# ACMENet: an underwater acoustic sensor network protocol for real-time environmental monitoring in coastal areas

G. Açar and A.E. Adams

**Abstract:** Owing to the recent advances in subsea acoustic communication technology, there is a growing interest in underwater acoustic sensor networks. The most obvious applications of such networks are environmental and military observation of coastal and ocean areas, control of autonomous underwater vehicles, fish detection, collision avoidance and long-term geological surveys. In comparison to radio frequency wireless links, underwater acoustic channels have very limited bandwidth, much longer propagation delays and higher error rates. A new network protocol, ACMENet, that provides highly efficient use of the scarce network resources – bandwidth and sensor battery lifetimes – for an underwater acoustic sensor network is proposed. The protocol was developed and tested in sea trials within the framework of the ACME project sponsored by the European Union. The objective of this research is to present the ACMENet protocol and the results of the Second and Third ACME sea trials in which ACMENet was implemented.

## 1 Introduction

Systems that are capable of monitoring the underwater environment and/or controlling underwater equipments over large areas in quasi-real-time are required by various research communities, environmentalist organisations, offshore oil companies, river and harbour authorities, and navies. The traditional solution to this problem has been to deploy sensors on the sea floor or in the water column, and transfer the measured data to the shore via cable or radio frequency (RF) links. Surface buoys equipped with radio transmitters are required above underwater sensors to provide RF links to the shore. However, it is often not possible to use surface buoys or cables on the sea floor because of the associated cost, or environmental conditions such as waves and strong currents, or human activities such as trawling, navigation and fishing. Under such conditions, underwater acoustic communication links are the only means to transfer the data measured at the sensors to the shore, or to control underwater vehicles from the shore.

The ACME project, sponsored by the European Union [1, 2], aimed at designing robust and efficient communication algorithms and network protocols capable of transferring data from a number of underwater sensor nodes to a central node via acoustic channels in shallow-water coastal areas. The project involved three sea trials. One of the specific objectives of ACME was to verify, within the third sea trial in Westerschelde, that the network can convey real-time current profile measurements. In Westerschelde shipping lane, which is the main access to Antwerp, the accurate knowledge of the current is crucial for the safe guidance of the ships to the harbour. At the moment, the current is measured using fixed poles at the sides of the shipping lane. The current in the middle of the channel is only estimated via a simulation tool.

Underwater acoustic networks pose different challenges from wired and RF networks. The fundamental problem is the limited available bandwidth, with most systems operating below 30 KHz [3]. In shallow-water areas, the attainable transmission rate is further reduced due to time-varying multipath propagation, large Doppler shifts and time spreading [4]. In addition, coastal areas have highly variable sound-speed profiles due to wind-driven, estuarine and tidal mixing processes, as well as high and variable levels of ambient noise associated with commercial shipping activity [5]. Typical acoustic modems support a bit rate of 100 to 1000 bps in shallow-water channels. Higher transmission rates, such as 2500 bps in Eggen *et al.* [6] and 16 kbps in Sharif *et al.* [7] over a range of up to 3000 m, have been reported. A summary of published results of many acoustic modems is presented in Kilfoyle and Baggeroer [2].

Subsea acoustic channels also exhibit very long propagation delays that drastically reduce maximum throughput and complicate network protocol design. The typical subsea sound propagation speed is 1500 m/s, which corresponds to 0.67 s/km propagation delay for underwater acoustic signals. This is in great contrast to the propagation delays in wired and RF networks, which are around 5 μs/km.

One of the most important constraints in the design of a sensor network protocol is the low power-consumption requirement at sensor nodes [4]. Subsea sensor nodes are powered by batteries that are costly to replace or to recharge. Thus, while the traditional network protocols aim to achieve high quality of service (QoS), protocols

*IEE Proceedings* online no. 20045060

doi:10.1049/ip-rsn:20045060

Paper first received 4th August 2004 and in revised form 9th September 2005

G. Açar was with the University of Newcastle upon Tyne and is now with the Centre for Communication System Research, School of Electronics and Physical Sciences, University of Surrey, Guildford GU2 7XH, UK

A.E. Adams is with the School of Electrical, Electronics and Computer Engineering, University of Newcastle upon Tyne, Newcastle upon Tyne NE1 7RU, UK

E-mail: g.acar@surrey.ac.uk

designed for sensor networks must also focus on energy conservation [8].

Medium access control (MAC) in most underwater acoustic networks that are proposed in the literature is based on random access protocols, which may result in packet collisions that not only reduce network throughput and cause delays, but also waste scarce energy. Random access performs well in wireless networks that support independent point-to-point connections with thin and highly unpredictable traffic characteristics. In contrast, sensor networks need to support highly correlated and mostly periodic traffic flows. MAC in ACMENet is a TDMA-based, centralised protocol that enables a master node to perform scheduled, polled and relayed data retrievals from a number of slave nodes.

## 2 Underwater acoustic communication networks

### 2.1 Multi-access problem

Multi-access medium forms the basis for underwater acoustic communication networks as well as satellite networks and radio networks. Access to the shared medium must be regulated by an MAC protocol in conjunction with a multiple access scheme. Basic multiple access schemes (FDMA, TDMA and CDMA) have been considered for subsea acoustic sensor networks in earlier research studies.

The limited bandwidth of underwater acoustic channels inhibits the use of FDMA to achieve asynchronous multi-access in different frequency bands. Instead, the use of FDMA was proposed to reduce interference between adjacent clusters in some underwater acoustic networks. Sozer *et al*. [4] and Green *et al*. [9] describe an underwater acoustic network that is divided into a number of clusters. Each cluster is assigned a dedicated frequency band, and a separate frequency band is reserved for inter-cluster communication.

Owing to its high flexibility, TDMA has been proposed for many underwater acoustic networks reported in the literature. However, TDMA requires some level of synchronisation among network nodes, which may be difficult to attain in underwater acoustic networks where propagation delays can be long and variable.

Direct Sequence CDMA (DS-CDMA) is a promising multiple access scheme for underwater acoustic networks because of its resistance against multipath propagation and frequency-selective fading, and its potential for asynchronous multi-access provision. However, because of the high complexity of the associated algorithms, real-time asynchronous channel multi-access using DS-CDMA in underwater acoustic modems has not been realised yet. Off-line performance analyses of several such receivers over experimental data can be found in Tsemenidis *et al*. [10] and Yeo *et al*. [11].

There is a wide range of MAC protocols proposed for satellite networks and radio networks. There are two extremes among the many MAC protocols that have been developed: random access (a.k.a., contention-oriented) and scheduled access (a.k.a., contention-free) protocols. In random access protocols, each network node may access the channel at a random time independently from other network nodes. Collisions will happen when signals from multiple sending nodes arrive at a network node. Many random access protocols exist with different collision avoidance and retransmission strategies. In scheduled access protocols, collisions are alleviated by dynamically reserving a dedicated time interval to each network node. Scheduled access protocols can be distributed or centralised. Many scheduled access protocols with different reservation strategies have been studied in the literature.

Although carrier-sensing in wired LANs reduces collision probability of random access by making network nodes avoid accessing the channel during ongoing transmissions, MAC in wireless networks cannot rely on carrier-sensing alone because not all network nodes can be guaranteed to be within the radio range of each other. Medium Access with Collision Avoidance (MACA) is proposed in Karn [12] to circumvent this problem. In MACA, data transmissions are preceded by an exchange of utility messages, Request-to-Send (RTS) and Clear-to-Send (CTS), between sender and receiver. The RTS–CTS message exchange also serves as a warning to the neighbouring nodes about the planned data transmission that is soon to take place. MACA for Wireless (MACAW) [13] improves MACA by introducing additional utility messages, Data Sending (DS) and Acknowledgement (ACK). Neither MACA nor MACAW is a contention-free protocol. RTS packets from several nodes may collide at the receiver node. Also, in the presence of noise and radio interference, neighbouring network nodes may fail to detect RTS/CTS packets and initiate packet transmissions that cause collisions.

Most underwater acoustic networks that are reported in the literature involve MAC protocols that are based on MACA. Random access protocols, such as MACA and MACAW, make the fundamental assumption of stochastically distributed traffic and tend to support independent point-to-point flows. However, MAC protocols for sensor networks must be able to support variable but highly correlated and predominantly periodic traffic [8]. This is especially true for underwater acoustic monitoring networks where sensor nodes periodically perform environmental measurements at a sampling rate that is already known at the master node. The throughput and delay performance of random access protocols are further reduced in underwater environments because of long propagation delays and frequent link outages that are typical features of subsea acoustic channels. Under such conditions, to avoid instability with random access, the measurement network must be operated at a very low duty cycle. Nonetheless, random access protocols can be very useful in event-tracking sensor networks that monitor the sensor field to detect rare events such as submarines.

Scheduled access protocols are well known to provide high network throughput and low delay at high network loads in networks with long propagation delays such as geostationary satellite networks [14]. Many centralised scheduled access protocols have been proposed for satellite networks [15]. Centralised scheduled access protocols are deemed inappropriate for multi-hop wireless sensor networks with no central controlling agent. Instead, several distributed scheduled access protocols have been proposed in Sohrabi *et al*. [16] and Woo and Culler [17]. However, such protocols are designed for dense wireless sensor networks with negligible propagation delays, very-low-bit error probabilities and large communication bandwidth that can be divided into separate frequency bands for control and data signalling.

### 2.2 Related work

Although subsea acoustic communication has been around since 1945 [18], research in networked underwater acoustic modems has only gained popularity in the last decade. Acoustic Local Area Network (ALAN) consists of a master node at the surface that communicates with a number of bottom nodes over vertical channels of about 1000 m distance [19]. Bottom nodes may initiate a data transfer by first sending a request to the master node associated with the length of the planned data transmission. The master node sends an ACK indicating that the bottom node may begin its data transmission. To reduce packet collision probability, a different frequency band is used for request, ACK and data transmission. Thus, packet collision can only take place among request packets from several bottom nodes. The request–ACK message exchange that precedes each packet transmission may seriously reduce the throughput over channels with long propagation delays and high error rate (i.e. shallow-water acoustic channels).

Talavage *et al.* [20] propose a shallow-water version of ALAN with an access protocol that is based on the Packet Radio Network (PRN) protocol that was described in Jubin and Tornow [21]. The proposed protocol allows each network node to transmit data to a neighbour node without any request–ACK message exchange. The receiving node sends an ACK packet to the source network node and relays the received data packet. Unlike the original PRN protocol, the protocol described in Talarage *et al.* [20] assigns a different frequency band for data transmission, ACK transmission and relayed data transmission.

SEAWEB is an initiative established by the US Navy for advancing an infrastructure linking diverse undersea assets to manned command centres [22]. A non-military application of the SEAWEB technology is the FRONT project [23] that defines a network involving sensor nodes, repeaters and gateways. Benthos ATM885 modems have been used in SEAWEB and FRONT experiments. Although they can be configured to operate in three different frequency bands, these modems are currently not capable of transmitting and receiving simultaneously in more than one frequency band. The first implementation of SEAWEB adopted a 3-cluster FDMA strategy with each cluster in the network using one of the three frequency bands. However, since the targeted reduction in probability of collision was not realised, FDMA was abandoned in subsequent SEAWEB implementations in favour of TDMA and hybrid TDMA/CDMA [24]. MAC in SEAWEB is based on the MACA protocol.

## 3 ACME project

The Acoustic Communication network for Monitoring of Environment in coastal areas (ACME) project was funded by the European Union within the framework of the Energy, Environment and Sustainable Development thematic program. The objective of the ACME project is to design and implement reliable communication algorithms and network protocols to enable long-term, real-time observation of the underwater environment in coastal areas, via subsea acoustic sensor networks. The scientific results that were obtained in other related projects – SWAN, LOTUS and ROBLINKS – that were previously sponsored by EU MAST III program have been exploited in the ACME project. ACMENet is a master–slave network protocol designed and implemented within the framework of the ACME project.

Three sea trials were conducted to evaluate the performance of communication algorithms and demonstrate the capabilities of the network protocol. The first sea trial aimed at determining the characteristics of the environment in the Westerschelde shipping lane, in the Netherlands, in order to define the limiting factors for an underwater acoustic communication link (noise spectra, frequency and time spreading), at collecting an environmental data set (sound-speed profiles, bathymetry and sediment properties) and to determine the usability and limitations of the existing modems in the area. The second sea trial was designed to collect similar data at Bay of Douarnenez in France and demonstrate several capabilities of the ACMENet protocol. The third sea trial aimed to test and evaluate the whole prototype network in a configuration defined to meet the end-users' needs which means long-term, real-time, online monitoring of underwater environment in the Westerschelde.

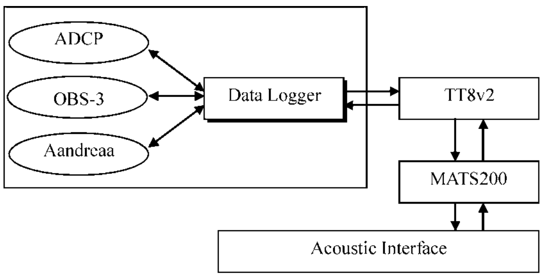


**Fig. 1** *Slave node in ACME*

### 3.1 ACME slave sensor nodes

Fig. 1 shows the elements of a slave node in the ACME project. The slave-side ACMENet protocol is designed and implemented by the University of Newcastle, UK, in a TattleTale 8v2 (TT8v2) microcontroller developed by Onset Computer Corporation, MA, USA. The software component that operates in TT8v2 not only implements the ACMENet protocol, but also manages the MATS200 acoustic modem, which is developed by ORCA Instrumentation in France. Each slave node may contain different sensor units that are integrated to TT8v2 by a data logger. Three commercially available sensor units have been employed. The acoustic Doppler current profiler (ADCP) sensor was used to measure the velocity and direction of the water flow at various depths. An Aandreaa sensor system measured the water temperature and conductivity, and the OBS-3 sensor measures turbidity and suspended solids.

Both TT8v2 and MATS200 modems have deep sleep modes that minimise energy consumption during inactive periods. MATS200 modems can be activated either by the modem commands from TT8v2 or by acoustic wakeup signals that are received via the command link on MATS200 acoustic interface. The command link is based on linear frequency modulation with integrated error correction, and can only provide an effective transmission rate of about 6 bits/s. MATS200 data link can support higher transmission rates via different modulation types without error correction. However, the modem needs an explicit

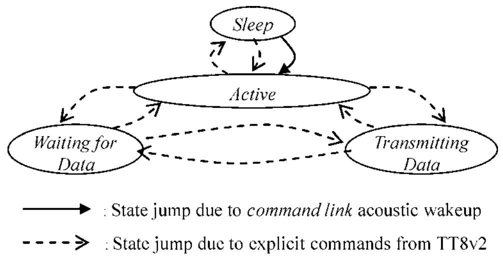

Sleep
Active
Waiting for Data
Transmitting Data
: State jump due to *command link* acoustic wakeup
: State jump due to explicit commands from TT8v2

**Fig. 2** *MATS200 states*

Table 1: **Modulation schemes in ACME**

| | Second Sea Trial | | Third Sea Trial | | Bit rate (bps) |
|---|---|---|---|---|---|
| | Tx | Rx | Tx | Rx | |
| MFSK-1 | ✓ | ✓ | ✓ | ✓ | 100 |
| MFSK-2 | ✓ | ✓ | ✓ | ✓ | 200 |
| 7-bit PN | | | | | |
| CDMA-1 BPSK | ✓ | × | × | × | 129 |
| CDMA-2 QPSK | ✓ | × | × | × | 258 |
| 15-bit PN | | | | | |
| CDMA-3 BPSK | ✓ | × | ✓ | ✓ | 267 |
| CDMA-4 QPSK | ✓ | × | ✓ | ✓ | 533 |
| 31-bit PN | | | | | |
| CDMA-5 BPSK | ✓ | × | × | × | 571 |
| CDMA-6 QPSK | ✓ | × | ✓ | ✓ | 1142 |

Table 2: **MATS200 power consumption**

| | Min | Medium | Max |
|---|---|---|---|
| Acoustic source level (dB) | 173 | 179 | 185 |
| Power consumption in transmission (W) | 5.5 | 21 | 84 |
| Power consumption in reception (W) | | 1.92 | |
| Power consumption in sleep mode (W) | | 0.024 | |
| Battery | | 24 V dc/38 Ah | |
| Transducer | | Omnidirectional | |

command from TT8v2 to go into Transmitting Data or Waiting for Data states at a specific modulation/demodulation rate (see Fig. 2).

Table 1 summarises the modulation schemes implemented on MATS200 data link during the Second and Third ACME Sea Trials. The real-time demodulation of CDMA signals was not performed during the sea trials. MATS200 modems have three transmission power levels. Table 2 provides a summary of issues related to MATS200 energy consumption.

## 4 ACMENet protocol

ACMENet is a TDMA-based master–slave network protocol designed for a small (i.e. maximum of 13 slaves in its current format) underwater acoustic sensor network. As illustrated in Fig. 3, in ACMENet, the master node acts as the information sink where data from the slave nodes are collected. Slave nodes consist of sensors, microcontrollers, batteries and acoustic modems. The lifetime of a slave node is usually limited by the lifetime of its battery. Minimising the energy consumption at a slave node is crucial for the cost efficiency of the whole network. To minimise the energy consumption at slave nodes, the slave-side ACMENet protocol is designed to be simple and reliable. Much of the computational complexity and intelligence in ACMENet is placed at the master node, which is assumed to have much more energy. As shown in Fig. 3, the master node may also serve as a slave node to a primary master in a larger underwater sensor network.

### *4.1 ACMENet scheduler*

In scheduled data retrieval, the master node computes the transmission schedule according to the propagation delays between the master and each slave node. The transmission schedule is encapsulated in a CTRL (i.e. ConTRoL) packet and broadcast to slave nodes. Reception of the CTRL packet at a slave node acts as a reference point in time. The transmission schedule specifies how long each slave must wait after the reception of the CTRL packet and before the slave node may start its data transmission.

As illustrated in Fig. 4, because of the low acoustic propagation speed, a slave that is only 4.5 km further away from the master than another slave will wait for 3 s longer before a CTRL packet arrives from the master. It is possible for the nearer slaves to complete their data transmissions before the CTRL packet arrives at the furthest slave in the network. To minimise the total time required to perform data retrieval from all slave nodes, data packets from nearer slaves are scheduled to arrive at the master earlier than data packets from further slaves. Accordingly, the transmission schedule is designed in such a way that data packets from slaves are received at the master node in the ascending order of propagation delays between the master and each slave node.

To avoid packet collisions at the master node, the computation of the transmission schedule must take into account the length of data packet expected from each slave node. The computation of the transmission schedule begins with the sorting of slave node indices in the ascending order of their propagation delays from the master node. Accordingly, in a network of $N$ slave nodes, define $\bar{\boldsymbol{T}} = [\tau_1, \tau_2, \ldots, \tau_N]$ as the vector of propagation delays in the ascending order. Similarly, define $\bar{\boldsymbol{D}} = [D_1, D_2, \ldots, D_N]$ as the vector of data transmission times in ascending order of the propagation delays. First consider the impact of the propagation delays on the return path (see Fig. 3). We assume that all slaves have already received the CTRL packet and are ready to transmit data. If there is zero propagation delay in the network, ideally the data packets would be received at the time instants shown in Fig. 5, where $t_g$ stands for the guard time between two consecutive packet receptions. The packet arrival times can be expressed by the vector $\bar{\boldsymbol{A}} = [0, (D_1 + t_g), \ldots, \sum_{j=1}^{n} (D_{j-1} + (j-1)t_g), \ldots, \sum_{j=1}^{N} (D_{j-1} + (j-1)t_g)]$ for $n = 1, \ldots, N$ and $D_0 \equiv 0$.

However, in the presence of non-zero propagation delays, in order to attain the same order of packet reception together with the guard times, the packet arrival times must be shifted by $\Delta t = \max\ (\bar{\boldsymbol{T}} - \bar{\boldsymbol{A}})$, also shown in Fig. 5. The new packet arrival times can be represented by the vector $\boldsymbol{R} = [\Delta t, (\Delta t + D_1 + t_g), \ldots, \Delta t + \sum_{j=1}^{n} (D_{j-1} + (j-1)t_g), \ldots, \Delta t + \sum_{j=1}^{N} (D_{j-1} + (j-1)t_g)]$. This is achieved by making each slave wait before data transmission commences. The vector $\bar{\boldsymbol{W}}_R = \bar{\boldsymbol{R}} - \bar{\boldsymbol{T}}$ contains the waiting time for each slave node so that the data packets can be received at the time instants expressed by the vector $\bar{\boldsymbol{R}}$.

So far, only the impact of the propagation delays on the return path has been considered. In reality, each slave node must complete the reception of the CTRL packet, which contains the transmission schedule, before data transmission. The earliest time when all slaves can be ready to transmit is dictated by the longest propagation delay in the network. Slaves that are closer to the master must wait for some time after receiving the CTRL packet until it is received at the furthest node. In reference to the sorted propagation delays vector, $\bar{\boldsymbol{T}} = [\tau_1, \tau_2, \ldots, \tau_N]$, the amount of time each slave should wait after receiving the

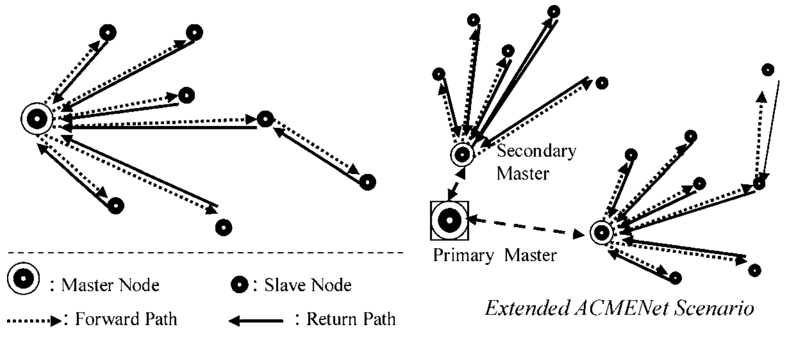


**Fig. 3** *Generic network scenario for ACMENet and extended ACMENet scenario*

CTRL packet is expressed by the vector $\bar{W}_F = [(\tau_N - \tau_1), (\tau_N - (\tau_2), \ldots, (\tau_N - \tau_{N-1}), 0]$.

$\bar{W}_F$ and $\bar{W}_R$ and are derived by decoupling the return path from the forward path. $\bar{W}_F$ is the set of waiting times for slave nodes so that all slaves can complete the reception of the CTRL packet and be ready for data transmission. $\bar{W}_R$ is the set of waiting times for slave nodes so that the data packets are received at the master node in the desired order together with the guard times. When the forward path is coupled with the return path, the waiting times expressed by $\bar{W}_F$ and $\bar{W}_R$ must be combined to obtain the overall waiting time for each slave node after receiving the CTRL packet. Simple addition of $\bar{W}_F$ and $\bar{W}_R$ may result in some redundant waiting times. For instance, consider a network of only two slave nodes where slave-2 has to wait for 1 s according to $\bar{W}_R = [0, 1]$ and slave-1 has to wait for 1 s according to $\bar{W}_F = [0, 1]$. Indeed, when the forward path and the return path are considered together, neither slave has to wait at all, because the 1 s late arrival of the CTRL packet at slave-2 can be counted for the 1 s waiting time that slave-$^2$ has to go through according to $\bar{W}_R$. This argument can be generalised for networks with more than two slave nodes, and the overall waiting time for each slave after the reception of the CTRL packet can be expressed by the vector $\bar{W} = \bar{W}_F + \bar{W}_R - [1\ 1 \cdots 1] \cdot \min(\bar{W}_F + \bar{W}_R)$. In the final stage of the computation of the transmission schedule, the vector $\bar{W}$ is re-ordered according to the slave node indices.

### 4.2 Packet formats in ACMENet

All packets in ACMENet begin with a generic header that includes TO, FROM, RELAY TO and TYPE fields. The RELAY TO field indicates the address of the node to which the packet is to be relayed. Three packet types are defined in ACMENet: CTRL, DATA and SYNC. The TYPE field designates the type of the packet and presents type-dependent information such as the number of instructions in CTRL packets or the slave status in DATA packets.

CTRL packets carry instructions from the master to slaves. Each instruction is composed of a 1-byte instruction type field and a variable length of instruction data. Each instruction is also accompanied by a 1-byte sensor specific command (SSC) that is for the sensor(s) that is (are) attached to the slave node. CTRL packets can be broadcast or unicast. A unicast CTRL packet contains only one instruction and SSC pair. A broadcast CTRL packet may contain several instructions and SSCs. SYNC packets are short unicast packets that are exchanged between the master node and a slave node during the synchronisation operation that is required to estimate the master-to-slave propagation delay. Although it is mostly designed for the network initialisation phase, it can be invoked during the normal operation of the network. The master node is also capable of updating propagation delays after each data retrieval. Data packets are unicast packets that carry the sensor measurements from the slave nodes to the master node. Fig. 6 shows the ACMENet packet formats.

Note that MATS200 command link must be used to send CTRL and SYNC packets to enable them to acoustically wake up MATS200 modems (see Fig. 2). The low transmission speed of the command link (i.e. 6 bps) seriously restricts the length of CTRL/SYNC packets. Section 5.2 presents our solution to avoid prolonged CTRL transmissions by limiting the total length of instructions.

### 4.3 Instructions in ACMENet

Instructions in ACMENet convey information from the master node to slave nodes regarding transmission schedule, desired modulation type, desired transmission power levels and sensor-specific commands (SSCs). ACMENet instructions can be unicast, broadcast and multicast. A unicast instruction is issued to an individual slave node and carried in a unicast CTRL packet. A broadcast instruction is destined to all slave nodes in the network and carried in a broadcast CTRL packet. Multicast instructions are needed to allow the master node to retrieve data from a subset of slave nodes in the network. A multicast instruction contains the list of addresses for its recipient slave nodes.

All ACMENet instructions request data from one or more slaves. Although some instructions such as BR_DATA_SCH (broadcast data transmission schedule) contain a

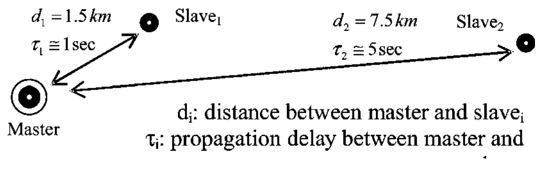


**Fig. 4** *Impact of distance on acoustic propagation*

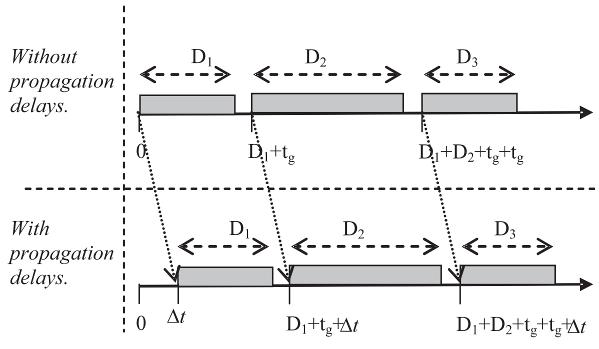


**Fig. 5** *Arrival times with and without propagation delays*

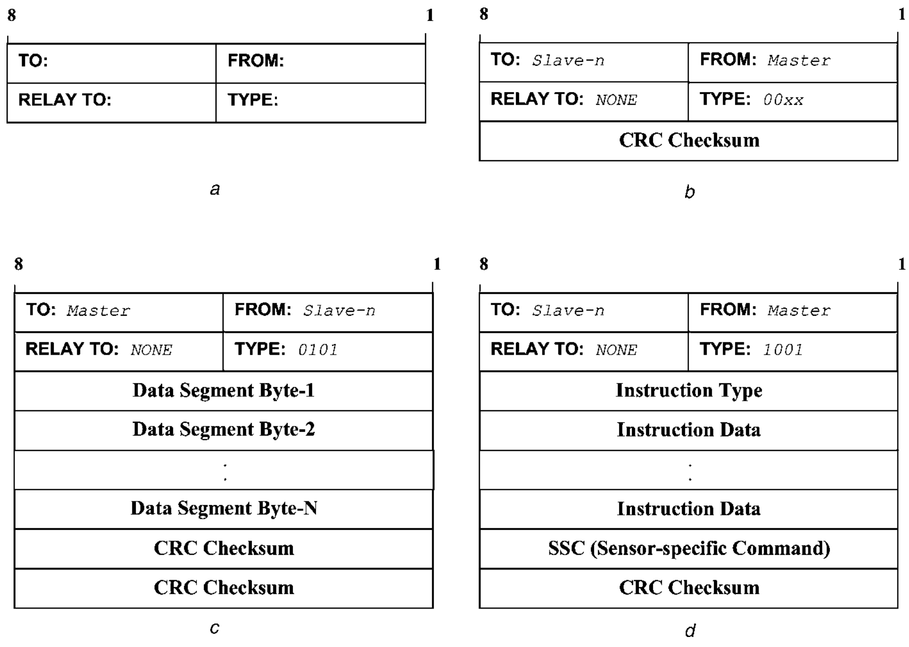


**Fig. 6** *Packet formats*
*a* Generic header
*b* SYNC packet
*c* DATA packet
*d* CTRL packet with one instruction

transmission schedule, other instructions such as BR_DATA_REQ (broadcast data request) instructs the slaves to use the last transmission schedule stored. Instructions such as BR_DATA_SCH_jk and BR_DATA_SCH_j are variations of BR_DATA_SCH notify the desired modulation type (indicated by *j*) and the desired transmission power level (indicated by *k*) at the slave node. Table 3 summarises the ACMENet instructions where MULTI- and UNI-prefixes designate multicast and unicast instructions, respectively. Fig. 7 shows the formats for BR_DATA_SCH, BR_DATA_REQ, MULTI_DATA_SCH, MULTI_DATA_REQ and UNI_DATA_REQ_jk instructions. In each instruction, the first octet defines the instruction type which is used to interpret the rest of the instruction. The instruction type is also used to convey modulation type and power level information. Accordingly, Table 3 also shows the range of values (in hexadecimal) of the first octet for each instruction type. Note that these ranges are derived for five modulation types and three power levels.

### *4.4 Relayed transmissions in ACMENet*

Packet relaying is supported for all packet types (i.e. CTRL, SYNC and DATA) in ACMENet. Relayed packet transmissions enable the master node to perform synchronisation and data retrieval operations with remote nodes via slave nodes acting as relay nodes.

Synchronisation with a remote slave via a relay slave node is illustrated in Fig. 8 that represents the SYNC messages as data structures with TO, FROM and RELAY TO fields. When a slave receives a SYNC packet with RELAY TO field equal to 15 (i.e. NONE) from the master node, it sends a SYNC packet to the master node. If the SYNC packet arrives from another slave, then the slave node creates a SYNC packet with RELAY TO field set to the master's address and sends it to the sender of the received SYNC packet. When a slave node receives a SYNC packet with the RELAY TO field indicating a valid slave address, the slave node modifies the packet header such that it

**Table 3: ACMENet Instructions**

| Instruction type | Transmission schedule | Modulation type | Power level | First octet range |
|---|---|---|---|---|
| BR_DATA_SCH | Encapsulated | None | None | 0 × 00–0 × 00 |
| BR_DATA_SCH_j | Encapsulated | Conveyed by *j* | None | 0 × 01–0 × 05 |
| BR_DATA_SCH_jk | Encapsulated | Conveyed by *j* | Conveyed by *k* | 0 × 06–0 × 14 |
| MULTI_DATA_SCH | Encapsulated | None | None | 0 × 15–0 × 15 |
| MULTI_DATA_SCH_j | Encapsulated | Conveyed by *j* | None | 0 × 16–0 × 1A |
| MULTI_DATA_SCH_jk | Encapsulated | Conveyed by *j* | Conveyed by *k* | 0 × 1B–0 × 29 |
| BR_DATA_REQ | None | None | None | 0 × 2A–0 × 2A |
| BR_DATA_REQ_k | None | None | Conveyed by *k* | 0 × 2B–0 × 2D |
| MULTI_DATA_REQ | None | None | None | 0 × 2E–0 × 2E |
| MULTI_DATA_REQ_k | None | None | Conveyed by *k* | 0 × 2F–0 × 31 |
| UNI_DATA_REQ_jk | None | Conveyed by *j* | Conveyed by *k* | 0 × 32–0 × 40 |

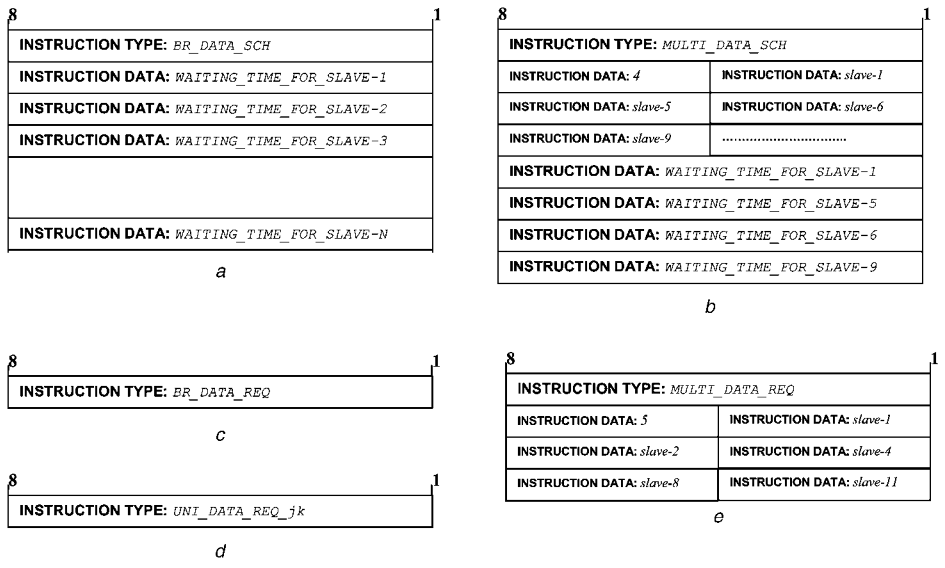

8
1
INSTRUCTION TYPE: BR_DATA_SCH
INSTRUCTION DATA: WAITING_TIME_FOR_SLAVE-1
INSTRUCTION DATA: WAITING_TIME_FOR_SLAVE-2
INSTRUCTION DATA: WAITING_TIME_FOR_SLAVE-3
INSTRUCTION DATA: WAITING_TIME_FOR_SLAVE-N
a
INSTRUCTION TYPE: MULTI_DATA_SCH
INSTRUCTION DATA: 4
INSTRUCTION DATA: slave-1
INSTRUCTION DATA: slave-5
INSTRUCTION DATA: slave-6
INSTRUCTION DATA: slave-9
INSTRUCTION DATA: WAITING_TIME_FOR_SLAVE-1
INSTRUCTION DATA: WAITING_TIME_FOR_SLAVE-5
INSTRUCTION DATA: WAITING_TIME_FOR_SLAVE-6
INSTRUCTION DATA: WAITING_TIME_FOR_SLAVE-9
b
INSTRUCTION TYPE: BR_DATA_REQ
c
INSTRUCTION TYPE: UNI_DATA_REQ_jk
d
INSTRUCTION TYPE: MULTI_DATA_REQ
INSTRUCTION DATA: 5
INSTRUCTION DATA: slave-1
INSTRUCTION DATA: slave-2
INSTRUCTION DATA: slave-4
INSTRUCTION DATA: slave-8
INSTRUCTION DATA: slave-11
e

**Fig. 7** *Instruction formats*
*a* BR_DATA_SCH
*b* MULTI_DATA_SCH
*c* BR_DATA_REQ
*d* UNI_DATA_REQ_jk
*e* MULTI_DATA_REQ

assigns the original RELAY TO value to the TO field, assigns its own address to the FROM field and assigns 15 (i.e. NONE) to the RELAY TO field. After the packet header is modified, the new packet is relayed to the address that was expressed in the RELAY TO field of the received SYNC packet. Fig. 9 presents a flow chart for the slave-side processing upon reception of a SYNC packet.

Relayed data retrieval can be used to recover data from remote sensor nodes. In a relayed data retrieval operation, the master node sends to an intermediate slave node a CTRL packet with the RELAY TO field equal to the address of a remote slave node. The first slave that receives the CTRL packet relays it to the second slave node. In response, the second slave transmits its sensor data to the first slave. Finally, the first slave relays the received data packet to the master. The described packet exchange is very similar to that in a relayed synchronisation operation. Fig. 10 shows the slave-side processing upon reception of a CTRL/DATA packet. Note that a data packet can be transmitted to a slave node only to be relayed to the master node. Thus, the slave node ignores all data packets that are received with FROM fields equal to the master node address, or with RELAY TO fields equal to NONE. Only data packets with RELAY TO fields equal to the master address, and FROM fields equal to another slave's address are considered valid at the slave node.

Note that the relay slave node and the remote slave node must have the same data modulation types during relayed data retrieval. Otherwise, the relay node will fail to receive the data packet that is being transmitted from the remote node.

In the current version of the ACMENet protocol, which has been implemented in the Third ACME Sea Trial, each slave node in the network maintains two sets of power level and modulation type parameters. One set of power level and modulation type parameters are used in response to direct data requests (i.e. non-relay) from the master node. The second set of power level and modulation type parameters are used when the slave node is part of a relayed data retrieval, either as a relay node or as the remote slave. The joint management of the power level and modulation type at both relay and remote nodes is achieved by allowing the relay node to peek at the instruction in the CTRL packet that it is to relay to the remote node. In other words, the relay node reads the instruction in the CTRL packet before the CTRL packet is relayed to the remote node. Note that the relay node only reads the instruction to update its relay mode power level and relay mode modulation type parameters.

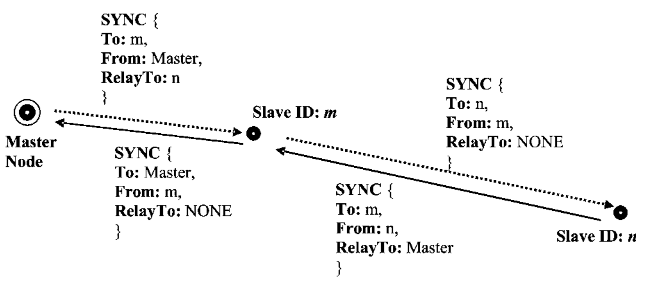


**Fig. 8** *Relayed synchronisation*

## 5 ACME controller

ACME controller is the unit that is responsible for a number of network management functions and the implementation of the master-side ACMENet protocol, computation of transmission schedules and building CTRL packets.

In the Third Sea Trial, the sensor network was designed to operate autonomously with minimal human intervention. The sensor network was integrated via a Sensor Controller unit to Rijkswaterstaat's (RWS) networked country-wide sensory data gathering system. The Sensor Controller unit was only responsible for initiating data requests. The ACME controller was responsible for the tasks listed below during the Third Sea Trial:

(i) Providing a MMI (Man Machine Interface) for the human network operator.
(ii) Managing the master MATS200 acoustic modem.

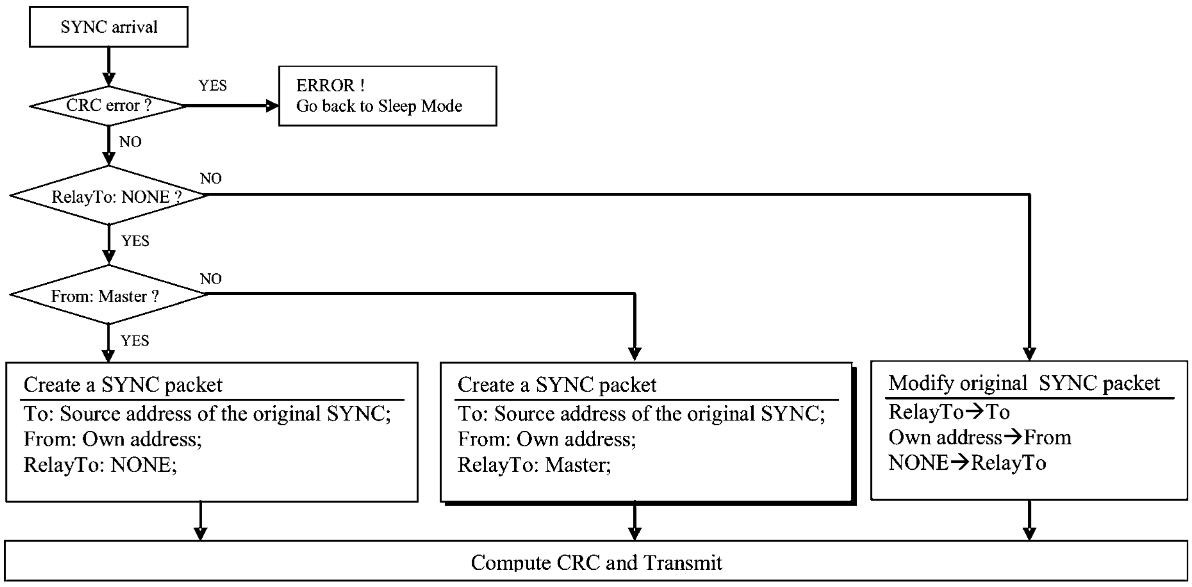


**Fig. 9** *Slave-side processing upon SYNC reception*

(iii) Measurement, storage, and display of link error statistics via the MMI.
(iv) Automatic management of slave transmission power levels and modulation type.
(v) Instruction Generation.
(vi) Interfacing to the Sensor Controller.

Fig. 11 shows the architecture of the ACME controller. The Task Manager receives, from the Sensor Controller, the list of slave nodes from which data are to be requested. At the same time, the power and rate controller (PRC) updates the modulation type and slave power levels according to an observation of the recent link error statistics. On the basis of the recommended power levels and modulation type by the PRC, the most recent propagation delay measurements and the list of target slave nodes, the Task Manager decides whether or not a new transmission schedule must be computed. The decision process involves simulating data arrivals from target slave nodes to ensure that collision will not occur. The Task Manager then invokes the Instruction Generator to find the shortest set of instructions that correspond to the desired slave power levels and the modulation type, the transmission schedule and the list of target slaves. The output of the Instruction Generator is encapsulated in an appropriate CTRL packet and transmitted to slave nodes. The Task Manager is also responsible for transferring the received data packets from the master modem to the Sensor Controller and for recording packet loss statistics and propagation delay measurements. Moreover, the Task Manager must keep a reliable record of current power levels, the modulation types and the transmission schedule at each slave node.

Minimal human intervention is required in the ACME controller to configure the PRC and to set up relay paths in the network. In the face of repeated packet losses from a specific slave node, the Task Manager may automatically resort to relayed data retrievals from this slave node along the relay paths set by the human operator.

### 5.1 Power and rate controller

As shown in Fig. 12, the PRC consists of two units: Event Generator and Rule-based Controller. According to a recent set of packet loss statistics and the user-specified

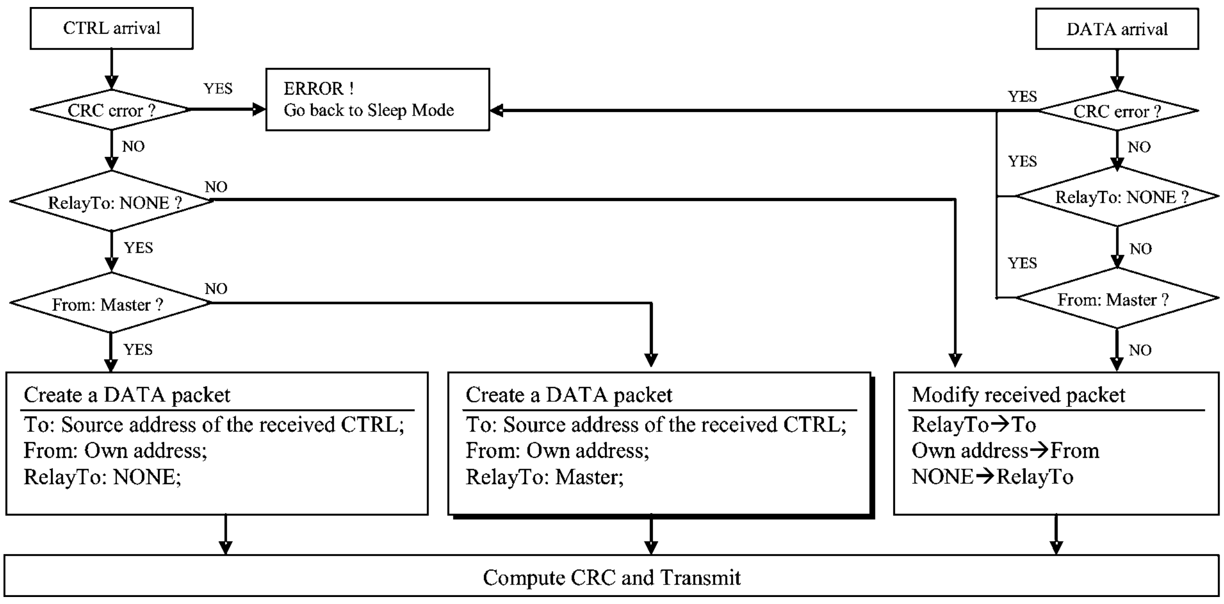


**Fig. 10** *Slave-side processing upon CTRL or DATA packet reception*

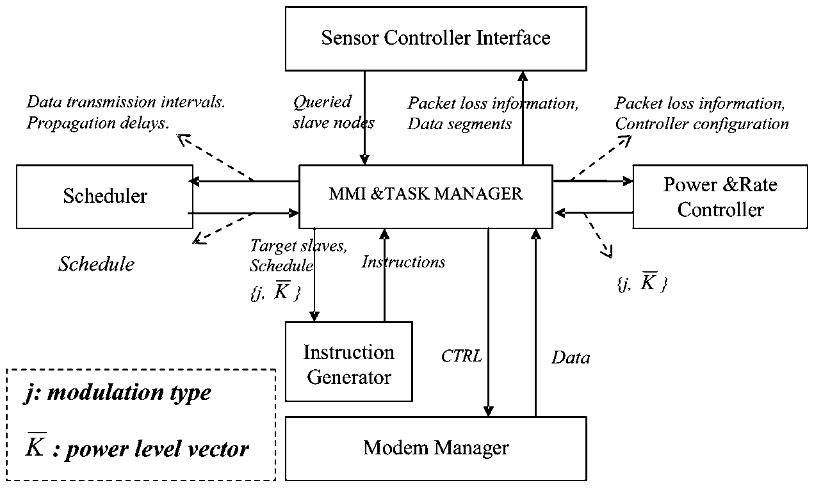

Sensor Controller Interface
Data transmission intervals. Propagation delays.
Queried slave nodes
Packet loss information, Data segments
Packet loss information, Controller configuration
Scheduler
MMI &TASK MANAGER
Power &Rate Controller
Schedule
Target slaves, Schedule {j, $\overline{K}$}
Instructions
{j, $\overline{K}$}
Instruction Generator
CTRL
Data
Modem Manager
j: modulation type
$\overline{K}$ : power level vector

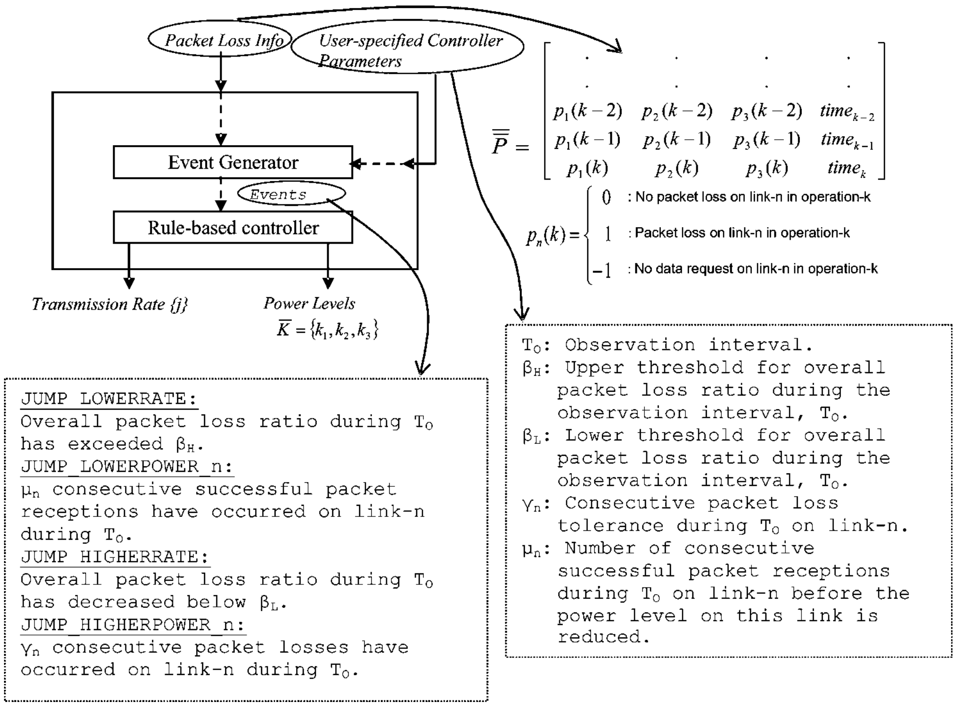
**Fig. 11** *ACME Controller*

controller parameters, the Event Generator triggers a combination of the events listed below:

- JUMP_LOWERRATE: to change to a lower rate modulation type.
- JUMP_HIGHERRATE: to change to a higher rate modulation type.
- JUMP_LOWERPOWER_n: to decrease transmission power level at slave-*n*.
- JUMP_HIGHERPOWER_n: to increase the transmission power level at slave-*n*.

The operating principle of the Event Generator is to manage the network modulation type according to the global packet loss statistics while controlling the power levels of individual slave nodes according to the packet losses on the associated link. The rationale behind this design choice is the fact that the modulation type is a global parameter, whereas the slave power level only has an impact on the performance of the associated link between the slave and the master. Thus, persistent packet losses on a link are prevented from causing changes in the modulation type.

As expressed in Fig. 12, Packet Loss Info is a matrix where each row indicates per-link packet loss information for the *k*th operation, together with the time when the operation is performed. The user-specified controller parameters are also shown in Fig. 12. The Event Generator only considers the packet loss information that is associated with the operations that take place in a recent time interval, designated by $T_o$, the Observation Interval. If the overall packet loss ratio within $T_o$ exceeds $\beta_H$, the Global Packet Loss Tolerance, the Event Generator triggers a JUMP_LOWERRATE event. If the overall packet loss ratio within $T_o$ decreases below $\beta_L$, the Target Global Packet Loss Ratio, a JUMP_HIGHERRATE event is triggered. Although JUMP_LOWERRATE indicates that a lower rate, but more reliable, modulation type is required to reduce the global packet loss ratio in the network, JUMP_HIGHERRATE indicates that the channel conditions are good enough for a faster (and possibly less reliable) modulation type. In addition to the overall packet loss



$$\overline{P} = \begin{bmatrix} . & . & . & . \\ . & . & . & . \\ p_1(k-2) & p_2(k-2) & p_3(k-2) & time_{k-2} \\ p_1(k-1) & p_2(k-1) & p_3(k-1) & time_{k-1} \\ p_1(k) & p_2(k) & p_3(k) & time_k \end{bmatrix}$$

$$p_n(k) = \begin{cases} 0 & \text{: No packet loss on link-n in operation-k} \\ 1 & \text{: Packet loss on link-n in operation-k} \\ -1 & \text{: No data request on link-n in operation-k} \end{cases}$$

```
JUMP_LOWERRATE:
Overall packet loss ratio during To
has exceeded βH.
JUMP_LOWERPOWER_n:
μn consecutive successful packet
receptions have occurred on link-n
during To.
JUMP_HIGHERRATE:
Overall packet loss ratio during To
has decreased below βL.
JUMP_HIGHERPOWER_n:
γn consecutive packet losses have
occurred on link-n during To.
```

```
To: Observation interval.
βH: Upper threshold for overall
    packet loss ratio during the
    observation interval, To.
βL: Lower threshold for overall
    packet loss ratio during the
    observation interval, To.
γn: Consecutive packet loss
    tolerance during To on link-n.
μn: Number of consecutive
    successful packet receptions
    during To on link-n before the
    power level on this link is
    reduced.
```

**Fig. 12** *Power and rate controller*

ratio within $T_o$, the Event Generator also considers the packet losses on individual links within $T_o$. If the number of consecutive packet losses on link-$n$ exceeds $\gamma_n$, which is the Consecutive Packet Loss Tolerance for link-$n$, the Event Generator triggers a JUMP_HIGHERPOWER_n event. If the number of consecutive successful packet transmissions on link-$n$ exceeds $\mu_n$, which is the Target Consecutive Successful Transmissions for link-$n$, a JUMP_LOWERPOWER_*n* event is triggered. Although JUMP_HIGHERPOWER_*n* indicates the requirement for a higher transmission power for slave-$n$, JUMP_LOWERPOWER_*n* indicates the possibility of energy conservation by using a lower power level at slave-$n$.

It is interesting to note that the events triggered by the Event Generator may have conflicting implications. For instance, in case JUMP_HIGHERRATE and JUMP_HIGHERPOWER_*n* are triggered together, the former will suggest transition to a lower energy state along modulation type and the latter will suggest a jump to a higher energy state along power level. Similarly, the Event Generator may trigger events that accentuate the outcomes of each other. For example, if JUMP_LOWERRATE and JUMP_HIGHERPOWER_n are triggered together, the result may be a jump to modulation type dimension and any power level dimension should not happen at the same time because such state-jumps may have attenuating or accentuating effects on each other. In other words, the PRC is designed to avoid making changes in modulation type and slave power levels in one CTRL packet. An important outcome of this design choice is the prevention of bloated CTRL packets that are full of multicast data requests together with a hefty transmission schedule. In reference to Fig. 12, it is the Rule-based Controller's task to convert these events into updates in modulation type and slave power levels. The Rule-based Controller in Fig. 13 is a forward-chaining system with a conflict resolution strategy that is based on the order of the fired rules in the agenda. The controller checks all rules in order until either a rule fires or the end of the rule list is reached. When a rule fires, the controller goes back to the beginning of the list. If the end of the list is reached without a fired rule, the controller terminates.

As mentioned earlier, the order of the rules determines the conflict resolution strategy of the system. The first four rules resolve cases where input events imply changes both in the transmission rate and power level. Rule-8 converts JUMP_HIGHERRATE to JUMP_LOWERPOWER_n when the transmission rate cannot be increased further. Rule-12 makes a similar conversion from JUMP_HIGHERPOWER_n to JUMP_LOWERRATE when the power level at slave-$n$ is already at maximum.

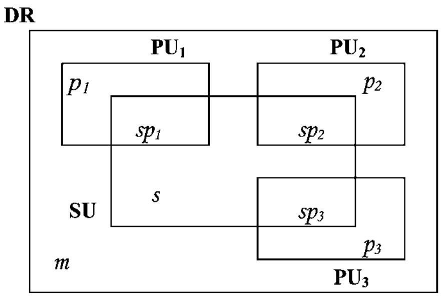
## 5.2 *Instruction Generator*

The Instruction Generator finds the shortest set of instructions that corresponds to a generic data request. The non-trivial nature of this problem is best explained by an example. Consider a sensor network with 12 slaves where the master intends to retrieve data from slaves {1, 2, 4, 5, 6, 8, 10, 12}. The transmission schedule in slaves {1, 2, 5, 10, 12} ise to be updated, whereas slaves {4, 6, 8} are allowed to use their current waiting times. Slaves {4, 5, 8} should use power level-2, slaves {1, 12} should use power level-3 and the rest of the slaves should transmit in their current power levels. There are many combinations of ACMENet instructions that correspond to this complex example data request. The Venn diagram in Fig. 14 can be used to describe a generic data request. In Fig. 14, the sets DR, SU, PU$_1$, PU$_2$ and PU$_3$ are defined together with their cardinal values. Corresponding to the example data request at hand, DR = {1, 2, 4, 5, 6, 8, 10, 12}, SU = {1, 2, 5, 10, 12}, PU$_1$ = {}, PU$_2$ = {4, 5, 8} and PU$_3$ = {1, 12}. An obvious set of instructions that correspond to the example data request are a MULTI_DATA_SCH for $\mathrm{SU}\backslash(\bigcup_{k=1}^{3}\mathrm{PU}_k)$, a MULTI_DATA_SCH_jk for each $\mathrm{SU} \cap \mathrm{PU}_k$, a MULTI_DATA_REQ_k for each PU$_k$ and a MULTI_DATA_REQ for $\mathrm{DR}\backslash(\mathrm{SU} \cup (\bigcup_{k=1}^{3} \mathrm{PU}_k))$. Obviously, an instruction will not be generated for an empty set. The total length of these instructions is presented in octets in (1), where $s = 5$, $\mathrm{sp}_1 = 0$, $\mathrm{sp}_2 = 1$, $\mathrm{sp}_3 = 2$, $p_1 = 0$, $p_2 = 3$, $p_3 = 2$ and $m = 1$ (see Fig. 14 for definitions of $s$, $\mathrm{sp}_k$, $p_k$ and $m$)

$$L = \mathrm{MDS}(s - \mathrm{sp}_1 - \mathrm{sp}_2 - \mathrm{sp}_3) + \sum_{k=1}^{3} \mathrm{MDS}(sp_k) + \sum_{k=1}^{3} \mathrm{MDQ}(p_k) + \mathrm{MDQ}(m) \qquad (1)$$

```
1- IF JUMP_HIGHERRATE AND JUMP_HIGHERPOWER_n => DELETE JUMP_HIGHERRATE
2- IF JUMP_HIGHERRATE AND JUMP_LOWERPOWER_n => KEEP THE JUMP THAT SAVES MORE ENERGY
3- IF JUMP_LOWERRATE AND JUMP_HIGHERPOWER_n => DELETE JUMP_HIGHERPOWER_n
4- IF JUMP_LOWERRATE AND JUMP_LOWERPOWER_n => DELETE JUMP_LOWERPOWER_n
5- IF JUMP_LOWERRATE AND RATE==MIN => DELETE JUMP_LOWERRATE
6- IF JUMP_LOWERRATE AND RATE>MIN => DELETE JUMP_LOWERRATE AND REDUCE RATE
7- IF JUMP_HIGHERRATE AND RATE<MAX => DELETE JUMP_HIGHERRATE AND INCREASE RATE
8- IF JUMP_HIGHERRATE AND RATE==MAX
      => CONVERT JUMP_HIGHERRATE TO JUMP_LOWERPOWER_n SUCH THAT MAX ENERGY IS SAVED
9- IF JUMP_LOWERPOWER_n AND POWER_n==MIN => DELETE JUMP_LOWERPOWER_n
10- IF JUMP_LOWERPOWER_n AND POWER_n>MIN
      => DELETE JUMP_LOWERPOWER_n AND REDUCE POWER_n
11- IF JUMP_HIGHERPOWER_n AND POWER_n<MAX
      => DELETE JUMP_HIGHERPOWER_n AND INCREASE POWER_n
12- IF JUMP_HIGHERPOWER_n AND POWER_n==MAX
      => CONVERT JUMP_HIGHERPOWER_n TO JUMP_LOWERRATE
```

**Fig. 13** *Power and rate controller rule base*



**DR:** {slaves from which **D**ata is **R**equested}
**SU:** {slaves with **S**chedule **U**pdate}
**PU₁:** {slaves with **P**ower level **U**pdate to **1**}
**PU₂:** {slaves with **P**ower level **U**pdate to **2**}
**PU₃:** {slaves with **P**ower level **U**pdate to **3**}

$sp_1$= cardinal$(SU \cap PU_1)$, $p_1$= cardinal$(PU_1)$
$sp_2$= cardinal$(SU \cap PU_2)$, $p_2$= cardinal$(PU_2)$
$sp_3$= cardinal$(SU \cap PU_3)$, $p_3$= cardinal$(PU_3)$

$s$= cardinal$(SU)$, $m$= cardinal$\left(DR \setminus \left(SU \cup \left(\bigcup_{k=1}^{3} PU_k\right)\right)\right)$

**Fig. 14** *Venn diagram describing a generic data request*

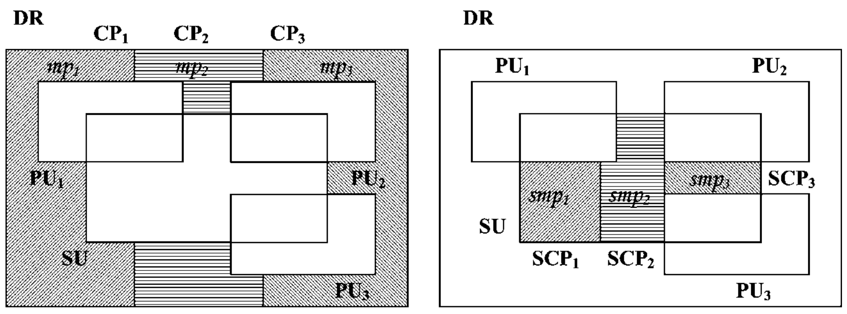
In (1), the functions MDS($n$) and MDQ($n$) designate the lengths of MULTI_DATA_SCH/MULTI_DATA_SCH_jk and MULTI_DATA_REQ/MULTI_DATA_REQ_k instructions plus 1-byte of SSC length for $n$ slave nodes, respectively. $\mathrm{MDS}(n) = u(n) \cdot (\lceil (n+1)/2 \rceil + n + 2)$ and $\mathrm{MDQ}(n) = \mathrm{u}(n) \cdot (\lceil\ (n+1)/2 \rceil + 2)$, where $u(n)$=1 for $n > 0$ and zero otherwise. Thus, the total length of the instructions for the example data request is 27 octets. An alterative set of instructions could be obtained by merging $\mathrm{DR} \setminus (\mathrm{SU} \cup (\bigcup_{k=1}^{3} \mathrm{PU}_k))$ into $\mathrm{SU} \setminus (\bigcup_{k=1}^{3} \mathrm{PU}_k)$. In this solution, the Instruction Generator issues a MULTI_DATA_SCH instruction for all slaves in $\mathrm{DR} \setminus (\bigcup_{k=1}^{3} \mathrm{PU}_k)$, where the associated transmission schedule for any slave in $\mathrm{DR} \setminus (\mathrm{SU} \cup (\bigcup_{k=1}^{3} \mathrm{PU}_k))$ is the slave's current waiting time. A MULTI_DATA_REQ_k for each $\mathrm{PU}_k$ will also be generated. The total instruction length in this case is expressed in (2), which is 25 octets

$$L = \mathrm{MDS}\left(s + m - \sum_{k=1}^{3} \mathrm{sp}_k\right) + \sum_{k=1}^{3} \mathrm{MDS}(\mathrm{sp}_k) + \sum_{k=1}^{3} \mathrm{MDQ}(p_k) \tag{2}$$

An even shorter set of instructions can be attained for the example data request by generating one MULTI_DATA_SCH instruction for $\mathrm{DR} \setminus (\bigcup_{k=1}^{3} (\mathrm{PU}_k \setminus \mathrm{SU}))$ instead of an individual MULTI_DATA_SCH_jk for each $\mathrm{SU} \cap \mathrm{PU}_k$. Note that a MULTI_DATA_REQ_k for each $\mathrm{PU}_k$ is still used to convey the power level update for slaves in $\mathrm{SU} \cap \mathrm{PU}_k$. The total instruction length in this case is expressed in (3), which is 20 octets

$$L = \mathrm{MDS}(s + m) + \sum_{k=1}^{3} \mathrm{MDQ}(p_k) \tag{3}$$

For different values of $s$, $m$, $\mathrm{sp}_k$ and $p_k$, a different instruction set may end up being the shortest one among the ones shown in (1)–(3). It is possible to generate alternative solutions by considering valid instructions for other unions and intersections of the sets shown in Fig. 19. Indeed, the number of alternative solutions can be further increased by exploiting the information regarding current power levels of slaves in DR and SU. The relevant sets of slaves $\mathrm{CP}_1$, $\mathrm{CP}_2$, $\mathrm{CP}_3$ and $\mathrm{SCP}_1$, $\mathrm{SCP}_2$, $\mathrm{SCP}_3$ together with their cardinal values are defined in Fig. 15. The idea is that a slave in $\mathrm{CP}_k$ can be included in $\mathrm{P}_k$ and covered by the corresponding MULTI_DATA_REQ_k instruction. Similarly, a slave in $\mathrm{SCP}_k$ can be included in $\mathrm{SU} \cap \mathrm{PU}_k$ and covered by the corresponding MULTI_DATA_SCH_jk instruction. For example, a valid set of instructions that correspond to the Venn diagrams in Figs. 14 and 15 are a MULTI_DATA_REQ_k for each $(\mathrm{CP}_k \cup \mathrm{PU}_k)$ if $\mathrm{PU}_k \neq \{\}$, a MULTI_DATA_REQ for $(\bigcup_k^3 = 1\,\mathrm{CP}_k)$, and a MULTI_DATA_SCH_j for SU. The total length of the instructions listed above is presented in (4), where $u(k) = 1$ for $k > 0$ and zero otherwise. Whether or not this instruction set is



**CPₖ:** {slaves in $DR \setminus \left(SU \cup \left(\bigcup_{k=1}^{3} PU_k\right)\right)$ with **C**urrent **P**ower level equal to **k**}

**SCPₖ:** {slaves in SU$\setminus \left(\bigcup_{k=1}^{3} PU_k\right)$ with **C**urrent **P**ower level equal to **k** }

$\mathrm{mp}_k$= cardinal$(CP_k)$ and $\mathrm{smp}_k$= cardinal$(SCP_k)$

**Fig. 15** *Venn diagram describing current slave power levels*

the shortest depends on the values of $s$, $\mathrm{sp}_k$, $p_k$, $m$ and $\mathrm{mp}_k$

$$L = \sum_{k=1}^{3} \mathrm{MDQ}\big(p_k + u(p_k)\cdot \mathrm{mp}_k\big) + \mathrm{MDQ}\left(\sum_{k=1}^{3}(1-u(p_k))\cdot \mathrm{mp}_k\right) + \mathrm{MDS}(s) \quad (4)$$

A valid set of instructions should satisfy the constraints below:

(i) Only slaves that are in DR will send data.
(ii) The transmission schedule that is dictated by the Task Manager is not violated.
(iii) Power level updates that are input by the Task Manager are not violated.

Note that if $n$, the number of recipient slaves for an instruction, is equal to the total number of slave nodes in the network, using BR_DATA_SCH and BR_DATA_REQ is more efficient than using MULTI_DATA_SCH and MULTI_DATA_REQ, because these are shorter instructions. Thus, in case they will be issued for all slaves in the network, MDS($n$) and MDQ($n$) should be replaced with $\mathrm{BDS}(n) = u(n)\cdot(n+2)$ and $\mathrm{BDQ}(n) = u(n)\cdot 2$, which are the lengths of BR_DATA_SCH and BR_DATA_REQ instructions including 1-byte of SSC length for $n$ slave nodes.

For each data request from the Task Manager, the Instruction Generator builds the Venn diagrams of Figs. 14 and 15, and considers 14 different solutions. The shortest instruction set is sent back to the Task Manager.

## 6 Conclusions

ACMENet is a TDMA-based master–slave network protocol that enables the master node to poll individual slave nodes for data, to request a scheduled data transfer from a subset of slave nodes and to poll a remote slave node for data via another slave node acting as a relay. ACMENet includes a rich set of instructions to allow data requests in various modulation scheme and power levels. The Second ACME Sea Trial demonstrated that ACMENet satisfies its design objectives in terms of polled, scheduled and relayed data retrievals. After the Second Sea Trial, ACMENet was augmented with an advanced ACME Controller that provided autonomous control of the network. The experiments in the Third Sea Trial were designed to demonstrate real-time and autonomous observation of the underwater environment using ACMENet. During the tests in an anechoic basin prior to the sea trial, it was demonstrated that ACMENet is capable of autonomously controlling an underwater acoustic sensor network that was made up of three slave nodes; however, because of a number of obstacles caused by experimental conditions, ACMENet could only be tested with a single slave node during the actual sea trial. The major reason behind the failure of the experiments was the very harsh conditions that damaged the slave nodes 1 and 3 burying them by sand quickly after the start of the sea trial. These slave nodes were re-deployed a couple of times; however, the time interval before they got buried again was too short to conduct any meaningful experiment. After two re-deployments, the slave modems were realised to be permanently damaged.

In addition to the sea trial, extensive laboratory tests and simulations have verified the stability of the ACMENet protocol. High packet loss ratios and extended link outages because of the acoustic noise generated by shipping traffic were observed to a have a significant detrimental effect on the performance of the prototype sensor network in the sea trial. Forward error correction can be incorporated into future sea trials of ACMENet to improve its resilience against transmission errors. In the face of extended link outages, the best that the network protocol can do is to temporarily avoid data retrievals to save slave batteries. The rule base within the PRC can be extended to include rules that deter data retrievals from slave nodes that persistently fail to transmit data over an extended period. Finally, random access can be incorporated in the future versions of the ACMENet protocol. With random access supported in the network, the edge nodes can detect increasing noise levels because of approaching ships and warn the rest of the network by randomly accessing the shared medium to broadcast very short CTRL packets.

## 7 References

1 Adams, A.E., and Açar, G.: 'A Master–slave protocol for underwater acoustic communication networks'. Proc. 7th European Conf. on Underwater Acoustics, ECUA 2004, Delft, July 2004, vol. 2, Paper No. 260, pp. 1229–1234

2 Adams, A.E., and Açar, G.: 'An acoustic network protocol for sub-sea sensor systems'. Proc. IEEE Oceans Conf., Brest, France, June 2005

3 Kilfoyle, D.B., and Baggeroer, A.B.: 'The state of the art in underwater acoustic telemetry', *IEEE J. Ocean. Eng.*, 2000, **25**, (1), pp. 4–27

4 Sozer, E.M., Stojanovic, M., and Proakis, J.G.: 'Underwater acoustic networks', *IEEE J. Ocean. Eng.*, 2000, **25**, (1), pp. 72–83

5 Codiga, D.L., Rice, J.A., and Baxley, P.A.: 'Networked acoustic modems for real-time data delivery from distributed subsurface instruments in the coastal ocean: initial system development and performance', *J. Atmos. Ocean. Technol.*, 2003, **21**, (2), pp. 331–346

6 Eggen, T.H., Baggeroer, A.B., and Preisig, J.C.: 'Communication over Doppler spread channels. Part I. Channel and receiver presentation', *IEEE J. Ocean. Eng.*, 2000, **25**, (1), pp. 62–71

7 Sharif, B.S., Neasham, J., Hinton, O.R., and Adams, A.E.: 'A computationally efficient doppler compensation system for underwater acoustic communications', *IEEE J. Ocean. Eng.*, 2000, **25**, (1), pp. 52–61

8 Akyildiz, I.F., Su, W., Sankarasubramaniam, Y., and Cayirci, E.: 'Wireless sensor networks: a survey', *Comput. Netw.*, 2002, **38**, (4), pp. 393–422

9 Green, M., Rice, J.A., and Merriam, S.: 'Underwater acoustic modem configured for use in a local area network (LAN)'. Proc. Oceans'98 Conf., Nice, France, vol. 2, pp. 634–638

10 Tsemenidis, C.C., Hinton, O.R., Adams, A.E., and Sharif, B.S.: 'Underwater acoustic receiver employing direct-sequence spread spectrum and spatial diversity combining for shallow-water multi-access networking', *IEEE J. Ocean. Eng.*, 2001, **26**, (4), pp. 594–603

11 Yeo, H.K., Sharif, B.S., Adams, A.E., and Hinton, O.R.: 'Performances of multi-element multi-user detection strategies in a shallow-water acoustic network (SWAN)', *IEEE J. Ocean. Eng.*, 2001, **26**, (4), pp. 604–611

12 Karn, P.: 'MACA–A new channel access method for packet radio'. ARRL/CRRL Amateur Radio 9th Computer Network Conf., London, Ontario, Canada, September 1990, pp. 134–140

13 Bhargavan, V., Demers, A., Shenker, S., and Zhang, L.: 'MACAW: a media access protocol for wireless LANs'. Proc. ACM SIGCOMM, London, UK, 1994

14 Peyravi, H.: 'Medium access control protocols performance in satellite communications', *IEEE Commun. Mag.*, 1999, **37**, (3), pp. 62–71

15 Açar, G., and Rosenberg, C.: 'Weighted fair bandwidth-on-demand (WFBoD) for geostationary satellite networks with on-board processing', *Comput. Netw.*, 2002, **39**, (1), pp. 5–20

16 Sohrabi, K., *et al.*: 'Protocols for Self-organization of a Wireless Sensor Network', *IEEE Pers. Commun.*, 2000, pp. 16–27

17 Woo, A., and Culler, D.: 'A transmission control scheme for media access in sensor networks'. Proc. ACM MobiCom'01, July 2001, pp. 221–235

18 Quazi, A.H., and Konrad, W.L.: 'Underwater acoustic communications', *IEEE Commun. Mag.*, 1982, **20**, (2), pp. 24–30

19 Catipovic, J., Brady, D., and Etchemendy, S.: 'Development of underwater acoustic modems and networks', *Oceanogr. Mag.*, 1994, **6**, pp. 112–119

20 Talavage, J.L., Thiel, T.E., and Brady, D.: 'An efficient store-and-forward protocol for a shallow-water acoustic local area network'. Proc. Oceans'94, Brest, France, pp. 883–888
21 Jubin, J., and Tornow, J.D.: 'The DARPA packet radio network protocols', *Proc. IEEE*, 1987, **75**, pp. 21–32
22 Rice, J.A.: 'Seaweb network for FRONT oceanographic sensors'. FY02 Annual Project Report – National Oceanographic Partnership Program
23 Codiga, D.L., Rice, J.A., and Bogden, P.S.: 'Real-time delivery of subsurface coastal circulation measurements from distributed instruments using networked acoustic modems'. Oceans'2000 MTS/IEEE Conf. Proc., Providence, RI, USA, vol. 1, pp. 575–582
24 Gibson, J., Larraza, A., Rice, J., Smith, K., and Xie, G.: 'On the impacts and benefits of implementing full-duplex communications links in an underwater acoustic network'. Proc. 5th Int. Mine Symposium, Naval Postgraduate School, Monterey, CA, USA, October 2002, pp. 204–213
25 http://www.ifremer.fr/sismer/program/acme/data.htm
26 Passerieux, J.M.: 'Experimental evaluation of DS-CDMA modulations for use in a shallow water acoustic communication network'. 7th European Conf. Underwater Acoustics (ECUA), Delft, Netherlands, June 2004

## 8 Appendix

### 8.1 ACME Sea Trials

The Second ACME Sea Trial was conducted for 9 days in Bay of Douarnenez, France, in September 2002. The primary objective of this trial is to demonstrate the capabilities of ACMENet protocol regarding to relayed, scheduled and polled data retrievals with various modulation schemes and power levels. Additional objectives were to analyse the acoustic characteristics of the communication channel in the bay in terms of noise levels, frequency spreading and time spreading. Also, environmental data such as sound-speed profiles and bathymetry were collected during the sea trial [25, 26].

Three bottom-mounted slave nodes were managed by a master modem that was deployed from a measurement ship.A human operator was responsible for initiating data retrievals and for managing slave power levels and network modulation scheme (i.e. the automatic version of the ACME Controller was not implemented at this time). The slave nodes were not equipped with actual sensors. Instead, the TT8v2 units at each slave node generated a 256-byte data packet with a pre-specified content. Numerous topologies covering ranges beyond 5000 m were tested in different parts of the bay, and the ACME Controller collected bit error ratio and packet loss statistics throughout the experiments. The modulation schemes employed during the Second ACME Sea Trial are shown in Table 1.

On the data link with MFSK-1 and MFSK-2 modulation schemes, there were two reasons for packet loss: framing errors and bit errors within data segment (i.e. payload) of data packets. Framing errors, which are caused by carrier loss, noise demodulation, multipath propagation and bit errors in packet header fields, result in the complete loss of the associated data packet. The ACME Controller could not perform bit error rate (BER) analysis on data packets that were lost due to framing errors, as it was impossible to detect individual bit errors in the data segment. Therefore, lost frames are not included in the BER analyses.

During the 9-day sea trial, 503 data packet receptions were attempted in MFSK-1 or MFSK-2 modulation scheme at one of the three defined power levels over ranges exceeding 5000 m. Sixty-one of these attempts failed because of framing errors. Out of 442 data packets that were received, 372 packets contained error-free data segments and the remaining 70 packets had a number of bit errors. In other words, 26% of all data packets were lost. The overall bit error rate was measured to be 0.003. Table 4 provides more detail on data transmission errors with respect to modulation types and transmission power levels. Framing errors were more frequently observed with MFSK-1 than with MFSK-2. Moreover, MFSK-2 performed better than MFSK-1 in terms of BER. However, as bit errors were observed in the form of burst errors in MFSK-1, this modulation scheme resulted in a higher ratio of error-free data segments than MFSK-2. As expected, overall packet loss ratio was higher with MFSK-2. The impact of the transmission power level on data transmission error statistics was less prominent. The power level was observed to have more decisive influence on the frequency of framing errors rather than BER performance.

In addition to scheduled and polled data retrievals, 147 relayed data retrievals were performed from remote slave nodes via a relay slave node with 34% success ratio. Various remote–relay slave pairs were tried during the experiments. No relayed data retrieval was achieved with the MFSK-1 modulation scheme and 42% of the transmissions were successful with the MFSK-2 modulation scheme. With MFSK-1, data reception at the relay slave node consistently failed on the return path. Table 5 summarises our results on relay experiments together with the results of synchronisation experiments.

The experiments during the Second Sea Trial demonstrated that the ACMENet protocol can be successfully implemented to perform relayed, scheduled and polled data retrievals from a number of slave nodes covering large coastal underwater areas in various topologies. However, the highly unreliable nature of the acoustic communication links in subsea environment significantly reduced the ratio of error-free data retrievals in relay scenario. Increasing the transmission power level had little impact on the packet loss characteristics. It is believed that the reliability of relayed data retrievals can be improved by implementing forward error correction at both the relay node and the master node. It is especially important using

**Table 4: Error statistics on the data link in the Second Sea Trial**

| | With respect to modulation type | | With respect to power level | | |
|---|---|---|---|---|---|
| | MFSK-1 | MFSK-2 | Min | Medium | Max |
| Framing error (%) | 13 | 5.5 | 8.7 | 5.8 | 4.6 |
| Data Segments in error (%) | 5 | 17 | 17 | 16 | 15 |
| Packet Loss (%) | 17 | 21 | 24 | 21 | 18 |
| BER ($\times 10^{-3}$) | 3.3 | 2.9 | 5.5 | 1.2 | 2.5 |

**Table 5: Error statistics for relay operations and synchronisation**

| | Success ratio |
|---|---|
| Synchronisation | 82% |
| Relayed synchronisation | 51% |
| Relayed data retrieval | 34%<br>With MFSK-1: 0%<br>With MFSK-2: 42% |

**Table 6: Sensor types in the Third Sea Trial**

| Slave | Sensor | Type | Output message size, bytes |
|---|---|---|---|
| S1 | OBS-3® | Turbidity | 10 bytes |
| S2 | Aandreaa® | Temperature | 8 bytes |
| S3 | ADCP (Acoustic Doppler Current Profiler) | Current flow | 346 bytes |

error correction at the relay node to decouple the error statistics on the remote-to-relay and the relay-to-master links.

The Third ACME Sea Trial was carried out for 18 days in September 2003 at a location in the Westerschelde Estuary, the Netherlands. The objective of this trial is to demonstrate the autonomous operation of a real-time underwater acoustic sensor network in a busy estuary using the ACMENet protocol. The underwater acoustic sensor network was integrated into the country-wide environmental measurement network of RWS in the Netherlands to provide current flow, turbidity and temperature measurements in the Westerschelde Estuary, and to make these measurements available on the Internet. Three slave nodes were deployed at key locations in the estuary and a master modem was mounted on a measurement pole at the shore near the town of Hansweert. The master modem was connected to the ACME Controller at RWS offices in Middelburg, 30 km away, via a telephone line. Table 6 provides information on the environmental sensor units contained at each slave node. The modulation schemes employed during the Third Sea Trial were presented in Table 1.

The actual sea trial commenced on 1st September. However, because of a series of problems regarding the experimental conditions in the Westerschelde, communication between the master modem and slaves 1 and 3 could not be maintained for any significant amount of time. The sand masses carried by the strong current flow in the estuary completely buried these two slave nodes. After two re-deployments, it was realised that slave nodes 1 and 3 were permanently damaged. Consequently, important capabilities of the ACMENet protocol such as relayed data retrieval and scheduled data retrieval could not be demonstrated in the Westerschelde. The PRC successfully performed autonomous control of the modulation scheme and power level at slave 2. Similarly, the Instruction Generator at the ACME Controller functioned satisfactorily. During the actual sea trial, the data retrieval success ratios could be as high as 80%; however, the communication between the master node and slave-2 experienced extended link outages while ships were passing nearby.

## 8.2 *Instruction generation solutions*

*Solution-1*: A MULTI_DATA_REQ_k for each non-empty $\mathrm{PU}_k$ and a MULTI_DATA_SCH_j for $\mathrm{DR}\backslash(\bigcup_{k=1}^{3}(\mathrm{PU}_k\backslash\mathrm{SU}))$.

$$\text{Total length:}\quad L=\sum_{k=1}^{3}\mathrm{MDQ}(p_k)+\mathrm{MDS}(s+m)$$

*Solution-2*: A MULTI_DATA_REQ_k for each non-empty $\mathrm{PU}_k$, a MULTI_DATA_REQ for $\mathrm{DR}\backslash(\mathrm{SU}\cup(\bigcup_{k=1}^{3}\mathrm{PU}_k))$, and a MULTI_DATA_SCH_j for SU.

$$\text{Total length:}\quad L=\sum_{k=1}^{3}\mathrm{MDQ}(p_k)+\mathrm{MDQ}(m)+\mathrm{MDS}(s)$$

*Solution-3*: A MULTI_DATA_REQ_k for each $(\mathrm{CP}_k\cup\mathrm{PU}_k)$ if $\mathrm{PU}_k\neq\{\}$ and a MULTI_DATA_SCH_jk for $\mathrm{SU}\cup\bigcup_{k=1,\mathrm{PU}_{k=\{\}}}^{3}\mathrm{CP}_k$.

$$\text{Total length:}\quad L=\sum_{k=1}^{3}\mathrm{MDQ}\big(p_k+u(p_k)\cdot\mathrm{mp}_k\big)+\mathrm{MDS}\left(s+\sum_{k=1}^{3}(1-u(p_k))\cdot\mathrm{mp}_k\right)$$

*Solution-4*: A MULTI_DATA_REQ_k for each $(\mathrm{CP}_k\cup\mathrm{PU}_k)$ if $\mathrm{PU}_k\neq\{\}$, a MULTI_DATA_REQ for $\bigcup_{k=1,\mathrm{PU}_{k=\{\}}}^{3}\mathrm{CP}_k$, and a MULTI_DATA_SCH_J for SU.

$$\text{Total length:}\quad L=\sum_{k=1}^{3}\mathrm{MDQ}\big(p_k+u(p_k)\cdot\mathrm{mp}_k\big)+\mathrm{MDS}\left(\sum_{k=1}^{3}(1-u(p_k))\cdot\mathrm{mp}_k\right)+\mathrm{MDS}(s)$$

*Solution-5*: A MULTI_DATA_REQ_k for each $\mathrm{PU}_k\backslash\mathrm{SU}$, a MULTI_DATA_SCH_jk for each $\mathrm{SU}\cap\mathrm{PU}_k$, and a MULTI_DATA_SCH_j for $\mathrm{DR}\backslash(\bigcup_{k=1}^{3}\mathrm{PU}_k)$.

$$\text{Total length: } L=\sum_{k=1}^{3}\mathrm{MDQ}\big(p_k-\mathrm{sp}_k\big)+\sum_{k=1}^{3}\mathrm{MDS}\big(\mathrm{sp}_k\big)+\mathrm{MDS}\left(s+m-\sum_{k=1}^{3}\mathrm{sp}_k\right)$$

*Solution-6*: A MULTI_DATA_REQ_k for each $(\mathrm{PU}_k\cup\mathrm{CP}_k)\backslash(\mathrm{SU}\cap\mathrm{PU}_k$ if $\mathrm{PU}_k\neq\{\}$, a MULTI_DATA_SCH_jk for each $\mathrm{SU}\cap\mathrm{PU}_k$ and a MULTI_DATA_SCH_j for $\mathrm{SU}\cup\bigcup_{k=1,\mathrm{PU}_{k=\{\}}}^{3}\mathrm{CP}_k$.

$$\text{Total length:}\quad L=\sum_{k=1}^{3}\mathrm{MDQ}(p_k+u(p_k)\cdot(mp_k-sp_k))+\sum_{k=1}^{3}\mathrm{MDS}\big(\mathrm{sp}_k\big)+\mathrm{MDS}\times\left(s+\sum_{k=1}^{3}(1-u(p_k))\cdot mp_k\right)$$

*Solution-7*: A MULTI_DATA_REQ_k for each $(\mathrm{PU}_k\cup\mathrm{CP}_k)/(\mathrm{SU}\cap\mathrm{PU}_k$ if $(\mathrm{PU}_k\backslash\mathrm{SU})\neq\{\}$, a MULTI_DATA_REQ for $\bigcup_{k=1,(\mathrm{PU}_k\backslash\mathrm{SU})=\{\}}^{3}\mathrm{CP}_k$, a MULTI_DATA_SCH_jk for each $\mathrm{SU}\cap\mathrm{PU}_k$ and a MULTI_DATA_SCH_j for

SU$\backslash(\bigcup_{k=1}^{3}(\mathrm{SU} \cap \mathrm{PU}_k))$.

Total length:

$$L = \sum_{k=1}^{3} \mathrm{MDQ}(p_k + u(p_k - sp_k) \cdot (mp_k - sp_k)) + \mathrm{MDQ}\left(\sum_{k=1}^{3}((1 - u(p_k - sp_k)) \cdot mp_k)\right) + \sum_{k=1}^{3} \mathrm{MDS}(\mathrm{sp}_k) + \mathrm{MDS}\left(s - \sum_{k=1}^{3} sp_k\right)$$

*Solution-8*: A MULTI_DATA_REQ_k for each $(\mathrm{PU}_k \cup \mathrm{CP}_k)$ if $\mathrm{PU}_k \neq \{\}$ and $(\mathrm{SU} \cap \mathrm{PU}_k) = \{\}$, a MULTI_DATA_SCH_jk for each $(\mathrm{PU}_k \cup \mathrm{CP}_k)$ if $(\mathrm{SU} \cap \mathrm{PU}_k) \neq \{\}$ and a MULTI_DATA_SCH_j for $\mathrm{SU} \cup [\bigcup_{k=1,\mathrm{PU}_{k=\{\}}}^{3} \mathrm{CP}_k]\backslash[(\bigcup_{k=1}^{3}(\mathrm{SU} \cap \mathrm{PU}_k)]$.

$$\text{Total length:} \quad L = \sum_{k=1}^{3}(u(p_k) \cdot (1 - u(\mathrm{sp}_k)) \times \mathrm{MDQ}(p_k + \mathrm{mp}_k) + u(\mathrm{sp}_k) \times \mathrm{MDS}(p_k + mp_k)) + \mathrm{MDS}\left(s + \sum_{k=1}^{3}((1 - u(p_k)) \times \mathrm{mp}_k) - \sum_{k=1}^{3} \mathrm{sp}_k\right)$$

*Solution-9*: A MULTI_DATA_REQ_k for each $(\mathrm{PU}_k \cup \mathrm{CP}_k)$ if $\mathrm{PU}_k \neq \{\}$ and $(\mathrm{SU} \cap \mathrm{PU}_k) \neq \{\}$ a MULTI_DATA_SCH_jk for each $(\mathrm{PU}_k \cup \mathrm{CP}_k)$ if $(\mathrm{SU} \cap \mathrm{PU}_k) \neq \{\}$, a MULTI_DATA_REQ for $\bigcup_{k=1,\mathrm{PU}_{k=\{\}}}^{3} \mathrm{CP}_k$ and a MULTI_DATA_SCH for $\mathrm{SU}\backslash(\bigcup_{k=1}^{3}(\mathrm{SU} \cap \mathrm{PU}_k))$.

$$\text{Total length:} \quad L = \sum_{k=1}^{3}(u(p_k) \cdot (1 - u(sp_k)) \times \mathrm{MDQ}(p_k + \mathrm{mp}_k) + u(\mathrm{sp}_k) \times \mathrm{MDS}(p_k + mp_k)) + \mathrm{MDQ}\left(\sum_{k=1}^{3}((1 - u(p_k)) \cdot \mathrm{mp}_k)\right) + \mathrm{MDS}\left(s - \sum_{k=1}^{3} sp_k\right)$$

*Solution-10*: A MULTI_DATA_REQ_k for each $\mathrm{PU}_k\backslash\mathrm{SU}$, a MULTI_DATA_SCH_jk for each $(\mathrm{SU} \cap \mathrm{PU}_k) \cup \mathrm{CP}_k \cup \mathrm{SCP}_k$ if $(\mathrm{SU} \cap \mathrm{PU}_k) \neq \{\}$ and a MULTI_DATA_SCH_j for $(\bigcup_{k=1,(\mathrm{SU}\cap\mathrm{PU}_k)=\{\}}^{3} \mathrm{SCP}_k) \cup (\bigcup_{k=1,(\mathrm{SU}\cap\mathrm{PU}_k)=\{\}}^{3} \mathrm{CP}_k)$.

$$\text{Total length:} \quad L = \sum_{k=1}^{3} \mathrm{MDQ}(p_k - \mathrm{sp}_k) + \sum_{k=1}^{3}(u(\mathrm{sp}_k) \cdot \mathrm{MDS}(sp_k + mp_k + smp_k)) + \mathrm{MDS}\left(\sum_{k=1}^{3}((1 - u(\mathrm{sp}_k)) \cdot (\mathrm{smp}_k + \mathrm{mp}_k))\right)$$

*Solution-11*: A MULTI_DATA_REQ_k for each $(\mathrm{PU}_k \cup \mathrm{CP}_k)/(\mathrm{SU} \cap \mathrm{PU}_k)$ if $\mathrm{PU}_k \neq \{\}$, a MULTI_DATA_SCH_jk for each $(\mathrm{SU} \cap \mathrm{PU}_k) \cup \mathrm{SCP}_k$ if $(\mathrm{SU} \cap \mathrm{PU}_k) \neq \{\}$ and a MULTI_DATA_SCH_j for $(\bigcup_{k=1,(\mathrm{SU}\cap\mathrm{PU}_k)=\{\}}^{3} \mathrm{SCP}_k) \cup (\bigcup_{k=1,\mathrm{PU}_k=\{\}}^{3} \mathrm{CP}_k)$.

$$\text{Total length:} \quad L = \sum_{k=1}^{3}(u(p_k) \cdot \mathrm{MDQ}(p_k + \mathrm{mp}_k - \mathrm{sp}_k)) + \sum_{k=1}^{3}(u(\mathrm{sp}_k) \cdot \mathrm{MDS}(\mathrm{sp}_k + \mathrm{smp}_k)) + \mathrm{MDS}\left(\sum_{k=1}^{3}((1 - u(\mathrm{sp}_k)) \cdot \mathrm{smp}_k) + \sum_{k=1}^{3}((1 - u(p_k)) \cdot \mathrm{mp}_k)\right)$$

*Solution-12*: A MULTI_DATA_REQ_k for each $(\mathrm{PU}_k \cup \mathrm{CP}_k)/(\mathrm{SU}_k \cap \mathrm{PU}_k)$ if $(\mathrm{PU}_k/\mathrm{SU}) \neq \{\}$, a MULTI_DATA_REQ for $\bigcup_{k=1,(\mathrm{PU}_k/\mathrm{SU})=\{\}}^{3} \mathrm{CP}_k$ a MULTI_DATA_SCH_jk for each $(\mathrm{SU} \cap \mathrm{PU}_k) \cup \mathrm{SCP}_k$ if $(\mathrm{SU} \cap \mathrm{PU}_k) \neq \{\}$ and a MULTI_DATA_SCH_j for $\bigcup_{k=1,(\mathrm{SU}\cap\mathrm{PU}_k)=\{\}}^{3} \mathrm{SCP}_k$.

$$\text{Total length:} \quad L = \sum_{k=1}^{3}(u(p_k - \mathrm{sp}_k) \cdot \mathrm{MDQ}(p_k + \mathrm{mp}_k - \mathrm{sp}_k)) + \mathrm{MDQ}\left(\sum_{k=1}^{3}((1 - u(p_k - \mathrm{sp}_k)) \cdot \mathrm{mp}_k)\right) + \mathrm{MDS}\left(\sum_{k=1}^{3}((1 - u(\mathrm{sp}_k)) \cdot \mathrm{smp}_k)\right) + \sum_{k=1}^{3}(u(\mathrm{sp}_k) \cdot \mathrm{MDS}(\mathrm{sp}_k + \mathrm{smp}_k))$$

*Solution-13*: A MULTI_DATA_REQ_k for each $(\mathrm{PU}_k \cup \mathrm{CP}_k)$ if $\mathrm{PU}_k \neq \{\}$ and $(\mathrm{SU} \cap \mathrm{PU}_k) = \{\}$, a MULTI_DATA_SCH_jk for each $(\mathrm{PU}_k \cup \mathrm{CP}_k \cup \mathrm{SCP}_k)$ if $(\mathrm{SU} \cap \mathrm{PU}_k) \neq \{\}$ and a MULTI_DATA_SCH_j for $(\bigcup_{k=1,(\mathrm{SU}\cap\mathrm{PU}_k)=\{\}}^{3} \mathrm{SCP}_k) \cup (\bigcup_{k=1,\mathrm{PU}_k=\{\}}^{3} \mathrm{CP}_k)$.

Total length:

$$L = \sum_{k=1}^{3}(u(p_k) \cdot (1 - u(\mathrm{sp}_k)) \cdot \mathrm{MDQ}(p_k + \mathrm{mp}_k)) + \sum_{k=1}^{3}(u(\mathrm{sp}_k) \cdot \mathrm{MDS}(p_k + \mathrm{smp}_k + \mathrm{mp}_k)) + \mathrm{MDS}\left(\sum_{k=1}^{3}((1 - u(\mathrm{sp}_k)) \cdot \mathrm{smp}_k) + \sum_{k=1}^{3}((1 - u(p_k)) \cdot \mathrm{mp}_k)\right)$$

*Solution-14*: A MULTI_DATA_REQ_k for each $(\mathrm{PU}_k \cup \mathrm{CP}_k)$ if $\mathrm{PU}_k \neq \{\}$ and $(\mathrm{SU} \cap \mathrm{PU}_k) = \{\}$ a MULTI_DATA_SCH_jk for each $\mathrm{PU}_k \cup \mathrm{CP}_k \cup \mathrm{SCP}_k)$ if $(\mathrm{SU} \cap \mathrm{PU}_k) \neq \{\}$, a MULTI_DATA_REQ for $\bigcup_{k=1,\mathrm{PU}_{k=\{\}}}^{3} \mathrm{CP}_k$ and a MULTI_DATA_SCH_j for $\bigcup_{k=1,(\mathrm{SU}\cap\mathrm{PU}_k)=\{\}}^{3} \mathrm{SCP}_k$.

Total length:

$$L = \sum_{k=1}^{3}(u(p_k) \cdot (1 - u(\mathrm{sp}_k)) \cdot \mathrm{MDQ}(p_k + \mathrm{mp}_k))$$

$$+ \sum_{k=1}^{3}(u(\mathrm{sp}_k) \cdot \mathrm{MDS}(p_k + \mathrm{smp}_k + \mathrm{mp}_k))$$

$$+ \mathrm{MDQ}\left(\sum_{k=1}^{3}((1 - u(p_k)) \cdot \mathrm{mp}_k)\right)$$

$$+ \mathrm{MDS}\left(\sum_{k=1}^{3}((1 - u(\mathrm{sp}_k)) \cdot \mathrm{smp}_k)\right)$$

Each of these solutions is tested by the Instruction Generator. Note that the functions MDS($n$) and MDQ($n$) in the equations in all solutions listed above are replaced by BDS($n$) and BDQ($n$) when the input argument, $n$, is equal to the total number of slaves in the network. The solution that yields the shortest total length of instructions is chosen.