1|2
11	company1	1	2	vehicle	200	Gateway|File|Chat|Email	Chat	Present	SFGPUCIZ---D	1	1|2|3
12	company1	1	2	charlie-cmd	2			Present	SFGPUCI----A	1	1|2
13	company1	1	2	charlie	2	Backup_Gateway		Present	SFGPUCI----A	1	1|2
14	company1	1	2	charlie	2	Backup_Gateway		Present	SFGPUCI----A	1	1|2
15	company1	1	2	charlie	2	Backup_Gateway		Present	SFGPUCI----A	1	1|2
16	company1	1	2	vehicle	200	Gateway|File|Chat|Email	Email	Present	SFGPUCIZ---D	1	1|2|3
17	company1	1	2	delta-cmd	2			Present	SFGPUCI----A	1	1|2
18	company1	1	2	delta	2	Backup_Gateway		Present	SFGPUCI----A	1	1|2
19	company1	1	2	delta	2	Backup_Gateway		Present	SFGPUCI----A	1	1|2
20	company1	1	2	delta	2	Backup_Gateway		Present	SFGPUCI----A	1	1|2
21	company1	2	1	vehicle	200	Gateway|File|Chat|Email	Gateway	Present	SFGPUCIZ---D	2	1|2|3
22	company1	2	1	charlie-cmd	2			Present	SFGPUCI----A	2	1|2
23	company1	2	1	charlie	2	Backup_Gateway		Present	SFGPUCI----A	2	1|2
24	company1	2	1	charlie	2	Backup_Gateway		Present	SFGPUCI----A	2	1|2
25	company1	2	1	charlie	2	Backup_Gateway		Present	SFGPUCI----A	2	1|2

Initial Plan

id	company	troop	section	member	Battery	Available_Role	Role	State	SIDC	UHF_Channel	UHF_Channels
1	company1	1	1	vehicle	200	Gateway|File|Chat|Email	Gateway	Present	SFGPUCIZ---D	1	1|2|3
2	company1	1	1	charlie-cmd	2			Present	SFGPUCI----A	2	1|2
3	company1	1	1	charlie	2	Backup_Gateway		Present	SFGPUCI----A	1	1|2
4	company1	1	1	charlie	2	Backup_Gateway		Present	SFGPUCI----A	1	1|2
5	company1	1	1	charlie	2	Backup_Gateway		Present	SFGPUCI----A	1	1|2
6	company1	1	1	vehicle	200	Gateway|File|Chat|Email	File	Present	SFGPUCIZ---D	3	1|2|3
7	company1	1	1	delta-cmd	2			Present	SFGPUCI----A	1	1|2
8	company1	1	1	delta	2	Backup_Gateway		Present	SFGPUCI----A	2	1|2
9	company1	1	1	delta	2	Backup_Gateway		Present	SFGPUCI----A	1	1|2
10	company1	1	1	delta	2	Backup_Gateway		Present	SFGPUCI----A	2	1|2
11	company1	1	2	vehicle	200	Gateway|File|Chat|Email	Chat	Present	SFGPUCIZ---D	1	1|2|3
12	company1	1	2	charlie-cmd	2			Present	SFGPUCI----A	2	1|2
13	company1	1	2	charlie	2	Backup_Gateway		Present	SFGPUCI----A	1	1|2
14	company1	1	2	charlie	2	Backup_Gateway		Present	SFGPUCI----A	2	1|2
15	company1	1	2	charlie	2	Backup_Gateway		Present	SFGPUCI----A	1	1|2
16	company1	1	2	vehicle	200	Gateway|File|Chat|Email	Email	Present	SFGPUCIZ---D	2	1|2|3
17	company1	1	2	delta-cmd	2			Present	SFGPUCI----A	1	1|2
18	company1	1	2	delta	2	Backup_Gateway		Present	SFGPUCI----A	2	1|2
19	company1	1	2	delta	2	Backup_Gateway		Present	SFGPUCI----A	1	1|2
20	company1	1	2	delta	2	Backup_Gateway		Present	SFGPUCI----A	2	1|2
21	company1	2	1	vehicle	200	Gateway|File|Chat|Email	Gateway	Present	SFGPUCIZ---D	3	1|2|3
22	company1	2	1	charlie-cmd	2			Present	SFGPUCI----A	1	1|2
23	company1	2	1	charlie	2	Backup_Gateway		Present	SFGPUCI----A	2	1|2
24	company1	2	1	charlie	2	Backup_Gateway		Present	SFGPUCI----A	2	1|2
25	company1	2	1	charlie	2	Backup_Gateway		Present	SFGPUCI----A	2	1|2

Target Plan

Figure 9. Initial and target communications plans showing the first 25 radio definitions

This target plan is requesting 20 specified assets to be on Channel 1, 20 on Channel 2 and 20 on Channel 3. For the purposes of the demonstration, twenty out of the sixty assets have been initially located on a channel that is not the required final channel and the objective is to discover these assets and instruct them to move to the required channel. To make the demonstration clearer the assets are given a unique identification number so that we are requesting specified assets to switch their radio channel. In Section 9 we describe some of the alternative ways in which the requests can be made so that a mixture of specific assets (i.e., a specified id) and assets with particular characteristics can be requested without the need to specify the id of these assets.

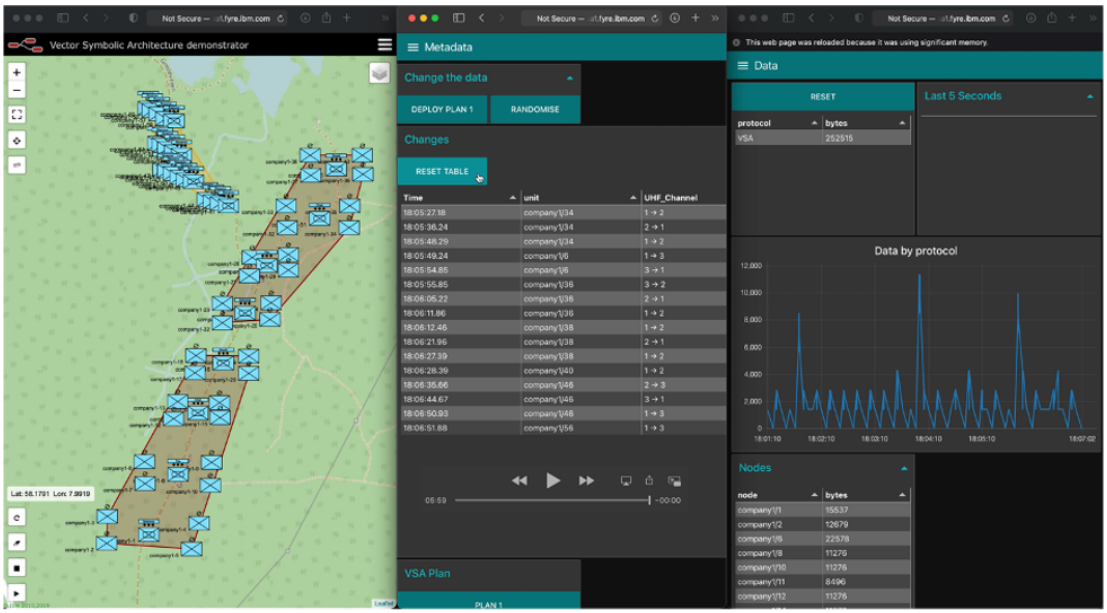

Figure 10. The demonstrator screen showing the location of the different radios and the communications that are occuring as the scenrio plays out.

A video of the demonstration is available at:
https://drive.google.com/file/d/1ZdYaak3q7w07XPXXj3K4xijWa7JJCJ8H/view?usp=sharing

The layout of the demonstration display is shown in Figure 10. On the left hand side is a map which displays the location of the different assets. In the centre of the display are various panels that indicate the actions that are taking place and on the right hand side of the display are panels that show the data transfers (bytes transmitted) as the workflow progresses.

In Figure 11 we show the maps corresponding to the location and initial and final configuration of the assets. The shaded polygons, each corresponding to a radio channel, indicate the area within which assets on a particular radio channel are located.

The workflow vector is constructed as described in Section 6. In general we would include all sixty nodes but for the purpose of the demo we have limited the number workflow vectors to restrict the runtime of the demonstration whilst illustrating all of the important features of the VSA approach.

8.3 Sequence of Operations

To understand the sequence of operations in the demonstration video we describe here the first few actions that are occurring in the scenario to illustrate the different actions that are performed in the rest of the scenario. The initial steps are as follows:

1. The workflow vector is injected by Company1-1 which is on Channel 1. In the demonstration video this action can be seen in the map display by the multicast of the vector to all assets on Channel 1 and a corresponding transmission of 770bytes. The initial unbound vector is an instruction for Company1-2 to switch to Channel 1. Since we are using dynamic vector truncation the workflow vector only requires 770 bytes compared to the 1403 bytes that would be required for the corresponding 10Kbit vector message.

2. Company1-2, is initially on Channel 2 and so it does not see the initial request and there is no response.

3. Company1-1 does not receive a response on Channel 1 and so switches to Channel 2 and tries again. This time it receives a response from Company1-2 and replies with an acknowledgement vector and switches back to its original channel which is Channel 1.

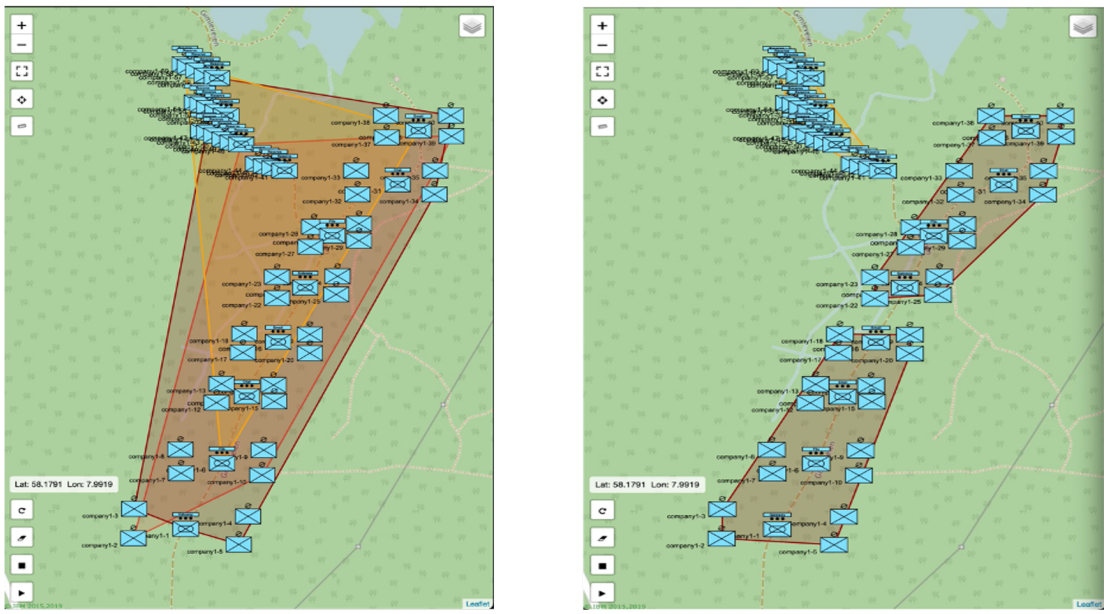

Figure 11. Initial and Final Configurations

4. Company1-2 now has control and switches to the instructed channel, which is Channel 1 and then unbinds the workflow vector and transmits this on Channel 1. The unbound workflow vector in this case is a request for Company1-6 to move to Channel 1.

5. Since Company1-6 is on channel 3 there is no response to the request on Channel 1 and so Company 1-2 switches to Channel 2 and transmits the vector but again there is no response.

6. Since Company 1-2 has now tried on all of its available channels it needs to discover an asset that can switch to channels that it cannot reach (in this case channel 3).

7. Company 1-2 therefore issues a 'Take-Over' request vector on Channel 2 and receives a response from Company1-56 which is a vehicle and can access all three channels.

8. Company 1-2 sends an acknowledgement to Company1-56.

9. Company1-56 now takes over control switches to Channel 3 and transmits the vector to which Company1-6 responds and Company1-56 acknowledges. This results in Company 1-56 returning to Channel 1 and Company 1-6 switching to Channel 1 as instructed.

The scenario proceeds until all of the assets are on the correct channel.

Figure 12 shows the state of the demonstration at step 7 in the scenario.

The data transmissions and bandwidth consumed were measured both with and without vector truncation respectively. Without vector truncation the total bandwidth required was 252,515 bytes and with truncation was 92,846 bytes a saving of 2.7x which is significant in the low bandwidth (16.5 Kbit/sec) TacCIS environement in which we are operating

It can be observed in the demonstration data protocol window Figures 10 and 12 that there are a number of larger spikes in the message traffic as the scenario progresses. These are as a result of the take-over requests and result from an artifact of the simulation that was unforeseen. The use of the TDMA in the radio model that we used means that when a take-over request is made all of the assets that can respond to the request on the channel respond since they all equally match the request vector. They therefore all respond in the same TDMA time-slot and in the current implementation it is not possible to suppress some of the responses on the basis that

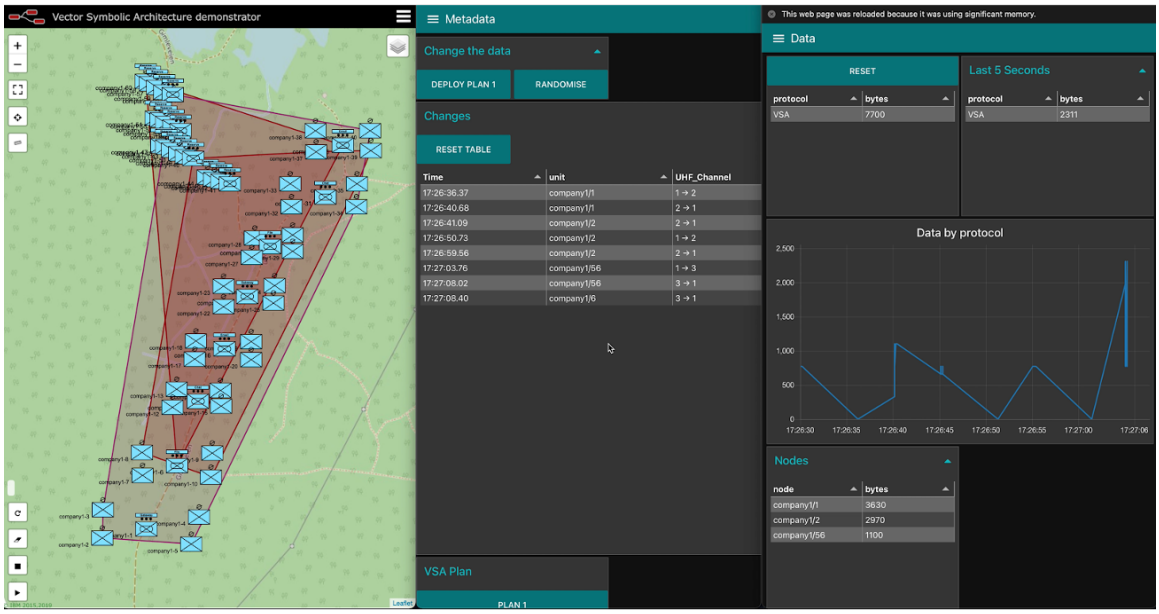

Figure 12. State of the scenario demonstrator at point 7

an asset 'hears' a response from another asset before it responds. We are currently working on a timing strategy that will avoid this and therefore reduce the number of transmissions during a take-over request.

The total time taken to switch all 20 assets to the correct channel was between 5 min 30 seconds and 6 min i.e., 16.5 seconds per channel with an average of 4642 bytes per asset. This time could potentially be significantly reduced by a more judicious choice of time delays bringing it closer to approximately 3 seconds per asset on 16.5 kb/sec channels.

9. CONCLUSION AND FUTURE DIRECTIONS

In this paper we have demonstrated how the Vector Symbolic Architecture (VSA) approach for defining and orchestrating distributed service workflows can be used to perform the task of communications replanning in a tactical communication and information systems environment (TacCIS). We have shown how communications plans can be represented in the VSA vector notation and how these vectors can be exchanged between assets in a TacCIS environment to discover and instruct assets on different radio channels to reconfigure into a desired configuration(communications plan). The associated demonstration environment shows the reconfiguration of twenty assets, initially assigned to random communications channels, can be discovered and instructed to achieve this goal in a completely decentralized manner. Whilst there is still scope for improvements in terms of the performance we have demonstrated the feasibility of the approach.

An important aspect of this paper has been to highlight the significant energy savings that can be obtained by performing the vector matching operations 'In Memory' using crossbar arrays of phase-change memory devices. These savings are particularly relevant in an IoBT/Iot environment where the devices and services being used to perform the workflow are operating in energy constrained environments. Our current work is focused on re-implementing the logic of our vector mapping and the binding and bundling operations to make greater use of this technology and on implementing all the required operations on a single highly compact and energy efficient VSA device.

The demonstration made use of the asset unique id to perform the reconfiguration however because the discovery process is based on a vector similarity measure, this constraint could be relaxed so that the matching is performed on the assets capability rather than its specific identity, or a mixture of both. This leads to the intriguing possibility that this type of vector representation could be used to perform the discovery and

tasking of assets to perform a range of functions based on their capabilities rather than on their specific identity. Such tasking could for example select assets that were closest to a particular location or that had the highest operational readiness in terms of factors such as vehicle fuel status, radio battery fife etc. The commander would simply define the requirements for the mission and the command and control workflow vector would be automatically generated and transmitted. This would result in the discovery of the specific assets and the other most capable assets that are available to perform the mission.

The VSA approach could therefore provide a completely new approach to agile command and control in which mission goals are achieved by discovering and maximizing the available assets minimizing the need for detailed pre-mission planning.

ACKNOWLEDGMENTS

This research was sponsored by the U.S. Army Research Laboratory and the U.K. Ministry of Defence under Agreement Number W911NF-16-3-0001. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Army Research Laboratory, the U.S. Government, the U.K. Ministry of Defence or the U.K. Government. The U.S. and U.K. Governments are authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation hereon. Part of this work was also supported in part by the European Research Council through the European Union's Horizon 2020 Research and Innovation Program under Grant 682675 and in part by the European Union's Horizon 2020 Research and Innovation Program through the project MNEMOSENE under Grant 780215.

REFERENCES

[1] Verma, D., Bent, G., and Taylor, I., "Towards a distributed federated brain architecture using cognitive iot devices," in [*9th International Conference on Advanced Cognitive Technologies and Applications (COGNITIVE 17)*], (2017).

[2] Simpkin, C., Taylor, I., Bent, G. A., De Mel, G., and Rallapalli, S., "Decentralized microservice workflows for coalition environments," in [*2017 IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computed, Scalable Computing & Communications, Cloud & Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI)*], IEEE (2017).

[3] Simpkin, C., Taylor, I., Bent, G. A., de Mel, G., Rallapalli, S., Ma, L., and Srivatsa, M., "Efficient orchestration of node-red iot workflows using a vector symbolic architecture," *Future Generation Computer Systems* **100**, 70–85 (2019).

[4] Simpkin, C., Taylor, I., Bent, G. A., Harbourne, D., Preece, A., and Ganti, R., "Constructing distributed time-critical applications using cognitive enabled services," *Future Generation Computer Systems* **111**, 117–131 (2020).

[5] N. Suri, e., "The anglova tactical military scenario and experimentation environment," in [*International Conference on Military Communications and Information Systems*], ICMCIS (2018).

[6] Kanerva, P., "Hyperdimensional computing: An introduction to computing in distributed representation with high-dimensional random vectors," *Cognitive computation* **1**(2), 139–159 (2009).

[7] Hinton, G. E., "Mapping part-whole hierarchies into connectionist networks," *Artificial Intelligence* **46**(1-2), 47–75 (1990).

[8] Eliasmith, C., "How to build a brain: from function to implementation," *Synthese* **159**(3), 373–388 (2007).

[9] Widdows, D. and Cohen, T., "Reasoning with vectors: A continuous model for fast robust inference," *Logic Journal of the IGPL* **23**, 141–173 (11 2014).

[10] Kleyko, D., *Pattern Recognition with Vector Symbolic Architectures*, PhD thesis, Luleå tekniska universitet (2016).

[11] Gayler, R. W., "Vector symbolic architectures answer jackendoff's challenges for cognitive neuroscience," *arXiv preprint cs/0412059* (2004).

[12] Levy, S. and Gayler, R., "Vector symbolic architectures: A new building material for artificial general intelligence," in [*Frontiers in Artificial Intelligence and Applications*], **171** (Jan. 2008).

[13] Levy, S. D. and Gayler, R. W., ""lateral inhibition" in a fully distributed connectionist architecture," in [*Proceedings of the Ninth International Conference on Cognitive Modeling*], (2009).

[14] C., S., G., B., and et al, "Coalition c3 information exchange using binary symbolic vectors," in [*MILCOM 2019 - 2019 IEEE Military Communications Conference (MILCOM), Norfolk, VA, USA*], 784–789) (2019).

[15] Yang, J. J., Strukov, D. B., and Stewart, D. R., "Memristive devices for computing. nature nanotechnology," *Nature nanotechnology* **8**, 13 (2013).

[16] Sebastian, A. and et al, "Temporal correlation detection using computational phase-change memory," *Nature Communications* **8**, 1115 (2017).

[17] Zidan, M. A., Strachan, J. P., and Lu, W. D., "The future of electronics based on memristive systems," *Nature Electronics* **1**, 22 (2018).

[18] Ielmini, D. and Wong, H.-S. P., "In-memory computing with resistive switching devices," *Nature Electronics* **1**, 333 (2018).

[19] Sebastian, A., Le Gallo, M., Khaddam-Aljameh, R., and Eleftheriou, E., "Memory devices and applications for in-memory computing," *Nature nanotechnology* **15**(7), 529–544 (2020).

[20] Li, H. and et al, "Hyperdimensional computing with 3d vrram in-memory kernels: Device-architecture co-design for energy-efficient, error-resilient language recognition," in [*International Electron Devices Meeting (IEDM)*], IEEE (2016).

[21] Li, H., Wu, T. F., Mitra, S., and Wong, H. S. P., "Device-architecture co-design for hyperdimensional computing with 3d vertical resistive switching random access memory (3d vrram)," in [*International Symposium on VLSI Technology, Systems and Application (VLSI-TSA), 1–2*], (2017).

[22] Wu, T. F. and et al, "Brain-inspired computing exploiting carbon nanotube fets and resistive ram: Hyperdimensional computing case study," in [*International Solid State Circuits Conference*], ISSCC (2018).

[23] Karunaratne, G., Le Gallo, M., Cherubini, G., and et al., "In-memory hyperdimensional computing," *Nature Electronics* **3**, 327–337 (2020).

[24] Plate, T. A., [*Distributed representations and nested compositional structure*], University of Toronto, Department of Computer Science (1994).

[25] Kanerva, P., "Binary spatter-coding of ordered k-tuples," in [*Artificial Neural Networks — ICANN 96*], von der Malsburg, C., von Seelen, W., Vorbrüggen, J. C., and Sendhoff, B., eds., 869–873, Springer Berlin Heidelberg, Berlin, Heidelberg (1996).

[26] Plate, T. A., [*Holographic Reduced Representation: Distributed Representation for Cognitive Structures*], CSLI Publications, Stanford, CA, USA (2003).

[27] Recchia, G., Sahlgren, M., Kanerva, P., and Jones, M. N., "Encoding sequential information in semantic space models: comparing holographic reduced representation and random permutation," *Computational intelligence and neuroscience* **2015**, 58 (2015).

[28] Mikolov, T., Chen, K., Corrado, G., and Dean, J., "Efficient estimation of word representations in vector space," *CoRR* **abs/1301.3781** (2013).

[29] Rachkovskij, D., "Estimation of vectors similarity by their randomized binary projections," *Cybernetics and Systems Analysis* **51**(5), 808–818 (2015).