IAC–24–B2.7.2

Networking with Dynamic Reconfigurability and Robustness for Modular Spacecraft

Mark A Post, David Grace
University of York, Heslington, York, England, YO10 5DD, mark.post@york.ac.uk; david.grace@york.ac.uk

James White
4Links Limited, 12 Shenley Pavilions, Chalkdell Dr, Shenley Wood, Milton Keynes, England, MK5 6LB, james.white@4links.space

Cameron Anderson, Murray Ireland, Georgia Harvey
Craft Prospect Ltd, Suite 12, Fairfield, 1048 Govan Rd, Govan, Glasgow, Scotland, G51 4XS, murray@craftprospect.com

Modular and dynamically reconfigurable space hardware systems are a high priority for research and development as they can significantly decrease development costs, facilitate multiple payloads, and enable indefinite lifetime extension for orbital assets through on-orbit servicing operations. However, dynamically reconfigurable space hardware systems have many fundamentally different design and implementation requirements from monolithic single-mission systems that are currently in use. Most critically, the communications between modular and reconfigurable components must be decentralized, robust to changes in physical organization, common to all components at the physical, electrical, and protocol levels, and able to accommodate new elements beyond the original mission specification.

This paper describes adaptable and robust networking technologies that are currently under development to support novel modular satellite and space robot systems that implement dynamic reconfiguration and autonomy. While it is desirable to adapt existing communications technologies to the use of modular spacecraft, a review of these technologies indicates a clear capability gap between current technologies and the needs imposed by autonomous reconfigurability.

To fill this gap, we describe a novel communications protocol that can be used over existing Ethernet hardware and CAN bus for wired communications that provides the reconfigurability and robustness needed for data and power networking of changing physical configurations of hardware. We also describe the extension of this protocol to wireless systems that could be deployed safely on space hardware and allow spacecraft modules to communicate between each other and with ground stations when not physically connected.

For robust fault tolerance and connectivity with ground stations, it is necessary to have the capability to recover from network degradation due to both internal system faults and external factors such as atmospheric conditions and cloud cover. We have created a Machine Learning (ML) application capable of interfacing with switch nodes in an emulated network to record traffic, predict link health using ML, and then re-route traffic to a more optimal path.

Decentralized adaptive routing and ML-based routing test results from network emulators have showed improvements in throughput from re-routing following fault injection. Improvements were also seen in robustness to network changes and latency of fault response compared to traditional networking solutions, and the nodes successfully used local autonomy to recover from dynamic reconfiguration faults. Future development will include implementing these protocols and ML technology in a fully representative space hardware network, with distant and autonomous modular satellite nodes.

I. Introduction

The rapid development and exploitation of space in a sustainable and cost-effective manner necessitates the development of new methodologies for constructing, maintaining, and managing the lifetimes of space hardware and software systems. Satellite spacecraft, deep space probes, and planetary rovers have previously been largely designed and deployed as one-off systems, with a finite lifetime and limited or no servicing and lifetime extension capability. Projects such as PETSAT,[21] iBOSS,[13] MOSAR,[15] HIVE[12] , CSR[4] and the modular space warehouse STARFAB[6] have proposed and explored modular ecosystem concepts for robotically-assembled, reconfigurable modular satellite hardware that could overcome these limitations and allow future missions to be constructed, serviced, and re-deployed.

Modular systems in general are critically reliant on the availability of ubiquitous communications between elements. Numerous proprietary spacecraft buses with purpose-built communications are used by separate manufacturers, but vendor-independent communications in modular space hardware systems is generally provided by SpaceWire and SpaceFibre as well as safety critical buses such as CAN,[26] Time-Triggered Ethernet (TTEthernet), and EtherCAT[19] being proposed. The use of Commercial Off-The-Shelf (COTS) hardware for space hardware buses has been considered apace as new terrestrial technologies are developed, with buses such as Inter-Integrated-Circuit (I^{2}C) and IEEE 1394 "Firewire" being recommended for space use, but under the stipulation that either mandatory changes or "implementation options" coupled with design practices such as redundancy provide the necessary robustness[22].[5] However, for autonomously reconfigurable modularity, such communications systems must be sufficiently robust for harsh environment use, designed in such a way as to allow future expansion to support connection to new functionality that is not available at the time of launch, and also support locally intelligent dynamic routing, plug-and-play style resource discovery, and protection against unexpected faults and security threats.[18] Fulfilling these requirements for modular space hardware is still an open problem for currently-deployed communications hardware and protocols.

The use of wireless communications in space for servicer spacecraft proximity operations, interaction with disconnected free-flying modular hardware, or to work around faults in a physical communications system is another critical consideration in enabling modular ecosystems to operate robustly and autonomously. Radio communications for long-range links to ground stations, and more recently other satellites in peer-to-peer configurations, have been well explored for purposes such as multichord Internet of Things (IoT) networks, with satellite IDs hashed from orbital elements[11] and CubeSat-based Internet extensions to remote areas with mixed deterministic and probabilistic links.[9] However, the use of short-range dynamic wireless networking between large numbers of autonomous modular elements has been largely unexplored for use in space, despite wireless mesh networking being a mature terrestrial technology. The core focus of our ongoing research described in this paper is to fill this gap in capability and equip modular space hardware with suitable dynamically-reconfigurable wired and wireless communication capability.

II. Background

II.i Networking Requirements

A "Space Internet" or "Interplanetary Internet" for networking space hardware to automate mission communications is superficially feasible, but in practice introduces many complications that terrestrial Internet technologies were not designed to overcome, primarily:

- Long link delays due to vast distances, more so in the case of deep space links;
- Frequent and lengthy link outages due to environmental and operational hazards;
- High channel and Bit Error Rates (BER) due to the harsh environment;
- Out-of-order and non-real-time arrival of data due to delays and error rates
- Asymmetric channel rates between spacecraft and ground links.

While these problems are less significant for the short range communications that are most common for modular spacecraft systems, achieving a scalable and universal networking capability with long-distance ground and interplanetary links still necessitates their consideration in system design. In particular, these problems make the required dynamic connections and routing for reconfigurable systems very difficult to achieve reliably and in short response times. Also, using an asymmetric "patchwork" of protocols will drive up cost, complexity, and the number of potential failure modes while decreasing designed compatibility between systems and should be avoided. It is desirable to keep communications as simple as possible, to increase reliability and minimize failure cases:

- Simple hardware based on COTS technology is cheaper, more accessible, and often more mature;
- Reducing protocol and routing complexity reduces the number of potential failure modes in the system;

- Ensuring that the system is introspectable, explainable and easily diagnosed increases maintainability;
- Less layers of abstraction and logic increases the response speed of the system to changes
- Keeping the protocol as "universal" as possible simplifies compatibility across both physical domains and temporal old/new versions of systems.

For a wired system, dynamic reconfiguration and routing are essential to cope with arbitrary changes in topology. For a wireless system, peer-to-peer mesh networking is at present the most appropriate topology as decentralization increases robustness for systems that may not be in close proximity to a functioning master device.

II.ii Communications Hardware

Modular spacecraft buses to date have been predominantly physically connected, and based on existing terrestrial and spacecraft interfaces. Previous research has recognized the value of modular and reconfigurable robots for use in space hardware. Yim et al. focused on three characteristics of modular robots that were advantageous: Compactness and Lightness, Robustness, Versatility and adaptability.[28] However, appropriately dynamic communications systems have not yet been developed for space use.[18] Considering instead the prototypes of modular robotic systems developed for terrestrial experimentation, a wide variety of docking systems and communications systems have been developed and tested.[25] Since CEBOT was introduced by Toshio Fukuda and Yoshio Kawauchi in 1990,[10] the majority of modular systems have used wireless infrared serial communications, wired UART-based serial streams, wired Controller Area Network (CAN) bus, and Bluetooth wireless communications, as shown in 1. While these systems are well established and inexpensively accessible in terrestrial communications systems, they have generally been chosen for convenience and efficiency rather than high performance and robustness in adverse conditions, with data rates generally below 1Mbps. However, many modern satellite and space robot sensor packages have much higher data rate requirements.

IEEE 802.3 based Ethernet and IEEE 802.11 based Wireless Ethernet have much higher data rate capacities, but the only example of wired Ethernet in modular robotics was found in CoSMO, part of the Symbricator project.[16] The Ethernet physical layer has many desirable properties for robust networking systems: it supports data rates well above 10Mbps, uses

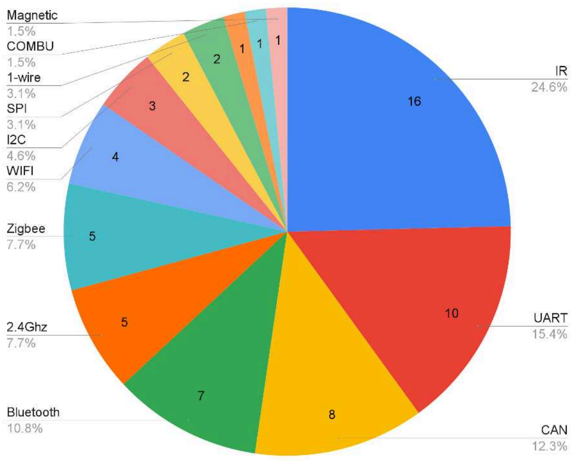

Fig. 1: Percentage use of terrestrial communication systems in modular robots[24]

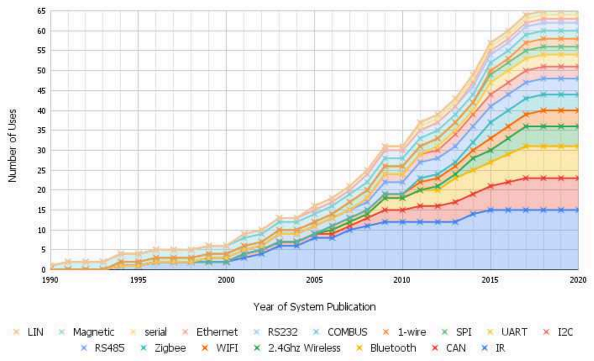

Fig. 2: Changes in the use of communications technologies in modular robots over time

EMI resistant differential pairs, is galvanically isolated, and uses readily available off-the-shelf components and wiring solutions. These features have made Ethernet a ubiquitous networking technology for terrestrial computing systems, and are an ideal basis for modular networking systems. SpaceWire itself is the only comparable alternative, but is in most cases reliant on rare and expensive hardware for the physical layer. The main disadvantages to be overcome are the complexity of the software stack required, and the reliance on static addressing and routing (a limitation shared with SpaceWire). As Figure 2 shows, there has been a steady movement towards more complex wireless communications such as Wifi and Bluetooth over time. However wired serial connections are still preferred in many cases.

II.iii Networking Technologies

Ubiquitous data transportation throughout the Space environment has been explored frequently through adaptation of terrestrial networking protocols, mainly Transfer Control Protocol/Internet Protocol (TCP/IP) which is used for most Internet traffic. The majority of these approaches have been classified by Wang et al. into protocols that simply build on TCP, protocols that modify both TCP and the underlying network hardware infrastructure, and novel protocols developed independently from TCP.[23] While TCP and its connectionless counterpart UDP (User Datagram Protocol) are excellent examples of protocols that have maintained near-universal compatibility between Internet devices for decades, they were also designed for a very different set of requirements and constraints than modern robots and space hardware devices have.

Communications channels in space typically have long propagation delays (less than a second to orbit, but up to several hours for interplanetary links), higher bit error rates (from 10^{-5} for intersatellite links to 10^{-1} for interplanetary links), and bandwidth asymmetry between uplink and downlink channels typically on the order of 1:1000.[1] TCP is designed under the assumption of very low error rates and uses a congestion window mechanism for avoiding packet collisions, which wastes significant amounts of link time and misinterprets any data loss as a congestion loss, and demands a retransmission if losses or errors are detected, resulting in long delays from small errors. This also makes TCP undesirable for use in cyber-physical and real-time systems in general. While TCP can be re-designed to some degree to mitigate these problems, the protocol will generally no longer be compatible with terrestrial TCP networks, necessitating the division of the network into multiple separate transport connections, and subsequently increasing complexity and potential modes of failure. Since this limits the value of re-using the TCP design, and it is not ideal for robotic use in any case, we focus in this work on novel communications concepts that are not based on TCP/IP. It is assumed that connecting these networks to existing Internet infrastructure will unavoidably involve a dedicated gateway. However, this does not prevent re-use of existing terrestrial hardware technology such as IEEE 802.3 physical interfaces that may already be suitable for space communications systems.

There is general agreement on a fundamental set of technologies required for space networking that can be derived from this previous work. Foremost among these are the concept of Delay-Tolerant Networking (DTN). DTN is a classification for protocols that provide higher reliability and performance for networks that experience the complications and problems described in the space environment, mainly characterized by the long and frequent delays in transmission that result. The main feature of DTN networks is usually the replacement of TCP/IP's immediate forwarding and anti-congestion and retransmission-on-error mechanisms with a store-and-forward information routing model, in which a copy of all information sent over the link is cached at each routing step, stored long enough to ensure that the data reaches its next destination as confirmed by an acknowledgement (ACK), and is used as a fallback in case retransmission is required.[3] This kind of protocol is well suited to robotics and real-time systems and makes the network far more robust to point failures and transmission delays. The Licklider Transmission Protocol (LTP) is one of the most mature DTN protocols, and uses selective block transmission mechanisms to ensure that complete datasets are transferred efficiently over channels with large BER, delays and channel asymmetry. IT has been modelled with respect to performance in deep space networks[27] and has several open-source implementations, particularly the Interplanetary Overlay Network (ION) developed by Ohio University and the NASA Jet Propulsion Laboratory.[14] Drawbacks of the protocol itself include that it is designed only for point to point links (not networks) and lacks congestion control, but these features can be added to the protocol itself or the software stack implementing it. Achieving a "single" protocol for end-to-end networking across modular, orbital, planetary, and interplanetary scales is often considered infeasible, as any of the networking complications listed above may occur at any link in this multi-scale chain of connections. One proposed solution to this is the use of Bundle Protocol (BP) as a store-and-forward "overlay" for custody-based delay-tolerant retransmission of a variety other link-specific protocols.[20] Although the LTP and BP approaches overlap in terms of storing information in DTN, it is intended to "bundle" LTP as well as TCP and UDP through the use of a Convergence Layer Protocol (CLP) compatibility layer.

Efficient mesh networking systems have also been developed for spacecraft constellations flying in formation, which are closest to the use case of a modular satellite system. A Mesh network architecture using a Time Division Multiple Access scheme was developed for small satellite constellations at NASA Marshall Space Flight Center.[2] TDMA has the advantages of saving power by turning off the transceiver during sleep periods in an active frame, and being implementable on a wide variety of serial communication radio hardware by moving network management into software. The system was tested using XBee Pro 2.4GHz radios, then on an AstroDev Li-1 UHF radio, then on a formation of 6 quadcopters using XBee radios with GPS. While this network does not provide all the functions of a robust DTN such as store-and-forward routing and bundling, it is a very simple and flexible transport for serial data in a mesh network.

II.iv Data Representations

Modular space hardware represents the extreme of hardware compatibility challenges, as modular hardware placed in space may be required to maintain compatibility with other systems engineered and deployed potentially decades after its original launch, without the possibility of detailed re-engineering work to maintain compatibility with evolving standards. As such, schema-based protocols and data serializers that are artificially limited to use of a fixed ontology or dictionary of messages are a bad fit to such systems. It is desirable to have the data payload of DTN networks flexible enough to carry any future message type, including variable-size data packets. Self-describing semantics and also simplicity in the network protocol and routing also contributes to future-proofing by not over-constraining the uses of the network. However it may also make interpretation of messages more complex. It is also important that heterogeneous physical layers can be networked using a single protocol as translation of data between multiple protocols can increase complexity immensely due to the number of translation permutations needed, significantly increasing processing power required and the number of failure modes.

Schema-free and self-describing communications (dynamically interpretable semantics) means that many strongly-typed data formats that rely on fixed schemas and message types should be avoided, for example ASN.1, Protocol Buffers, Cap'n Proto, flatbuffers, and most uses of XML. XML schemas such as XML Telemetric and Command Exchange (XTCE) are considered to be XML for comparison purposes, since the overheads for translation to binary and the limitations in dynamic expansion f the schema still exist. In the case of many transformations such as that used in ASN.1, there is a significant encoding overhead incurred when converting a memory representation of a message into a serial stream, as was made clear in the implementation of the InFuse Common Data Fusion Framework using TASTE[7].[8] MessagePack is an example of a serializer without the requirement for a schema and uses a very efficient serialization transformation. However, there is still a serialization overhead when encoding arbitrary data into the data stream. The most efficient solution is to make the memory storage format of the information to be transferred equivalent to the serialization format when sent through communications channels, as is done in the Cap'n Proto serialization. This requires consideration of the serialization of data at low levels in the communications software stack and potentially inclusion of the serialization mechanism within the transport layer of communications. Isomorphism to convert data to and from markup languages such as XML is also desirable given the dependence of many data management systems on human-readable storage formats, and for facilitation of debugging and system explainability to operators. AXON eXtended Object Notation and Rusty Object Notation (RON) with Serde XML are formats that facilitate this capability. Currently, Msgpack and Cap'n Proto are used for serializaton, and further investigation of the suitability of Flatbuffers and Dhall will also be done. A comparison chart of serialization protocols is shown in Table 1.

III. Modular System Prototype

III.i Modular Hardware

As a research platform for reconfigurable modular cyber-physical systems, we have developed a heterogeneous modular robot platform based on a cubit lattice topology that can be assembled into a variety of structures through a standardised docking mechanism that provides mechanical and electrical connection between modules. We have named this platform Mo* due to the wide range of modular technologies that it can serve. The Mo* modular robot platform has been developed to meet the following high level goals:

1. Hardware based on a 10cm cubic unit of volume
2. Heterogenous modules in different form factors possible using a common docking system
3. Design capacity to include arbitrary payloads including actuators, power, and advanced computation
4. Flat surfaced genderless docking system with single sided disconnect

Protocol	No Schema	Dynamic Size	Dynamic Types	Semantics	No Copy	Efficient	Readable
XML	Optional	Yes	Yes	Yes	No	No	Yes
YAML	Yes	Yes	Yes	Yes	No	No	Yes
ASN.1	No	No	No	No	No	Yes	As Code
AXON	Yes	Yes	Yes	Yes	No	No	Yes
RON	Yes	Yes	Yes	Yes	No	As Rust	With Serde
JSON	Yes	Limited	Limited	Yes	No	No	Yes
Protobuf	No	Yes	Yes	Yes	No	Yes	Yes
Msgpack	Yes	Yes	Yes	Yes	Nearly	Yes	Yes
Cap'n Proto	No	Yes	Yes	Yes	Yes	Yes	Yes
Flatbuffers	Optional	Yes	Yes	Yes	Yes	Yes	Yes
Dhall	Programmable	Yes	Yes	Yes	Yes	Yes	Yes

Table 1: Caption

5. Capability to transmit both power and data between modules

The physical module enclosures are simple in design and 3D printed from PLA for laboratory use as shown in Figure 3, but could similarly be constructed of more robust and environmentally tolerant materials. A cubic module is composed of six "tiles" that form the sides of the cube, and can contain electronic or mechanical hardware to perform part of the module's function. Four pairs of magnets with spring-loaded pogo pins behind them are currently used in the docking interfaces to ensure a strong connection between eight separate conductive contacts. In configurations where magnetic fields are undesirable, the magnets can be replaced with spring contacts and retracting mechanical latches, some initial designs of which have been prototyped for Mo*. Single-sided disconnect and mechanical unlatching are accomplished in active interfaces by rotating opposing polarity pairs of magnetic contacts such that alignment is reversed, pushing the interfaces apart physically. A micro metal gear motor rotates all four pairs simultaneously using a spur gear arrangement as shown in Figure 4.

A complete module is assembled as shown in Figure 5. Tiles need not all have active interfaces, nor any interfaces at all in the case of sensor or communications tile requiring a clear surface. Passive interfaces as shown in Figure 3 with static, unactuated magnets or latches can also be used to lower cost and complexity of modules without preventing an attached active interface from using single-sided disconnection although at least one interface must be active in order to achieve controlled mechanical connection and disconnection. The differing properties of each side of each module must be taken into account in the use of planning algorithms for assembly

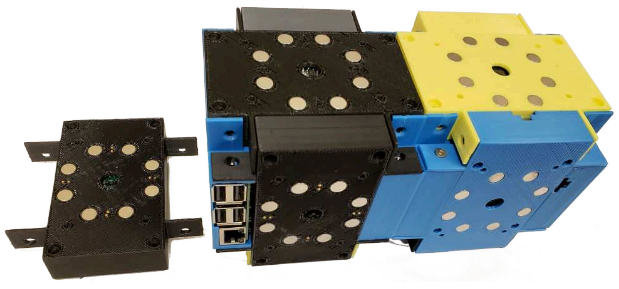
Fig. 3: 1U (10cm) prototypes of Mo* sub-modular spacecraft hardware[17]

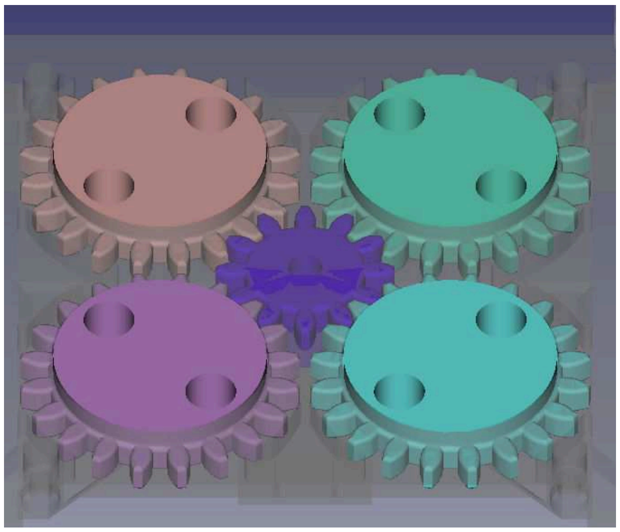
Fig. 4: Arrangement of gears in an active module docking interface[24]

of modular structures.

III.ii Physical Communications Interfaces

Following a review of mature and commercially-used hardware as a basis for building suitable communications hardware for harsh environments, The IEEE 802.3 100Base-TX Ethernet standard was chosen as the basis for modular networking in Mo*. This allows the interface to operate with only four connector wires, unlike 1000BASE-T which requires eight, and makes it possible to use the industry standard Reduced Media Independent Interfaces (RMII) and high-speed Medium Access Control (MAC) hardware which is the fastest interface for many commercial microcontrollers. Inexpensive and easily available MAC hardware can be used for essential galvanic isolation of the conductive contacts between modules. The complexity of Ethernet hardware and multiple-device routing topologies, not used on most modular robots, is seen as a necessary expense in this case, as bus topology communications such as RS-232 RS-485 and CAN bus have the drawback that a single faulty device on the bus can cause the entire bus to become unusable. Isolating each communications link between modular elements using dynamic routing of data and power, so that redundant routes around faulty links and hardware can be formed, is a critical benefit for modular systems and is the main feature of the modular communications system on Mo*.

The processing hardware used in each module for this communications research is built around a Xilinx Zynq 7020 System-on-a-Chip, which contains two arm processing cores and Artix-7 FPGA fabric, assembled on a Trenz Electronic TE0720 System on Module. This forms a communications ”master” tile shown in Figure 6 one of which is required in each module for routed bus communications through multiple modules.The FPGA fabric on the device is utilised for networking protocol implementation and connecting to the 6 Ethernet physical layer transceivers. The Texas instruments DP83825 Ethernet PHY is used for this. Each transceiver then connects to a corresponding tile in the module via Molex PicoBlade pin headers. The tile interface contains an IEEE 802.3 transformer IC which then connects to the tile magnets. In addition to this Ethernet data bus, Controller Area Network (CAN) is used within modules as a secondary low-bandwidth system management data bus. The CAN bus is isolated and not used externally as it is a multidrop bus topology and any electrical faults or external connections may disable it for all hardware within the module.

Table 2: Pinout of Power, 802.3 and CAN on internal PicoBlade connectors

TD _ P	TD _ N	GND	VBus	CANH	CANL	VBus	GND	RD _ P	RD _ N

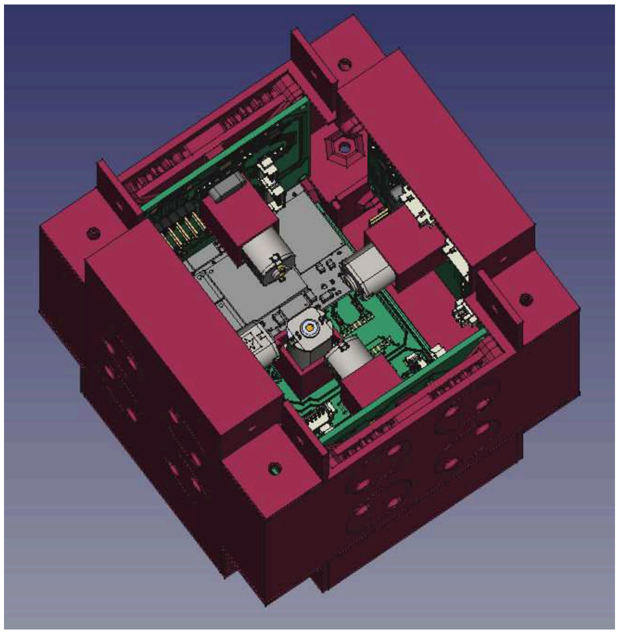

Fig. 5: 5 tiles assembled into a 1U cube with all interfaces actuated[24]

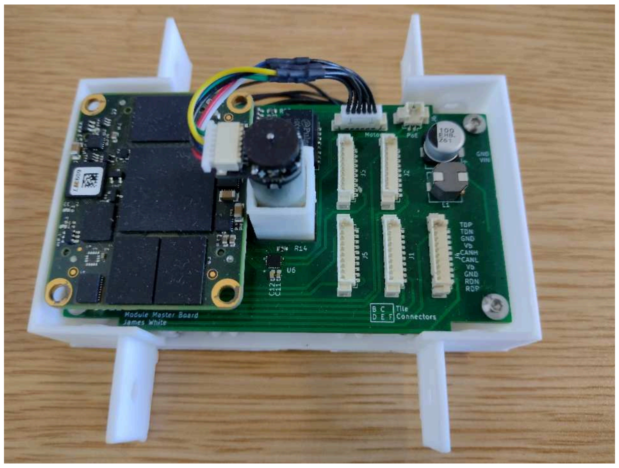

Fig. 6: Master tile using Xilinx Zynq System-on-Module and 6-port Ethernet-based switch implemented in FPGA[24]

An additional benefit of using Ethernet PHY technology is tolerance to connector contact permutations via automatic Media Dependant Interface (MDI) crossover detection. Any modular interface with rotational symmetry that allows connections at multiple angles, such as a cube face with 90° angles of symmetry, will cause a different permutation of contact connections between modules when rotated. The most common solutions to this are either to use geometrically invariant contacts such as concentric rings, or to duplicate all contacts at each angle of symmetry, both of which cause complications and restrictions on design. IEEE 802.3 Ethernet physical layers can detect whether a direct connection to a hub or switch with TX and RX contacts crossed over (MDI-X) is used, or connection to another identical interface (MDI), and also if differential pairs themselves are crossed over. Mo* interfaces use 8 contacts duplicated once for redundancy, and arranged in differential pairs as per IEEE 802.3 standard. Module docking causes TX and RX contacts normally crossed over (MDI-X) when they are connected front-to-front at 0° or 180°, but cross over differential pairs when connected at 90° or 270°. This can be automatically detected by the PHY without disruption to communications, and can also be used to help detect the orientation of connected modules internally.

Ethernet PHY Interface

State machines were implemented for handling transmission and reception on the RMII interface. There are 4 main state machines operating. One for generating the di-bit stream and handling the TX control signal on the RMII interface from byte-wise data, one for receiving the di-bit stream on the RMII interface and converting it to byte-wise data, one for frame transmission, and one for frame reception. The state transition diagram for the frame transmission state machine is shown in Figure 7. The state transition diagram for the frame reception state machine is shown in Figure 8

MDIO Interface

The MDIO (Management data input/output) interface is used for accessing the register space of the Ethernet PHYs, as defined in the IEEE 802.3 standard. It is a serial bus consisting of a clock and one data signal line. The data line is bi-drectional. The MDIO standard defines a packet structure to use to communicate with PHYs. The preamble is 32 bits long, consisting of repeating ’1’. ST is the start field that is two bits long and always ”01”. The OP field is the OP code which denotes whether the packet is

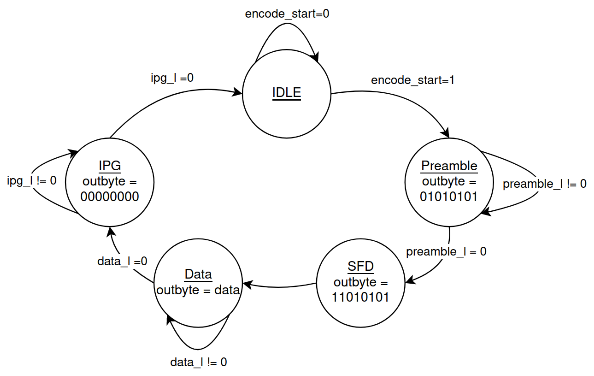

Fig. 7: State transition diagram for the state machine that handles RMII frame transmission.

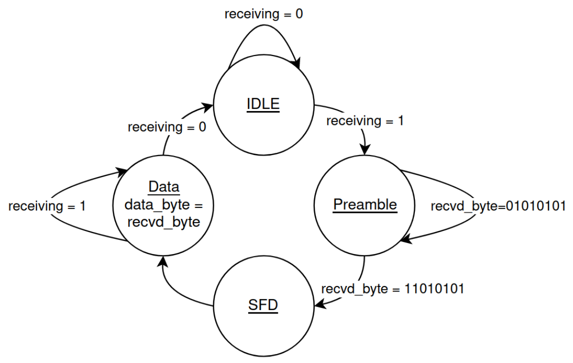

Fig. 8: State transition diagram for the state machine that handles RMII frame reception.

a read (”10”) or write (”01”) operation. PHYADDR is the address of the PHY to be accessed. The standard allows up to 32 phys to share the same mdio bus. REGADDR denotes the register to be read from or written to. TA is a turnaround field. When data is being written to the PHY, the MAC writes ”10” to this line. When data is being read, the MAC releases the data line for reception. DATA is a 16 bit data field for reading from or writing to the register space at the address specified earlier in the packet. In total a packet on the MDIO line is 64 bits long. The MDIO clock can be user defined, based on the chip being used. The DP83825 chip used in this system allows a maximum clock rate of 25MHz on the MDIO bus. The DP83825 PHY only supports up to four PHYs sharing an MDIO bus, so two seperate MDIO buses had to be used in this system.

Motor Interface

The motor that actuates the master tile is driven by the DRV8837 Low-voltage H-bridge driver from Texas Instruments. This has two motor control inputs and a sleep input. There is a quadrature encoder attached to the motor which has two outputs. The absolute position of the tile connector is obtained by a DRV5055 hall effect sensor that provides an analog output proportional to the observed magnetic field strength at a normal to the top of the device. All of the digital signals are electrically routed to the FPGA portion of the Zynq device and then internally routed to the EMIO peripheral of the Arm microprocessors. This allows direct control of the motor and reading of the quadrature encoder from the processor portion of the Zynq device. The analog output from the hall effect sensor is routed to an XADC peripheral on the Zynq device, and can be monitored from the ARM processors through the AXI interface. At power on, if the connector is not connecting to another module at present, the tile controller initiates a homing sequence where the connector rotates 1 full revolution counted by the number of steps from the encoder. The analog input from the hall effect sensor is also sampled at this stage, and where a positive peak forms is taken as the home position, as this is where the south facing side of a magnet is passing over the sensor. The connector then goes to its home position.

CAN interface

The CAN interface is implemented as an embedded CAN peripheral on the Zynq device. This interfaces with the ARM processing cores on the device directly. Xilinx provides a library for interfacing with the CAN peripheral. This library provides functions to set up the CAN device, pass data to a TX buffer, and receive data from an RX buffer. The CAN interface is primarily used to allow the master tile to control the docking connectors on each of the slave tiles. The master tile sends a message to the slave

Slave Tiles

The embedded software on the slave tile is primarily responsible for communicating with the master tile via the CAN bus and controlling the motor that actuates the tile docking connector. There is a CAN peripheral built in to the microcontroller. The device supports a bit rate of up to 1Mbit/s. Motor control is achieved using the same devices as on the master tile. At start up the microcontroller homes the connector, sweeping the connector through one full rotation while keeping track of encoder steps and sampling the hall effect sensor that is connected to one of the analog inputs on the device. After this the home position of the connector can be determined and the controller moves the connector to that position. After this, the controller sits idle waiting for a CAN message from the master tile to initiate any other actuations that are required. The slave tile board also includes a 12 pin GPIO extension connector. This has 10 GPIO connections and two power connections. Combinations of these pins can be used as digital IO, Analog IO or SPI,UART or I2C communication. This provides flexibility in designing extension boards for the slave tile.

IV. Mo* Network Protocol

IV.i Data Format

The hardware layer of the networking system is built around IEEE802.3 compliant Ethernet PHY transceivers. As such it is required to use the standard Ethernet frame format as the base data unit of the system. An Ethernet frame consists of a preamble that is used to ensure that the two connected PHYs at either end of the MDI link are synchronised. This is a stream of 7 bytes of ”10101010”. Following this there is a start of frame delimiter byte of ”10101011”. After this there is a field containing the destination MAC address followed by the source MAC address. These are each 6 bytes long. The MAC address is a unique identifier of the Ethernet MAC. After this there is a field containing the length of the frame, specified in the standard as a maximum of 1500 bytes for the data field. Finally there is a cyclic redundancy check field which is used to confirm that a packet has been successfully received. The structure of the packet is outlined in table 3.

On top of these Ethernet frames, the Mo* protocol packets are used as the packet for data transmission around the network. The aim of this protocol was to implement a system that is as simple as possible. Keeping the packet header to a minimum increases the rate at which usable data can be propagated around the network. It is the hope that this protocol could be implemented on other hardware mediums in modular systems. As such it is required to have another form of ID in the protocol header. If the protocol were only to be used on Ethernet hardware layers, then the MAC address could be used as the only identifier for modules, as this can be set in the packet construction logic in the FPGA. At current, the maximum number of modules expected to be in a network is less than 125, which would be a solid three dimensional structure of 5x5x5 modules or larger structures that are not completely filled, such as a rover configuration. As such a one byte field is used for each module ID. The packet begins with the destination ID, then the source ID. IDs are assigned during the topology discovery phase of the protocol. After this there is a data class field. This is used to allow efficient processing of packets. The relevant protocol state machine is assigned to process a packet based on this data class field. Following this there is a length field, which is equal to the length of the packet including the header. Finally the header contains a checksum that is uses to check the integrity of the packet. The data in the packet can be a minimum of eight bytes long and a maximum of 65529 bytes long. This is equal to $2^{16} - headerlength$. If these packets are larger than the maximum Ethernet frame size, then they are fragmented across multiple Ethernet frames and reassembled on the receiver side. This protocol packet structure is shown in Table 4.

IV.ii Store-and-Forward Serial Transfer

The core feature of Delay-Tolerant Networking is the use of a robust Store-and-Forward system between communications nodes on a network. Remote caching of data is used in some other networking systems, such as Network File System (NFS) local file caching and Bluetooth Low Energy (BLE), but the requirements for DTN are significantly more onerous as each node effectively needs to take ownership of all data and metadata at each routing node from the start to the destination.

Mo* implements this by caching several metadata for each received data packet at each communication node:

1. A database of packets that have recently passed through the node
2. Source and destination addresses for routing each packet received
3. Timestamps for reception and destination acknowledgement for each packet
4. A cached routing table to each destination in the current network
5. A cached table of content (services and capabilities) for each destination in the current network

Of these, the latter is the most costly in terms of storage, but is necessary to make the network robust to sudden disruptions. The routing table for modules within the network is refreshed when first connected, and on detection of topology changes or faults in the system. The routing table for each node also includes the semantics of its services and capabilities, currently limited to a simple table of static variables that need to be refreshed by sending packets through the network.

The process of resource and route discovery is computationally expensive, so periodic refreshes should be avoided as much as possible, and storing as much information as possible persistently on each node is important to efficient operation of the system. This is similar to the operation of Generic Attribute Profile (GATT) caching used in Bluetooth 5.1 and up to store remote attribute handles in a database on each mesh node. The challenge in this approach is being able to detect reliably when a refresh of the network is required. If a network link is detected to be non-functional (consistently no response on transmission of data over a timeout threshold), all nodes must then refresh their databases by re-building the routing and content table by sending a broadcast "refresh" packet throughout the network.

IV.iii Protocol Simulation

To enable development of networking protocols, reconfiguration algorithms, and task planning in the Mo* modular robotic system, a simulation environment has been created to model the relevant aspects of the system. The simulation environment contains two main components. The physical simulation is performed using the Unity engine, and a custom POSIX compliant network simulator has been created to model the network interconnections between

Table 3: Format of an IEEE802.3 compliant Ethernet frame

Bytes	1-7	1	6	6	2	46-1500	4
Field	Preamble	SFD	Dest MAC	Source MAC	Length	Data	CRC

Table 4: Mo* Protocol Format

Bytes	1	1	1	2	2	8-65529
Field	Dest ModID	Source ModID	Data Class	Length	Checksum	Data

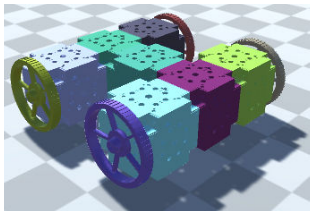
Fig. 9: A modular rover configuration in simulation

modules. Unity was chosen as the physical simulation engine as it is has a very flexible implementation. All aspects of the unity simulation environment are customisable through a common c# interface. Unity uses PhysX for its physics engine. The network simulator is time synchronised from the Unity simulator. Arbitrary module configurations can be created in the simulator. Currently a model of a kinematically static module and a wheel module exists. With this, modular rovers can be created in simulation, like that shown in Figure 9.

The high level architecture of the simulator is shown in Figure 10. The simulator is broadly split in to two parts; the backend simulator, and the unity simulator. The Unity simulator uses the Unity engine to simulate all of the physical aspects of the system including the connecting forces of the docking connectors. The backend simulator handles all of the electrical and software aspects of the modules including the network model. This has been created from scratch in C++. Attempts were made to integrate network simulators such as NS3 in to the Mo* simulation environment, but this proved out of the scope of this PhD. The two parts interface with each other using inter process communication via the sockets API in Linux.

The aim of the simulator was to create a flexible simulation platform for modular robotics in which it is relatively simple to alter components of the simulator for different scenarios. The back end simulator is split in to various components for the different parts of the system. There is a unity sim interface module that handles the passing of data between the backend simulator and the unity physics simulator through the sockets API. This interfaces with a simulation manager module that is responsible for managing data flow between and synchronising the Ethernet channel simulator and the module manager. The module manager is responsible for managing the data flow between the various parts of each of the module instances. In an attempt to make the simulator as versatile as possible, each module instance itself was split in to a PHY emulator, FPGA functions emulator and Module code section. The goal of this was to make it possible to write code for the different parts of a module between the application code and the FPGA, and test this with different PHY device implementations. In hindsight, this was too optimistic an undertaking. The system works as intended, but took a lot longer than expected to get to this stage. A simpler architecture would have allowed much more time for experiments to be run. It is hoped that the current implementation can be built upon to generate a unified simulation environment for modular robots.

Experiments have been run to test the models of the network side of the simulator. Experiments were run on a 3x3x3 module strictly orthogonal network of kinematically static modules. A broadcast packet is sent from one of the corner modules, and the time taken for that packet to reach all other modules in the system is recorded. This experiment is repeated with packet payload sizes from 1-5 bytes. The results from this experiment is shown in Figure 11. These results show the latency of the packet propagation increasing with payload size as expected. A render of the network of modules used in this simulation is shown in Figure 12. Once full integration of a module's mechanical and electronic components have been created, this experiment can be repeated in hardware to test the reality gap of the simulator, and tune the parameters of the models used.

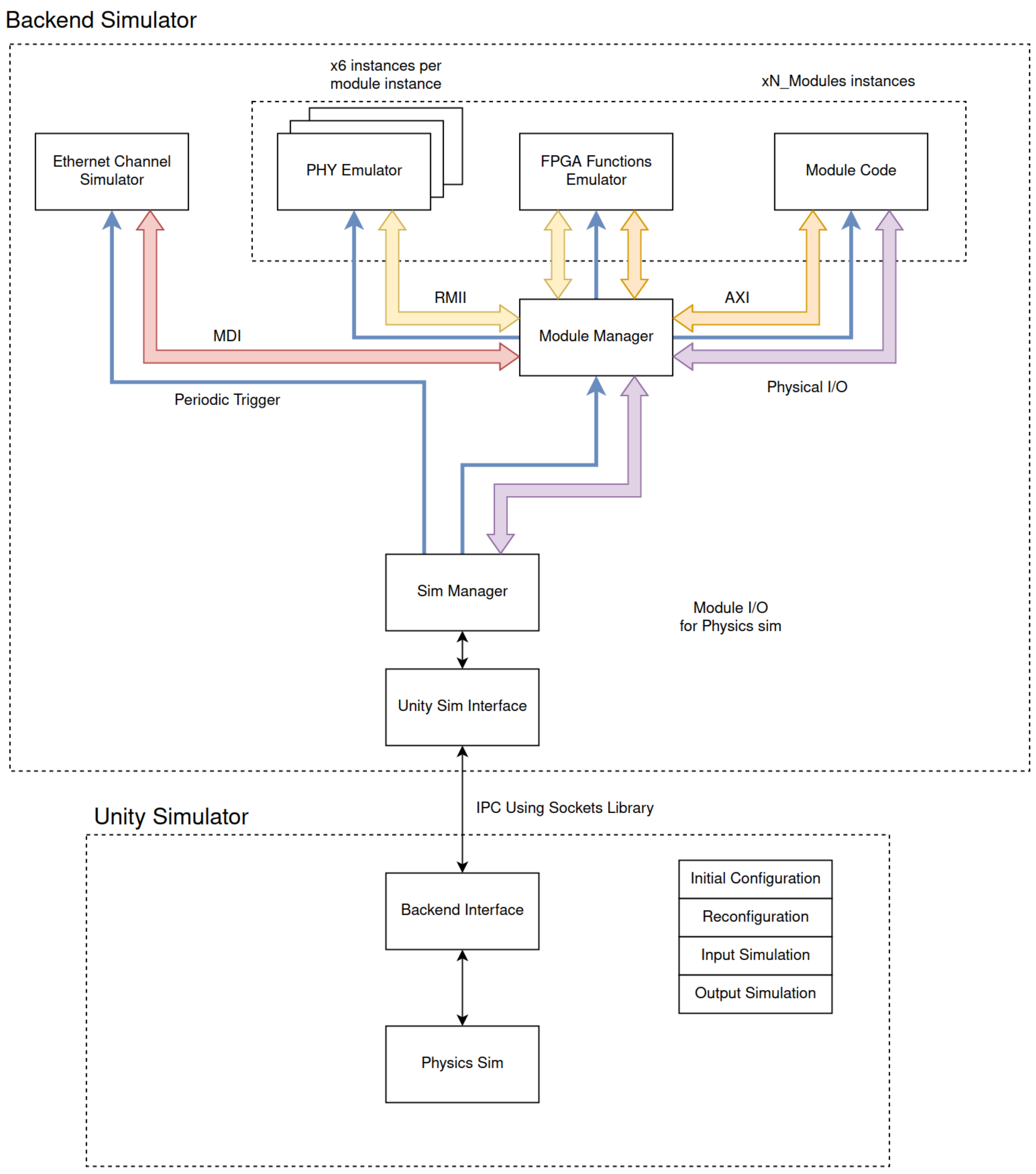

Fig. 10: Network simulation architecture used for verification of Mo* protocol

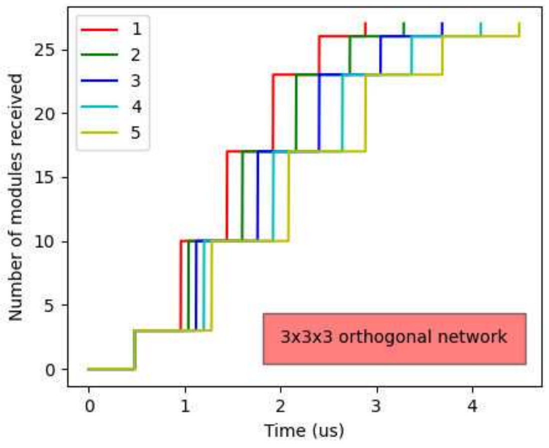

Fig. 11: Simulation results of propagation delay of a broadcast packet in a strictly orthogonal network for varying packet payload sizes (bytes).

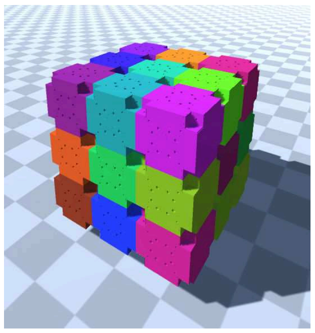

Fig. 12: Render of a 3x3x3 strictly orthogonal network of modules in simulation.

V. Wireless Communications

V.i Hardware Selection

For the requirements of a short-range, flexible, high-bandwidth mesh network with off-the-shelf components, IEEE 802.11 based ”WiFi” systems are closest to the requirements. WiFi has been used in several modular robotic systems, including Swarmbot, Imobot, SMORES and T.E.M.P.[24] The IEEE 802.11s standard published in 2011 is a wireless Local Area Network (LAN) standard for mesh networking using WiFi hardware. 802.11s defines an architecture and protocol for both multicast and unicast transmission using multi-hop routing, but uses 802.11a/b/g/n/ac/ax point-to-point standards to carry the actual traffic, and uses IEEE 802.21 based handoff between multiple network types. In addition, routing requires at minimum the Hybrid Wireless Mesh Protocol (HWMP) and optionally supports other protocols such as Optimized Link State Routing Protocol (OLSR), Better Approach to Mobile Ad-hoc Networking (B.A.T.M.A.N.), or others. This makes an 802.11s software stack complex, relatively power-hungry, and closely linked by design to TCP/IP and other Internet protocols.

Other mature wireless networking hardware systems with lower power or complexity requirements capable of mesh networking include Bluetooth, Bluetooth Low Energy (BLE), ZigBee, and NB-IoT which is a low-power wide-area (LPWA) technology used in 5G communications. High-performance proprietary mesh networks such as Wirepas and Rajant Kinetic Mesh have been developed for specific terrestrial applications, but are not considered in this work due to the closed and specialized technologies used. Bluetooth, and BLE in particular, is focused on schema-based variable writes and reads, which works fairly well for non-dynamic data systems but modular systems will require significant adaptivity in the descriptor-based framework. ZigBee is designed well for mesh networking but generally requires a coordinator or gateway node, and needs some adaptation to be suitable for self-healing peer-to-peer communications. NB-IoT as part of the 5G suite of technologies is potentially the best adapted, but hardware is not yet easily and inexpensively available and deployment is not complete as of yet in many applications.

V.ii Implementation

Implementation of wireless networking for Mo* is still in development, but currently targets software radio hardware and RISC-V ESP32-C6 microcontrollers for experiments. While implementation of general communications functions into hardware can be beneficial, moving network and protocol functions into software as in[2] has the advantages of allowing communications to be largely platform-agnostic, while additionally facilitating forward and backward compatibility of protocols, which has been stated as an important design goal. The implementation on software radio hardware is intended to be largely platform-agnostic for maximizing flexibility.

VI. Fault Tolerant Routing with ML

Dynamic and real-time optimization of routing between multiple elements in a network is a challenging, multi-objective task that is still being explored. One promising way to facilitate this is the use of Machine Learning (ML) to allow the system to learn to adapt to different configurations and operating conditions.

In a space-based optical network, optical links between satellites and OGS can be affected by atmospheric conditions such as turbulence or adverse weather conditions, which affects the requirements on optical links to have a clear line of sight. Additionally, ISLs could be affected by factors such as changing distances between satellites, capabilities of different LCTs and varying solar background conditions. These scenarios could cause performance degradation on network links and lower throughput.

The ML solution to make the data-plane resilient to outages is through implementing autonomic network functionality by deploying a software application on each switch node to monitor its links, analyse the traffic to make a prediction on the link quality using ML and then implement local re-routing to send the traffic on the best path to reach the desired end point. This software application will be termed an 'Agent'.

The main feature of the Agent is the prediction of link quality using a novel Machine Learning (ML) model termed a deep temporal regressor (DTR). The DTR takes as an input a small time series window of network statistics arriving at the given node it is deployed on and from this makes a prediction of the link quality. An overview of the model architecture is shown in Figure 13.

Following a prediction on link quality, the Agent determines if some local re-routing action is required and what the best path to the network end point is based on its knowledge of the network topology and latencies across the network that the different Agents placed around the network have predicted. The Agent applications can communicate with each other, sending notifications of path updates so knowledge of a link degradation is shared with the nodes immediately neighbouring a degraded link, or with nodes within a limited local radius of the degraded link. This allows for each Agent to calculate the shortest path from itself to the destination at any given time leading to a rapid local rerouting behaviour.

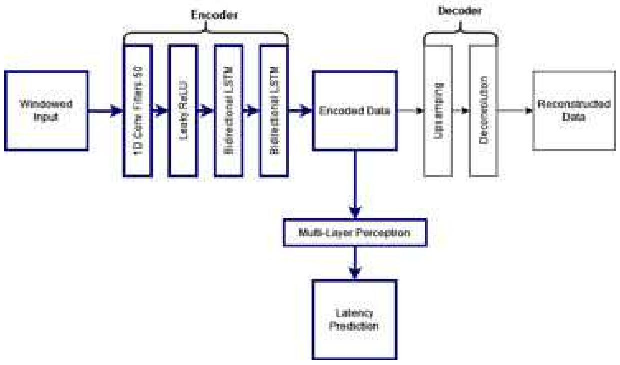

Fig. 13: Architecture of the deep temporal regression (DTR) model. Blue blocks are used at inference.

Results in Figure 14 show the improvement in throughput that is gained through using the Agents for local rerouting instead of a centralised SDN controller at higher bandwidth reductions. Testing was carried out at Craft Prospect's ML test bench by integrating Agents with an SDN network simulator based on mininet. It is also a suitable methodology for implementation on modular wired or wireless mesh networks under fault and variable connectivity conditions.

VII. Conclusions

We have presented our ongoing research towards the creation of a "Space Internet" with short-range networking capacity for modular space robots and satellites operating in proximity, aimed at applications such as the MOSAR and STARFAB modular space hardware ecosystems. The Mo* modular robotic hardware prototypes and networking protocol are part of an ongoing multi-disciplinary research and development programme aimed at maturing adaptable, robust modular hardware for both terrestrial and space applications by building on existing commercial hardware with key innovations required for modular "ecosystems" to be useful. The results to date show that it is possible to reach the challenging performance goals for modular space hardware by applying innovative technologies for interfacing, adaptivity, and ML-based optimization, although a

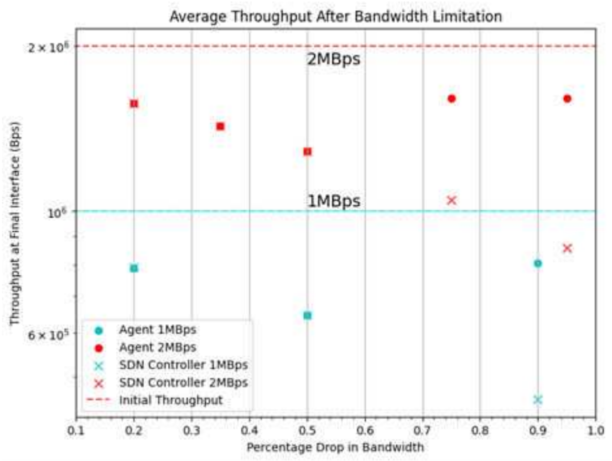

Fig. 14: the average throughput after bandwidth limitation for ML agent response and SDN controller response. Datapoints are grouped by starting throughput (1MBps and 2MBps) and the percentage reduction in bandwidth

significant amount of future work is needed to mature these technologies for safe use in space.

VIII. Future Work

VIII.i System Expansion

Currently, no sensor tiles have been created for the Mo* system. There are a wide variety of sensor tiles that could be created for varying applications. Autonomous navigation of a modular rover could be achieved with sensing capabilities such as camera, LIDAR or IMU tiles. Tiles to conduct scientific measurements gathering environmental data could also be created. One can imagine this in the context of a planetary exploration modular rover. Some such tiles that could be developed include temperature,humidity,pressure, atmospheric light, and radiation sensors.

An array of kinematic tiles and modules could be developed for the Mo* system. Wheel tiles could enable reconfigurable rovers to be created. Modules that allow even one rotational degree of freedom between two connection mechanisms could be connected together in a chain configuration to create complex mechanical structures with many degrees of freedom. Linear actuation modules could also be developed. This could enable configurations where active suspension could be applied to modular rovers, or sensor turrets that can accurately set their position relative to the target or help maintain sensor stability during locomotion.

Some work was done to implement a wheel module in the system, but this was not completed and tested within the time frame of the project. Giving the system locomotion capabilities would aid in demonstrating the capabilities of modular systems in real world settings. Multiple rover configurations could be implemented to give the system the ability to adapt to different terrains.

VIII.ii Environmental Testing

Evaluating the robustness of the Mo* communications system will require representative conditions of what modular spacecraft will encounter. This can include conditions ranging from Low Earth Orbit (LEO) to Deep Space, with significant differences in temperature, radiation exposure, and other factors. Software simulation of bit errors and carrier loss in communications channels, Single Event upsets (SEUs) and other hardware faults will be performed as a baseline. Ultimately, terrestrial network testing with environmental factors and fault injection will be done. It is also hoped that some hardware testing can be carried out under representative space conditions.

VIII.iii Routed Power Switching

The electrical components for power sharing over the Ethernet channel are similar to that of the IEEE802.3af standard. This allows a DC voltage differential to be applied between the data pairs. A system could be developed where certain modules are net producers of energy, containing large battery reserves. They could then supply power to other modules in the system that are net power consumers. Intelligent power routing schemes could be developed to efficiently share power around the system. This could enable modules that have excess energy, or have a lower mission priority, to share power to modules that are critical to mission success. This could create an intelligent power grid system with similar properties to national electrical grids, ensuring power is supplied to the correct parts of the system to ensure mission success.

VIII.iv Autonomous Software Deployment

Currently each of the tiles in a module need to have program code loaded on to them individually. This involves five programming operations for the slave tiles, with the code being the same apart from the CAN bus ID which is individually assigned for each tile. The master tile needs to be programmed separately using Xilinx Vivado tools. Work was done to implement programming of the slave tiles from the master tile, and loading new program code on to the master tile through the external ethernet interface. But this implementation has not been completed.

VIII.v ML-based Routing and Optimization

Simulation has verified that Machine Learning based optimization of routes can improve both throughput and response latency. Work is ongoing to explore how ML can be applied to the Mo* protocol and software simulation to improve routing efficiency, dynamic response, and robustness in modular wired and wireless mesh networks.

References

[1] Ian F Akyildiz, Özgür B Akan, Chao Chen, Jian Fang, and Weilian Su. Interplanetary internet: state-of-the-art and research challenges. *Computer Networks*, 43(2):75–112, 2003.

[2] Christopher Becker and Garrick Merrill. Mesh network architecture for enabling inter-spacecraft communication. 2017.

[3] Scott Burleigh, Adrian Hooke, Leigh Torgerson, Kevin Fall, Vint Cerf, Bob Durst, Keith Scott, and Howard Weiss. Delay-tolerant networking: an approach to interplanetary internet. *IEEE Communications Magazine*, 41(6):128–136, 2003.

[4] Haitao Chang, Panfeng Huang, Zhenyu Lu, Zhongjie Meng, Zhengxiong Liu, and Yizhai Zhang. Cellular space robot and its interactive model identification for spacecraft takeover control. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3069–3074. IEEE, 2016.

[5] Savio N Chau, Leon Alkalai, Ann T Tai, and John B Burt. Design of a fault-tolerant cots-based bus architecture. *IEEE Transactions on Reliability*, 48(4):351–359, 1999.

[6] Mathieu Deremetz, Raphael Boissonnade, Elisa Ballatore, Nicola Guercio, Vivien Croes, Maxence Debroise, Manon Wouters, Jan Behling, Frederic Veit, Niklas Ullrich, Mark Post, Pierre Letier, and Jeremi Gancet. Starfab: Concept of operations and preliminary design of an orbital automated hub for in space operation and service activities. In *Proceedings of the 75th International Astronautical Congress (IAC-2024), Milan, Italy, paper IAC-24.D1.1*, 2024.

[7] Raúl Dominguez, Romain Michalec, Nassir W Oumer, Fabrice Souvannavong, Mark Post, Shashank Govindaraj, Alexander Fabisch, Bilal Wehbe, Jérémi Gancet, Alessandro Bianco, et al. A common data fusion framework for space robotics: architecture and data fusion methods. ESA, 2018.

[8] Raul Dominguez, Mark Post, Alexander Fabisch, Romain Michalec, Vincent Bissonnette, and Shashank Govindaraj. Common data fusion framework: an open-source common data fusion framework for space robotics. *International Journal of Advanced Robotic Systems*, 17(2):1729881420911767, 2020.

[9] Marius Feldmann, Juan A Fraire, Felix Walter, and Scott C Burleigh. Ring road networks: Access for anyone. *IEEE Communications Magazine*, 60(4):38–44, 2022.

[10] Toshio Fukuda and Yoshio Kawauchi. Cellular robotic system (cebot) as one of the realization of self-organizing intelligent universal manipulator. In *Proceedings., IEEE International Conference on Robotics and Automation*, pages 662–667. IEEE, 1990.

[11] Ahmed Ismail Abdel Ghafar, Ángeles Vazquez Castro, and Mohamed Essam Khedr. Satellite iot services using multichord peer to peer networking. In *2019 IEEE 2nd 5G World Forum (5GWF)*, pages 566–571. IEEE, 2019.

[12] H Helvajian. Hive: A new architecture for space. In *Proceedings of the 70th International Astronautical Congress (IAC-2019), Washington DC, paper IAC-19-D4*, 2020.

[13] M Kortman, S Ruhl, Jana Weise, Joerg Kreisel, T Schervan, Hauke Schmidt, and Athanasios Dafnis. Building block based iboss approach: fully modular systems with standard interface to enhance future satellites. In *66th International Astronautical Congress (Jerusalem)*, volume 2, pages 1–11, 2015.

[14] NASA Jet Propulsion Laboratory. Nasa/jpl interplanetary overlay network (ion). `https://github.com/nasa-jpl/ION-DTN`, 2024. Accessed: 2024-09-30.

[15] Pierre Letier, Xiu T Yan, Mathieu Deremetz, Alessandro Bianco, Gerhard Grunwald, Maximo Roa, Rainer Krenn, Miguel Muñoz Arancón,

Pierre Dissaux, Juan Sánchez García Casarrubios, et al. Mosar: Modular spacecraft assembly and reconfiguration demonstrator. In *15th Symposium on Advanced Space Technologies in Robotics and Automation*, 2019.

[16] Jens Liedke, Rene Matthias, Lutz Winkler, and Heinz Wörn. The collective self-reconfigurable modular organism (cosmo). In *2013 IEEE/ASME International Conference on Advanced Intelligent Mechatronics*, pages 1–6. IEEE, 2013.

[17] Mark A Post and James Austin. Knowledge-based self-reconfiguration and self-aware demonstration for modular satellite assembly. In *10th International Workshop on Satellite Constellations & Formation Flying 2019*. York, 2019.

[18] Mark A Post, Xiu-Tian Yan, and Pierre Letier. Modularity for the future in space robotics: A review. *Acta Astronautica*, 189:530–547, 2021.

[19] Martin Rostan, Gerhard Grunwald, and Chris Thayer. Advancing space robotics with the ethercat communication standard. In *ASCEND 2022*, page 4293. 2022.

[20] Keith Scott and Scott Burleigh. Bundle protocol specification. Technical report, 2007.

[21] Yoshiki Sugawara, Shinichi Nakasuka, Kenji Higashi, Chisato Kobayashi, Kanichi Koyama, and Takanori Okada. Structure and thermal control of panel extension satellite (petsat). *Acta Astronautica*, 65(7-8):958–966, 2009.

[22] Ann T Tai, Savio N Chau, and Leon Alkalai. Cots-based fault tolerance in deep space: Qualitative and quantitative analyses of a bus network architecture. In *Proceedings 4th IEEE International Symposium on High-Assurance Systems Engineering*, pages 97–104. IEEE, 1999.

[23] Ruhai Wang, Tarik Taleb, Abbas Jamalipour, and Bo Sun. Protocols for reliable data transport in space internet. *IEEE Communications Surveys & Tutorials*, 11(2):21–32, 2009.

[24] James White. *A Novel Docking and Communications System for Heterogeneous Modular Robots.* PhD thesis, University of York, 2023.

[25] James White, Mark A Post, and Andy Tyrrell. A novel connection mechanism for dynamically reconfigurable modular robots. In *International Conference on Informatics in Control, Automation and Robotics (ICINCO) 2022*. IEEE, 2022.

[26] Jun Yang, Tao Zhang, Jingyan Song, Hanxu Sun, Guozhen Shi, and Yao Chen. Redundant design of a can bus testing and communication system for space robot arm. In *2008 10th International Conference on Control, Automation, Robotics and Vision*, pages 1894–1898. IEEE, 2008.

[27] Lei Yang, Jie Liang, Ruhai Wang, Xingya Liu, Mauro De Sanctis, Scott C Burleigh, and Kanglian Zhao. A study of licklider transmission protocol in deep-space communications in presence of link disruptions. *IEEE Transactions on Aerospace and Electronic Systems*, 59(5):6179–6191, 2023.

[28] Mark Yim, Kimon Roufas, David Duff, Ying Zhang, Craig Eldershaw, and Sam Homans. Modular reconfigurable robots in space applications. *Autonomous Robots*, 14:225–237, 2003.