Transport-Independent Protocols for Universal AER Communications ⋆

Alexander D. Rast, Alan B. Stokes, Sergio Davies, Samantha V. Adams, Himanshu Akolkar, David R. Lester, Chiara Bartolozzi, Angelo Cangelosi and Steve Furber

[1] School of Computer Science, University of Manchester
Manchester, UK M13 9PL
[2] Plymouth University, Plymouth, UK
[3] Istituto Italiano da Technologia, Genoa, Italy
{rasta,daviess,dlester}@cs.man.ac.uk
{steve.furber}@manchester.ac.uk
{samantha.adams,A.Cangelosi}@plymouth.ac.uk
{alan.stokes-2}@manchester.ac.uk
{Chiara.Bartolozzi}@iit.it
http://www.cs.manchester.ac.uk/apt
http://www.plymouth.ac.uk
http://www.iit.it

Abstract. The emergence of Address-Event Representation (AER) as a general communications method across a large variety of neural devices suggests that they might be made interoperable. If there were a standard AER interface, systems could communicate using native AER signalling, allowing the construction of large-scale, real-time, heterogeneous neural systems. We propose a transport-agnostic AER protocol that permits direct bidirectional event communications between systems over Ethernet, and demonstrate practical implementations that connect a neuromimetic chip: SpiNNaker, both to standard host PCs and to real-time robotic systems. The protocol specifies a header and packet format that supports a variety of different possible packet types while coping with questions of data alignment, time sequencing, and packet compression. Such a model creates a flexible solution either for real-time communications between neural devices or for live spike I/O and visualisation in a host PC. With its standard physical layer and flexible protocol, the specification provides a prototype for AER protocol standardisation that is at once compatible with legacy systems and expressive enough for future very-large-scale neural systems.

⋆ Alexander Rast, Alan B. Stokes, Sergio Davies, David R. Lester and Steve Furber are with the School of Computer Science, The University of Manchester, Manchester, UK (email: rasta@cs.man.ac.uk). Samantha V. Adams and Angelo Cangelosi are with Plymouth University, Plymouth, UK. Himanshu Akolkar and Chiara Bartolozzi are with the Istituto Italiano da Tecnologia, Genoa, Italy. This work has been partially supported by the European Union under grant nos. FP7-604102 (HBP), FP7-287701 (BrainScales-Extension), and ERC-320689 (BIMPC), by EPSRC grant EP/J004561/1 (BABEL) and by EPSRC grant EP/G015740/1 (BIMPA).

1 Neural AER Communications: The Case for Standardisation

Neuromorphic systems are entering the era of large-scale simulation. With an increasing diversity of available neuromorphic devices, it is desirable to link systems together into large, multisystem models having possibly millions of neurons. Many if not most devices use some form of Address-Event Representation (AER) [7] for communications, but there is as yet no *universal* AER standard. On the one hand it is useful to link event-generating sensors such as the various silicon retinas [8], to neuromorphic chips e.g. [6], whose event representation is a natural match, on the other to connect to very-large-scale systems in fixed locations that can provide the resource necessary for massive simulations [5], [10]. A common standard for spike exchange would facilitate such goals.

Some progress has been made, particularly with respect to the low-level hardware details of connectors, cabling, signal voltages etc. By 2009 the CAVIAR project had developed an end-to-end neuromorphic system [11], linking sensors to processors to actuators using a common hardware standard. Beginning with early work [4] to develop a hardware interface, a group at the CapoCaccia neuromorphic workshop engaged in an ongoing effort at AER standardisation and by 2013 had solved most of the hardware-level issues [9] using a simple AER protocol over Ethernet using UDP. While this work clearly shows the benefits of leveraging industry-standard protocols and hardware up to the transport level, the AER protocol itself was preliminary and did not attempt to address many of the potential use cases. What is needed is a generic AER protocol that can work over a variety of transports and support a range of different packet types. Here we develop a specification for such a protocol and a reference implementation that permits communication over a common interface between multiple heterogeneous neuromorphic systems as well as ordinary industry-standard PCs.

2 A Proposed Protocol for AER Communications

Building upon the AEtheRnet interface introduced in [9], we have expanded the protocol into a data format specification for spikes to be sent over Ethernet between compatible AER-generating and receiving devices.

2.1 Comments on Data Handling

The protocol provides considerable flexibility in data handling at the receiver. The specification does not specify the transport. In principle, any Ethernet-compatible transport could be used. Packets are transported over the wire in big-endian format. No devices acknowledge receipt of packets; the protocol is fire-and-forget. There is no built-in error detection or correction although devices could use transport facilities (e.g. CRC) to recover from errors if desired. While the interface is bidirectional, it is not required that a given device must support both directions nor that it be able to service all packet types. Devices may limit packet size to less than the maximum of 256 events (e.g. to fit the Ethernet MTU of 1500 bytes). Packets must be issued in strict event order and any spike buffering must meet any real-time spike timing constraints. Devices must drop

late spikes, spikes with timestamps out of sequential order, and unsupported packet types.

2.2 Supported Formats

The protocol consists of a 16-bit header followed by data. Communication is stateless: all the information to interpret the data is contained in the header itself. The format of the header is:

Bit															
15	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
P	F	D	T	Type		Version		Count							

The decode of the header configuration bits is as follows:

Field name	**Position**	**Value**	**Description**
P & F	15 - 14	00	Basic data packet, no address prefix
		01	Command packet
		10	Data packet with lower halfword address prefix
		11	Data packet with upper halfword address prefix
D	13	0	No payload prefix
		1	With payload prefix
T	12	0	Payload are not timestamps
		1	Payload are timestamps
Type	11-10	00	16-bit address
		01	16-bit address and payload, alternating
		10	32-bit address
		11	32-bit address and payload, alternating
Version	9-8	00	Version 0 of the protocol

The count parameter indicates how many events there are in the packet. Any transmission needs to be limited to the smallest of the supported packet sizes between the sender and the receiver. Any prefix signalled in the packet header is ORed with the value in the data part of the packet to obtain the transmitted value. The header provides for a special command packet format and 8 data type formats.

For the command packet, the bits are organized as follows:

Bit			
15	**14**	**13-0**	**Payload**
0	1	Command ID	Command and device-specific

The command ID and packet payload are device-specific.

Data packet formats have one of the following general structures:

Bit							
15	**14**	**13-0**	**Event**		**...**	**Event**	
0	0	Header Info	Address	(Payload)	...	Address	(Payload)

Bit										
15	**14**	**13-0**	**Prefix**		**Event**		**...**	**Event**	
1	x	Hdr Info	Addr Prefix	(Pay Prefix)	Address	(Payload)	...	Address	(Payload)

where fields in (round brackets) are optional. The number of events (address/payload pairs) is determined by the 'Count' parameter in the header field.

3 Implementation on a Real Platform

To implement the protocol, we chose UDP to provide the underlying transport since it is a natural fit to AER-type fire-and-forget systems, and is supported by a wide variety of AER-generating devices. As a test bed, we used two neuromorphic platforms and created a device simulator able to exercise the full feature set of the proposed protocol.

3.1 Platforms

The simulator is a host-based C++ application that includes options to configure it as a transmitter or receiver, or both, to generate patterns of events or pull them from a file, to dump received events to a file, reflect them back to the source, configure payloads and prefixes, and many other options.

One of the 'neuromorphic' platforms is a universal neural processor: SpiNNaker (described exhaustively elsewhere, e.g. [5]). The chip is an homogeneous array of cores embedded in an AER network fabric. Because of this feature, it is possible to assign cores to special-purpose roles like I/O and have them be automatically mapped and routed via the SpiNNaker configuration toolchain from a PyNN-based [2] description.

The other is a robotic platform: the iCub, of which we used both the 'normal' version and the 'neuromorphic iCub', which replaces traditional frame-based cameras at the sensory periphery with AER-generating spiking retinas. An interface board located in the iCub head translates AER events into generic Yarp (Yet Another Robotics Platform) packets that can be distributed to other Yarp ports and devices.

3.2 SpiNNaker-Side: The EIEIO Interface

The SpiNNaker implementation of the protocol uses the External Internal Event Input Output (EIEIO) interface: a series of modules loaded on cores in the chip, taking advantage as noted in 3.1 of the ability to dedicate cores to application management tasks. 3 modules comprise the interface.

Root Monitor This module resides on the core connected to the Ethernet and translates between external UDP and SDP, an internal communications layer able to transport messages around a SpiNNaker system using AER packets. The EIEIO interface uses the Root Monitor's 'SC&MP' system software to layer the external AER protocol onto SDP.

Live Packet Gatherer (LPG) This implements the spike output (from SpiNNaker) side of the interface. It records spikes from tagged populations and immediately forwards them over SDP to the Root Monitor. One LPG can record and output from several populations (although not from an arbitrary subset of neurons). A Python module compatible with PyNN enables spikes to be directed to the LPG using the form `activate_live_output_for(<Population>)`.

Reverse IP Tag Multicast Source (RIPTMS) This implements the input (to SpiNNaker) side of the interface. It is an event-triggered listener which waits for incoming SDP packets from the Root Monitor, decodes the spikes contained therein, and then injects them directly onto the internal communications fabric. A spike translation decoder translates spikes represented in a user-specified external address space into the internal SpiNNaker address space. Like the LPG the RIPTMS has a Python module which instantiates it as a PyNN Population (from which projections can be made in normal PyNN style using the SpiNNaker software stack).

3.3 iCub-side: The AER library

To provide users with an interface to the protocol without having to program low-level decoding details, we developed a general purpose C++ library and tested it on the iCub. Currently it supports only basic 32-bit data packets without prefixes, expecting timestamps in the input direction but issuing no payloads in the output direction. The library contains three main classes that are likely to be used in external programs:

SocketIF Handles socket creation for send and receive and also handshaking with the SpiNNaker toolchain for synchronisation and retrieving neuron id – SpiNNaker key mappings.

EIEIOSender Manages a FIFO queue of spikes for sending and bundles multiple spikes into a packet (max 63 spikes per packet). End users should call the addSpikeToSendQueue(int id) method in their program, passing in the neuron id as an argument.

EIEIOReceiver Manages a FIFO queue of spikes decoded from incoming EIEIO packets. End users should call the getNextSpike() method in their program to pop the spikes from the queue.

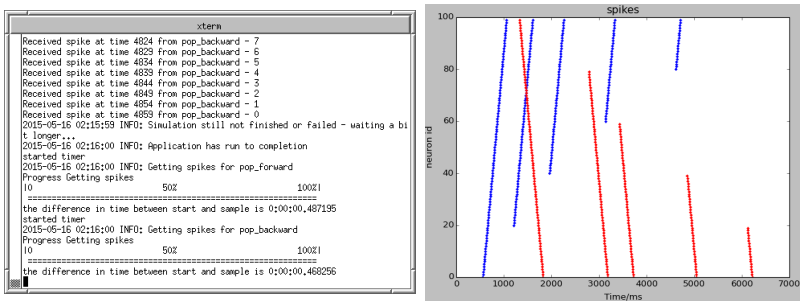
Both EIEIOSender and EIEIOReceiver have independent sockets and can be used in the same program to enable concurrent send and receive.

4 Multi-System, Multi-Modal Real-Time Operation

To test the AER interface we ran 3 different groups of experiments. The first group used the device simulator briefly described in sec. 3 to create a 'virtual device' on a laptop that could generate a variety of different I/O scenarios. The second group used the neuromorphic iCub to determine capacity limits and verify the ability to set up a connection. The third group used the regular iCub robot with the AER library to verify hardware connectivity with the new protocol.

4.1 Simulation with a Virtual Device

To verify the operation of the revised protocol using EIEIO, we tested 2 scenarios. The first scenario implements a use case where the protocol is being used exclusively for local I/O and visualisation. We configured a network with a chain of sequentially spiking neurons in 2 directions, whose input comes from a host-based spike source and whose output is redirected to a live visualisation tool. The host-based source was configured to inject a spike into, sequentially, neurons 0, 20, 40, 60, and 80 of 100 in the forward direction, and descending order (from 100) in the reverse direction, at a random interval, with weights from neuron to neuron in each population sufficient to trigger the next population in the sequence. Results (Fig. 1) show both the expected behaviour and the ability to receive spikes from an external host and display them in real-time using a live monitor, using the protocol.



(a) Console window

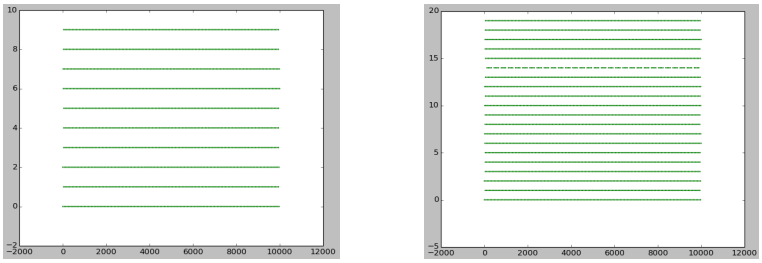
(b) Visualised raster

Fig. 1: Test of local input/monitoring facilities using the protocol on SpiNNaker

The second scenario implements the use case of communication between 2 different neuromorphic devices, possibly in widely different geographic locations. We created a spike reflector: a virtual device that would accept input from SpiNNaker and reflect it back to the chip. We then configured a network in SpiNNaker with a source population of 10 neurons and a destination population of 10 neurons, a Live Packet Gatherer and a Reverse IP Tag Multicast Source, and specified the connectivity so that the source sent to the virtual device and the destination received from the virtual device, each neuron receiving one connection with weight high enough to cause it to spike. We tested this both with a direct network connection and through a routed network (by connecting the SpiNNaker chip to a public network). Fig 2 shows the results: with the virtual device off only the source population spikes; with it on, both populations spike and all spikes are received.

4.2 Hardware Operation with the Proposed Interface

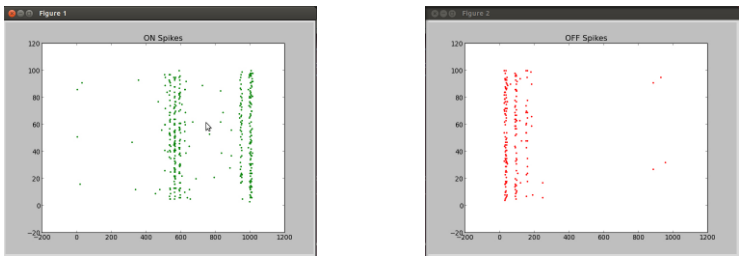
As a preliminary to protocol deployment, we ran a series of tests with the neuromorphic iCub and SpiNNaker using an enhanced version of the AEtheRnet interface with the



(a) Virtual device off (b) Virtual device on

Fig. 2: External input/output using the device simulator

same packet size and arrangement as that of the new protocol using the Basic packet format with 32-bit addresses. We connected the systems through a host that converted spikes to Yarp bottles. We swept a bar-shaped object, (a power strip, in fact) long enough to occupy the entire visual field in one dimension, in horizontal and vertical directions of both sweep and bar orientation, and had the robot move its gaze in the direction of the bar when the bar was in a preferred orientation. Results (fig. 3) demonstrate the basic feasibility of a real-time link using Ethernet and indicate a peak spike rate of about 8K spikes/s.

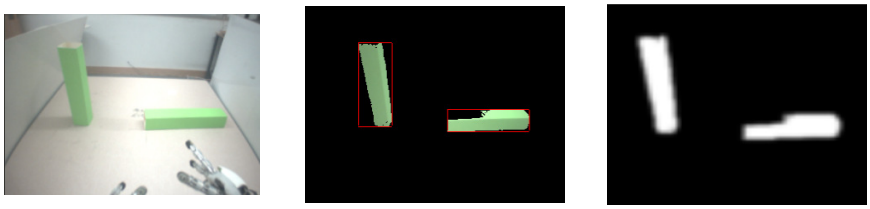


(a) Onset pixels (b) Offset pixels

Fig. 3: Data from DVS sensor on iCub, as seen on SpiNNaker, horizontal object

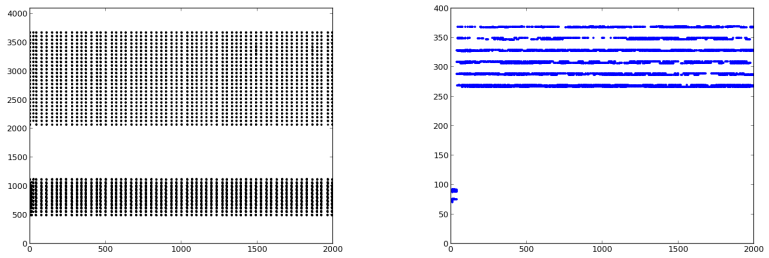
To test the new EIEIO protocol (and the AER library described in Section 3.4) in a real-world scenario, scaled up versions of the attentional network used in [1] were run on a 48-chip SpiNNaker board interfacing to the regular iCub robot. We used two sizes: 32x32 input layer size (6,148 total neurons) and 64x64 input layer size (24,592 total neurons). The scenario was as used for previous work, with two objects, of different orientations, one ‘preferred’, one ‘aversive’, the task being to cause the iCub robot to look/point at the ‘preferred’ object. Examples of the raw and processed input camera

images are given in Fig. 4. Examples of the input and output network spiking activity for the largest network tested are given in Fig 5.



(a) Raw camera input (b) Saturation view (c) Binary view

Fig. 4: Camera input from the iCub robot



(a) Input layer spikes (b) Output layer spikes

Fig. 5: Spiking activity for the largest network run with the iCub robot

This generated spike rates of about 15K spikes/s in the input layers and 2K spikes/s in the output.

5 Discussion: Potential and Challenges

5.1 Current Work: Immediate Implications

The new EIEIO protocol yielded several improvements to the attentional network implementation in [1]. Throughput management was much easier using the AER library as spike packet construction is handled automatically with only minimal iCub host interfacing code required. The EIEIO protocol also handled SpiNNaker key generation and neuron id to key mapping automatically, reducing the possibility of errors.

One obvious use of the interface is to connect several AER-generating or receiving devices. But equally, since the interface provides a general mechanism for getting spikes in and out of a chip, with a general-purpose device such as SpiNNaker, it can be used as a means to control and monitor a simulation occurring on a single platform. Tests of this facility worked successfully and indicate that a universal AER interface not only permits direct inter-device connections but also provides a channel for real-time simulation management. In future such interfaces could be used for loading of configuration data for neuromorphic chips, possibly remotely.

The protocol specifies the ability to use timestamps but does not indicate a definitive reference for the meaning of timestamps to a given device. In general, asynchronous neuromorphic chips operating in possibly remote locations could have large differences in their local clocks, and possibly different time units. Although the interface cannot guarantee perfect synchronisation, command packets offer a solution that allows for a form of approximate synchronisation by exchanging timing messages.

5.2 Next Steps: Future Work

Experiments with the current protocol revealed that while functional, it could benefit from some additional features, and at the same time some of the more complex features could be streamlined in their definition. We are currently revising the proposal to allow packets containing both a timestamp and an additional payload, and to allow stateful transceiver interfaces supporting a user-defined link setup to describe things such as compression or time representation using a handshake with command packets. If reliable packet reception is required the protocol may also be implemented over TCP.

6 Conclusions

This work grew out of the earlier AEtheRnet protocol and a desire in the community to establish standards for remote neuromorphic communications. The earlier protocol established some basic properties of such a possible standard; this new work indicates some additional important features, which can be considered as the main contributions:

Transport Independence:
The protocol is not bound to a particular underlying transport.

Payload Support:
Packets can contain payloads and timestamps as well as addresses.

Partial Support Allowed:
Device designers can choose what packet formats they support and drop others.

Idempotent Packets:
Receivers are not required to have stateful interfaces.

With these features the protocol can support both simple and complex existing AER devices in a variety of use cases while suggesting the form of a built-in (on-chip) interface for future neuromorphic chips. Paired with an appropriate higher-level software interface (e.g. MUSIC [3]) which would allow ready exchange of spikes between heterogeneous simulators, this system would allow for one unified communications layer

making interlinking and using spiking neural simulators as simple as, for example, using stream-based I/O in C/C++. While the high-level API still remains to be determined, we have developed an underlying base representation for these 'neural streams'.

Acknowledgements

This work has been partially supported by the European Union under grant nos. FP7-604102 (HBP), FP7-287701 (BrainScales-Extension), and ERC-320689 (BIMPC), by EPSRC grant EP/J004561/1 (BABEL) and by EPSRC grant EP/G015740/1 (BIMPA).

References

[1] S.V. Adams et al. "Towards Real-World Neurorobotics: Integrated Neuromorphic Visual Attention". In: *Proc. 21st Int'l. Conf. on Neural Information Processing (ICONIP 2014)*. Vol. 3. 2014, pp. 563–570.

[2] A. P. Davison et al. "PyNN: a common interface for neuronal network simulators". In: *Frontiers in Neuroinformatics* 2.11 (Jan. 2009).

[3] M. Djurfeldt et al. "Run-Time Interoperability Between Neuronal Network Simulators Based on the MUSIC Framework". In: *Neuroinformatics* 8.1 (Mar. 2010), pp. 43–60.

[4] D. B. Fasnacht, A. M. Whatley, and G. Indiveri. "A Serial Communication Infrastructure for Multi-Chip Address Event Systems". In: *Proc. 2008 IEEE Int'l Symp. Circuits and Systems (ISCAS2008)*. 2008, pp. 648–651.

[5] S. B. Furber et al. "Overview of the SpiNNaker system architecture". In: *IEEE Trans. Computers* 62.12 (Dec. 2013), pp. 2454–2467.

[6] G. Indiveri, E. Chicca, and R. Douglas. "A VLSI Array of Low-Power Spiking Neurons and Bistable Synapses With Spike-Timing Dependent Plasticity". In: *IEEE Trans. Neural Networks* 17.1 (Jan. 2006), pp. 211–221.

[7] J. Lazzaro et al. "Silicon Auditory Processors as Computer Peripherals". In: *IEEE Trans. Neural Networks* 4.3 (May 1993), pp. 523–528.

[8] P. Lichsteiner, C. Posch, and T. Delbrück. "A 128 $\times$ 128 120 dB 15 μs latency asynchronous contrast vision sensor". In: *IEEE J. of Solid-State Circuits* 43.2 (Feb. 2008), pp. 566–576.

[9] A.D. Rast et al. "A Location-Independent Direct Link Neuromorphic Interface". In: *Proc. 2013 Int'l Joint Conf. Neural Networks (IJCNN2013)*. 2013, pp. 1967–1974.

[10] J. Schemmel et al. "A Wafer-Scale Neuromorphic Hardware System for Large-Scale Neural Modeling". In: *Proc. 2010 IEEE Int'l Symp. Circuits and Systems (ISCAS2010)*. 2010, pp. 1947–1950.

[11] R. Serrano-Gotarredona et al. "CAVIAR: A 45k Neuron, 5M Synapse, 12G Connects/s AER Hardware Sensory-Processing-Learning-Actuating System for High-Speed Visual Object Recognition and Tracking". In: *IEEE Trans. Neural Networks* 20.9 (Sept. 2009), pp. 1417–1438.