Baseband Signal Processing with UnetStack

Mandar Chitre[*†], Rohit Bhatnagar[†], Manu Ignatius[‡], and Shailabh Suman[†]

[*] Department of Electrical & Computer Engineering, National University of Singapore.
[†] ARL, Tropical Marine Science Institute, National University of Singapore.
[‡] Subnero Pte Ltd, Singapore.

***Abstract*—Software-defined underwater modems implement key components of an underwater communication system in software, rather than hardware, and allow engineers and researchers to modify those components dynamically. These components include not only high-level network protocols, but also physical layer signal processing chains, modulation techniques and error-correction codes. The UnetStack project provides a flexible platform for software-defined open-architecture underwater modems and networks. We outline the software-defined modem functionality in UnetStack that enables users to process baseband acoustic signals directly. We demonstrate several practical applications of baseband signal processing, including an implementation of a software-defined OFDM modem, a delay-Doppler channel estimator, and an underwater vehicle to transponder range estimator.**

I. Introduction

SOFTWARE-defined radios (SDR) have received plenty of attention over the past two decades. The defining idea in SDR is to use software implementations of key components in a communication system, rather than a more traditional approach of implementing them in hardware [1]. The benefit is flexibility. SDR allows a single communication device to support multiple protocols. It also enables devices to adaptively change the communication technique used in response to the environment. This ability is of particular importance in challenging communication channels.

The underwater acoustic communication channel is known to be a challenging one [2]. The benefits that SDR brings to wireless communication are even more desirable in underwater acoustic modems. This gives rise a *software-defined modem* (SDM) – a device in which the communication protocols, modulation/demodulation, detection, error correction, etc is implemented in software. As researchers and engineers learn from experiments and develop new techniques to combat the challenges of the underwater environment, they are able to easily implement and test these changes in software in the modem (e.g. [3]). Since SDMs also offer the possibility of dynamically changing the communication techniques in response to changes in the environment, they can provide optimal performance in a changing ocean environment. Underwater modems today are typically expensive devices. The ability to upgrade the performance of a modem through software changes also provides a much-desired protection of investment on the modem.

The *UnetStack* project[1] provides a framework for *software agents* that collectively form an underwater network stack [4]. It also provides a collection of agents that implement various application, transport, routing, medium-access and physical layer protocols. The stack is highly extensible, and allows the user to develop, test and deploy new agents to customize the behavior of the stack, or add new functionality or protocols into the stack. UnetStack can be operated in a *simulator* mode to test new agents and to conduct underwater network simulations. Once an agent is ready and tested, it can simply be copied to a modem with UnetStack support for deployment in a real-world network.

UnetStack has explicit support for SDMs, and therefore agents are not only able to add new network functionality, but also able to introduce new signal processing chains, modulation schemes, and error-correction codes into the modem. In this paper, we outline this the baseband signal processing functionality of the UnetStack and show several practical applications. The applications include a software-defined OFDM modem, a delay-Doppler channel estimator and an autonomous underwater vehicle (AUV) to transponder range estimator.

II. The UnetStack Architecture

UnetStack consists of a set of software agents that provide well-defined *services*, and work together to form a complete underwater networking solution. Agents play the role that layers play in traditional network stacks. Since the agents are not organized in any enforced

[1]http://www.unetstack.net

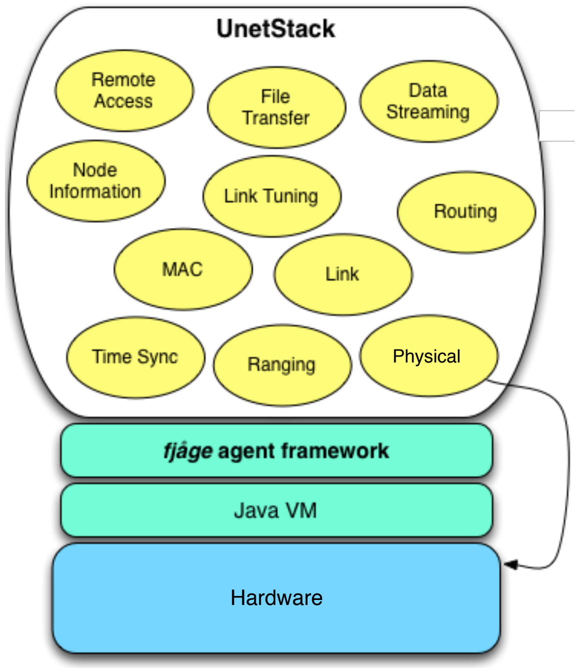

Fig. 1. The UnetStack architecture. Several agents providing common underwater network services are shown. The *physical* agent provides bindings for underwater modems and for underwater network simulation. It also optionally supports the *baseband* service that allows customization of the signal processing chain in the modem. The stack runs on a Java virtual machine and the fjåge open-source agent framework.

hierarchy, they are free to interact in any suitable way. This promotes low-overhead protocols and cross-layer information sharing. As shown in Fig. 1, UnetStack defines the interfaces for commonly needed agents in terms of the services, messages, capabilities and parameters that the agent must or may provide. Extensibility is at the heart of UnetStack, thus allowing agents to provide additional services needed by cross-layer optimized protocols. Wired and wireless radio links can also easily be included as part of a UnetStack-based network.

UnetStack is developed using Java, Groovy[2] and the open-source fjåge lightweight agent framework[3]. New agents are typically developed in Groovy or Java, although occasionally developers may resort to C (or other native languages) for direct hardware access, use of specialized native libraries, or for performance-critical optimized code. Since UnetStack supports distributed agents, it is also possible to run computationally intensive agents on a separate co-processor or a research laptop connected to the modem.

For further information on developing using UnetStack, we refer the interested reader to [4].

[2]Groovy (http://groovy.codehaus.org) is an easy-to-learn dynamic language based on the Java Virtual Machine.

[3]https://github.com/org-arl/fjage

III. Baseband Processing in UnetStack

Our focus in this paper is on UnetStack's ability to allow researchers and engineers to customize the low-level signal processing in the modem. This ability is supported through the *baseband* service that a *physical* agent may choose to provide. Currently, the drivers for the ARL UNET-II modem [5] and the Subnero modem[4], as well as the freely downloadable network simulator support the baseband service. Other modems with arbitrary waveform capabilities can be supported through development of appropriate drivers.

At the heart of the baseband service, we have two key messages – `TxBasebandSignalReq`, and `RxBasebandSignalNtf`. The first allows an agent to generate a baseband signal to transmit, while the second allows the agent to receive and process a baseband signal from a peer modem. The passband-baseband conversion is according to the `carrierFrequency` and `samplingRate` parameters defined in the messages. Of course, the parameters must be within the allowable range supported by the service/hardware.

Modems typically detect incoming frames using a *detection preamble* with good autocorrelation properties. These detection preambles are typically short (a few milliseconds in duration) and are followed by a longer data-carrying signal. A modem supporting the *baseband* service usually reserves one or more detection preambles for use by SDM agents. When a `TxBasebandSignalReq` is received by the baseband service provider, a signal consisting of an optional preamble (as specified by the `preambleID` parameter) followed by the signal contained in the message is transmitted. When a peer modem detects the preamble, it starts a signal acquisition for a predefined duration (`maxSignalLength` parameter of the baseband service provider). Once the signal acquisition is completed, it generates a `RxBasebandSignalNtf` message on its default notification topic. Any SDM agent interested in the signal subscribes to the topic and processes the notification message. In case the signal of interest does not have a recognizable preamble but its arrival time is known, signal acquisition can be triggered in software using the `RecordBasebandSignalReq`.

Ranging and other time-sensitive applications sometimes benefit from baseband processing as well. The transmission/reception time of the signals is captured by the baseband service. The reception time is recorded in the `rxTime` parameter of the

[4]http://www.subnero.com/

RxBasebandSignalNtf. The transmission time is published using a TxBasebandSignalNtf message on the default notification topic. If a signal is desired to be sent at a specific time, the txTime parameter in the TxBasebandSignalReq may be set. The signal transmission is triggered at the specified time provided the service/hardware supports the *TIMED_TX* capability.

While Java and Groovy have good support for basic mathematics, signal processing often involves linear algebra over complex numbers. There are numerous great libraries (e.g., jBLAS[5] and GroovyLab[6] for linear algebra, JTransforms[7] for FFT, etc) available for Java and Groovy for this purpose. These libraries can be effortlessly used from UnetStack agents or scripts. In some cases, users may benefit from the use of GNU radio [6] to implement the signal processing. Although GNU radio may be integrated with UnetStack's baseband functionality via the Java native interface (JNI), we have not tested the two frameworks together.

IV. Applications

To illustrate the use of the baseband processing in UnetStack, we present three different applications. The first is a typical SDM application where UnetStack is used to prototype an OFDM modem. The next two applications are signal processing applications, and demonstrate the flexibility afforded by UnetStack.

A. *Software-defined OFDM Modem*

Let us assume that the modem in use does not support OFDM, and so the default physical layer (represented by agent "phy") provides the *datagram* and *physical* services to higher layer agents based on some other modulation scheme (e.g. FSK). It, however, does support UnetStack with its baseband service. To illustrate how this can be used to prototype SDMs, we consider the addition of a rather simple OFDM modem functionality to this modem.

The OFDM SDM is implemented as an agent named "OFDM". The agent works closely with phy, which provides the baseband service. As part of the physical service that phy provides, it supports several messages such as TxFrameReq, TxFrameNtf and RxFrameNtf for FSK communication. In order to support OFDM communications, the OFDM agent has to support the same set of messages (and some associated parameters) for OFDM communication. It effectively translates between these messages and the equivalent TxBasebandSignalReq, RxBasebandSignalNtf and TxBasebandSignalNtf messages as shown in Fig. 2. This primarily involves converting the data bits in the TxFrameReq into a baseband signal in the TxBasebandSignalReq, and the signal in the RxBasebandSignalNtf into data bits in the RxFrameNtf.

The entire source code for the OFDM agent is available in the UnetStack simulator download[8] under the samples/baseband folder. To illustrate the simplicity of the process, we present an extract of the code that

[5]http://mikiobraun.github.io/jblas/

[6]http://groovy.codehaus.org/GroovyLab

[7]https://sites.google.com/site/piotrwendykier/software/jtransforms

[8]http://www.unetstack.net

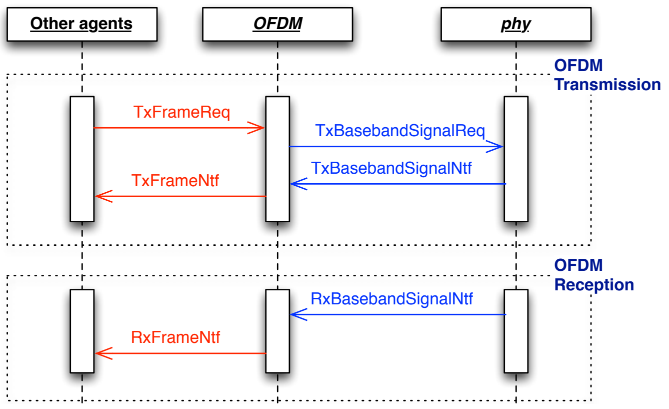

Fig. 2. Sequence diagram illustrating the translation of messages between the physical and baseband services by the OFDM agent.

```
def processTX(DatagramReq req) {
  // assemble 4 byte header followed by data
  BitBuffer buf = new BitBuffer(req.data.
      length + HDRSIZE)
  buf.write([node.address, req.to, req.
      protocol, req.data.length] as byte[])
  buf.write(req.data)
  buf.reset()
  // convert bits to baseband BPSK-OFDM
  int L = (buf.sizeInBits+nc-1)/nc
  def sig = new float[2*L*(nc+np)]
  FloatFFT_1D fft = new FloatFFT_1D(nc)
  def bpsk = [1: -1, 0: +1]
  for (int j = 0; j < L; j++) {
    int blkStart = 2*j*(nc+np) + 2*np
    for (int k = 0; k < nc; k++)
      sig[blkStart+2*k] = bpsk[buf.read()]?:0
    fft.complexForward(sig, blkStart)
    System.arraycopy(sig, blkStart+2*(nc-np),
        sig, blkStart-2*np, 2*np)
  }
  return new TxBasebandSignalReq(signal: sig)
}
```

Fig. 3. Code extract from the OFDM agent, showing how data bits from a TxFrameReq (or equivalently a DatagramReq) are translated into an OFDM baseband signal in TxBasebandSignalReq.

converts the data bits into an OFDM baseband signal in Fig. 3. The `OFDM` agent can be either instantiated in the modem or on a co-processor/laptop connected to the modem.

B. Delay-Doppler Channel Estimation

When working in the field with modems, it is often useful to have an estimate of the communication channel that the modems are dealing with. This channel estimate may be represented as a delay-Doppler plot. During the MISSION 2013 underwater networking experiment held in Singapore in November 2013, we deployed 7 static UNET nodes and 2 AUVs in the water. Of the 7 static nodes, one was at a central location and connected to a laptop on the surface, while the others were bottom-mounted and autonomous. We implemented the ability to display the delay-Doppler estimate on the laptop in real-time, for a link from any node to the central node. With UnetStack's distributed agent capability, the delay-Doppler estimation agent uses the computational power of the laptop while receiving signals directly from the baseband service in the modem.

When an estimate from node N to the central node was desired, we used the *remote access* capability in UnetStack to invoke a script on node N. The script simply made a `TxBasebandSignalReq` request to the baseband service, asking a 650 ms m-sequence to be transmitted along with the default preamble. On the laptop at the central node, we ran a "`ddplot`" agent that subscribed to the notification topic of the baseband service. Whenever the incoming preamble was detected, a signal was acquired for about 650 ms and a corresponding `RxBasebandSignalNtf` was sent on the notification topic. On receiving this, the `ddplot` agent matched-filtered the signal in the message against various delayed (from about -2 ms to 10 ms in steps of 59 μs) and Doppler-scaled ($\pm$15 Hz in steps of 1 Hz) copies of the original m-sequence. The resulting matrix was linear interpolated by a factor of 4 (or optionally 8) and displayed with a virtual colormap on the laptop screen. An sample screenshot is shown in Fig. 4.

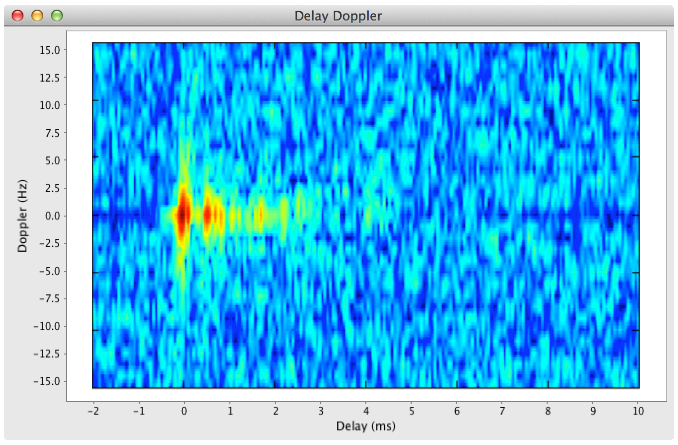

Fig. 4. A screenshot of the delay-Doppler estimate from node 29 to node 21 (central node) during the MISSION 2013 experiment, as displayed by the `ddplot` agent.

C. AUV-to-transponder Range Estimation

The STARFISH AUV [7] is equipped with the ARL UNET-2 modem. Certain search missions undertaken by the AUV require it to locate an underwater transponder by sequential ranging. Rather than use an additional payload to interrogate the transponder, we simply choose to use the modem on the AUV to find the range to the transponder. This is possible since the modem runs UnetStack with the baseband service, and the PXA320 processor on the UNET-2 modem has sufficient computational power for the signal processing.

We use an Applied Acoustics 219A transponder set to receive at 22 kHz and respond at 30.5 kHz after 30 ms. The transponder query agent "`interrogator`" running in the modem starts by sending a 5 ms long 22 kHz pulse without preamble (`preambleID` set to 0) using a `TxBasebandSignalReq`. It closely follows this request with a 300 ms recording request `RecordBasebandSignalReq`, ensuring that the modem begins recording immediately after the pulse is transmitted, and records long enough to receive the return (the transponder is expected to be within a range of 200 m of the AUV in our mission). The agent then receives two notifications – `TxBasebandSignalNtf` providing the exact time (μs accuracy) of transmission of the pulse, and `RxBasebandSignalNtf` providing the exact time of the recording as well as the recorded signal. The signal is bandpass filtered, Hilbert transformed and thresholded to find the start of the 30.5 kHz response pulse. Given the transmission time, recording time and the delay of the pulse in the recording, the agent computes the range to the transponder and publishes it on its notification topic. The command and control agents in the STARFISH AUV subscribe to this topic and use the range information for path-planning to eventually arrive at the transponder.

V. Conclusions

The UnetStack provides a flexible environment for the development of software-defined modems, networks and other underwater applications. In this paper, we outlined the *baseband* service in UnetStack that allows low-level access to acoustic signals in the modem, and demonstrated three practical applications of this service.

References

[1] M. Dillinger, K. Madani, and N. Alonistioti, *Software defined radio: Architectures, systems and functions*. John Wiley & Sons, 2005.

[2] M. Chitre, S. Shahabudeen, and M. Stojanovic, “Underwater acoustic communications and networking: Recent advances and future challenges,” *The Spring 2008 MTS Journal, ”The State of Technology in 2008”*, vol. 42, no. 1, pp. 103–116, 2008.

[3] H. Yan, L. Wan, S. Zhou, Z. Shi, J.-H. Cui, J. Huang, and H. Zhou, “DSP based receiver implementation for OFDM acoustic modems,” *Physical Communication*, vol. 5, no. 1, pp. 22–32, 2012.

[4] M. Chitre, R. Bhatnagar, and W.-S. Soh, “UnetStack: an agent-based software stack and simulator for underwater networks,” in *Proceedings of IEEE OCEANS’14 St John’s*, September 2014.

[5] M. Chitre, I. Topor, and T.-B. Koay, “The UNET-2 modem – an extensible tool for underwater networking research,” in *Proceedings of IEEE OCEANS’12 Yeosu*, May 2012.

[6] E. Blossom, “GNU radio: tools for exploring the radio frequency spectrum,” *Linux journal*, vol. 2004, no. 122, p. 4, 2004.

[7] T.-B. Koay, Y.-T. Tan, Y.-H. Eng, R. Gao, M. Chitre, J.-L. Chew, N. Chandhavarkar, R. Khan, T. Taher, and J. Koh, “STARFISH – a small team of autonomous robotic fish,” *Indian Journal of geo-Marine Science*, vol. 40, no. 2, pp. 157–167, April 2011.