# Tutorial on Underwater Acoustic Networking

**Mandar Chitre**
**Shiraz Shahabudeen**
**Prasad Anjangi**
**Chinmay Pendharkar**



28 May 2018

# Overview

1. Introduction
2. Network stack/simulators
3. Physical layer
4. Introduction to UnetStack
5. Medium Access Control (MAC)
6. Higher layers
7. Propagation delay
8. Other research topics

Section 1

# Introduction

# There is a demand for underwater networks!

- Wired underwater networks:

  SOSUS (1949-), OOI (2009-), NEPTUNE (2009-), VENUS (2006-), etc.
- Clear demand for underwater networks, but communication and power needs necessitated cables
- Wired networks are expensive, and impractical for short-term deployments
- Underwater point-to-point wireless communication technology is maturing
- Interest in underwater wireless networks is rapidly growing

# Communication links in underwater network

| Technology | Range | Data rate | Latency | Setup | Mobility |
|---|---|---|---|---|---|
| Fiber-optic cables | $O$(100 km) | $O$(Gbps) | $O(\mu s)$ | hard | limited |
| Underwater acoustic | $O$(km) | $O$(kbps) | $O$(s) | easy | good |
| Underwater optical | $O$(10 m) | $O$(Mbps) | $O(\mu s)$ | easy | good |
| Underwater EM | $O$(10 m) | $O$(kbps) | $O(\mu s)$ | easy | good |

- Choice of links depends on application/environment
- For nodes with surface expression, in-air radio links may also be used
- Practical networks may choose a mix of types of links
- Often networks are $O$(km) in size $\Rightarrow$ acoustic links
- Acoustic link performance depends on carrier frequency

  (low freq. $\rightarrow$ long range, low rate; high freq. $\rightarrow$ short range, high rate)

# What's special about underwater acoustic networks?

- Limited bandwidth $\Rightarrow$ Low data rate
- Slow speed of sound $\Rightarrow$ Long propagation delay
- Environmental variability & noise $\Rightarrow$ Packet loss
- Environmental variability $\Rightarrow$ Changing network topology

# Low data rate – what do we do about it?

- Bandwidth-efficient protocols needed
- Limited sizes of protocol headers
- Efficiency through cross-layer optimization

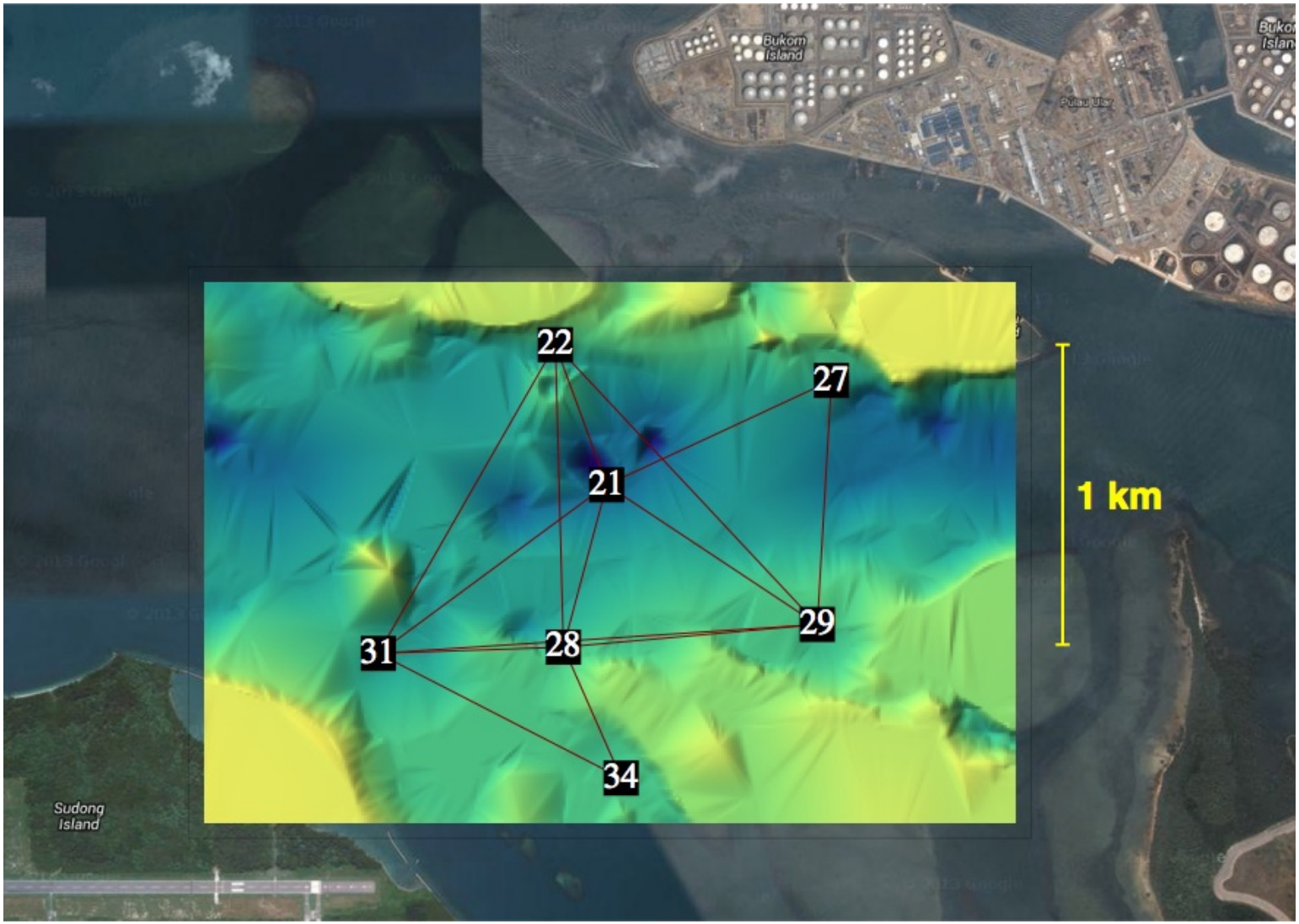
- If large data transfer is unavoidable, novel strategies such as *data muling* have to be employed

# Long propagation delay – how does it affect us?

- It's not just a question of adjusting protocol timeouts!
- Example: naïve TDMA with guard period
  (TDMA slot length: 1 s, guard period $>$ maximum propagation delay, for a 10 km network, guard period $\sim$ 7 s, channel utilization $\sim$ 12.5%)
- Example: naïve stop-and-wait ARQ
  (packet length: 1 s, transmission range: 9 km, waiting time: 12 s, channel utilization $<$ 8%)
- In more extreme cases, protocols may completely fail
  (example: adaptive modulation with feedback may fail on long range links if the channel changes over the period that the feedback arrives)
- Opportunity: multiple simultaneous transmissions without collision!

# MISSION 2013 Experiment

# MISSION 2013 Experiment

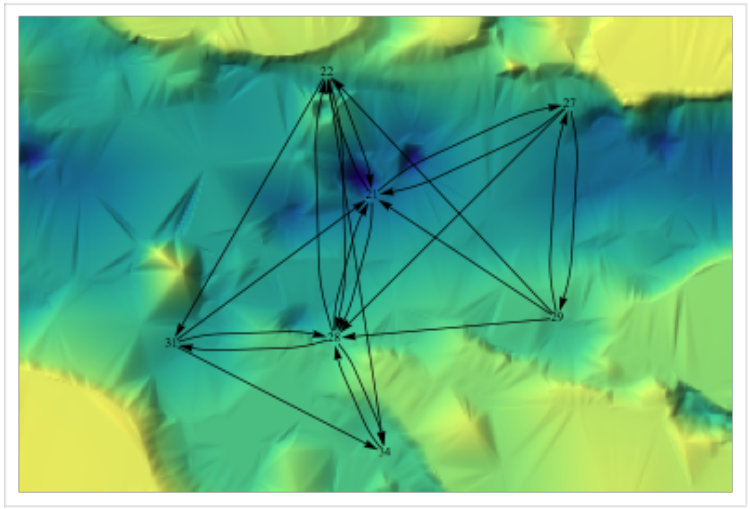


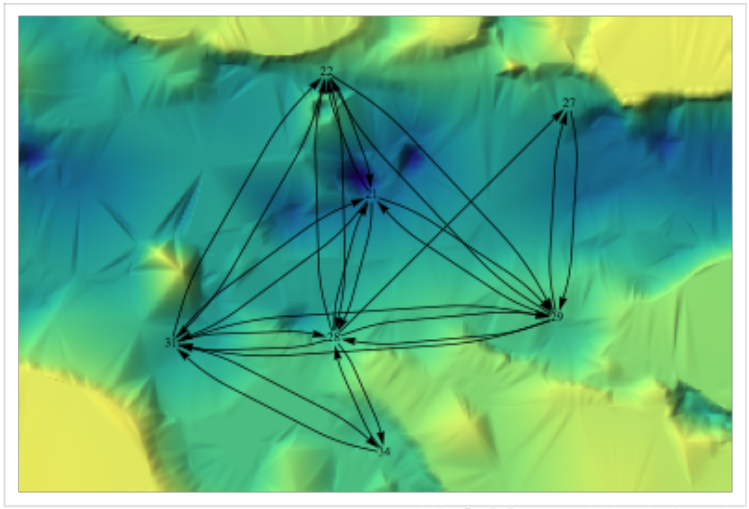
# Packet loss

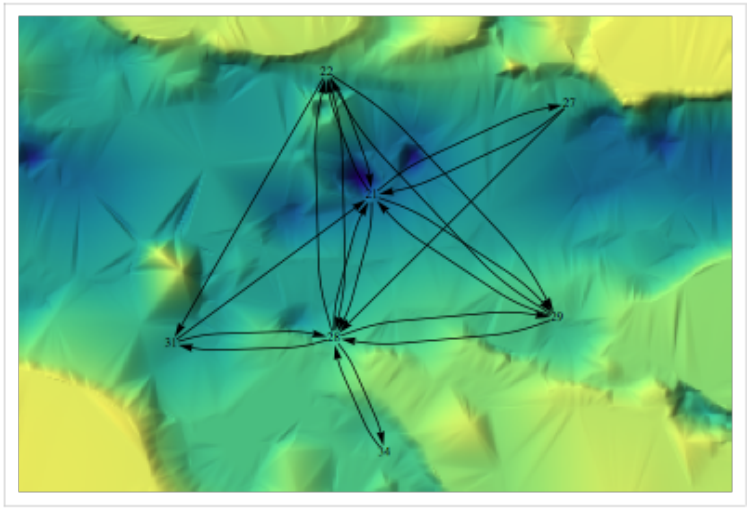
MISSION 2013 packet success rates:

| from/to | 21 | 22 | 27 | 28 | 29 | 31 | 34 |
|---|---|---|---|---|---|---|---|
| 21 | - | 0.926 | 0.266 | 0.917 | 0.912 | - | 0.552 |
| 22 | 0.867 | - | 0.471 | 0.751 | 0.850 | - | 0.288 |
| 27 | 0.359 | 0.381 | - | 0.313 | 0.322 | - | - |
| 28 | 0.847 | 0.869 | 0.390 | - | 0.845 | 0.925 | 0.863 |
| 29 | 0.539 | 0.693 | 0.333 | 0.688 | - | 0.374 | - |
| 31 | - | - | - | 0.902 | 0.805 | - | 0.795 |
| 34 | 0.236 | 0.436 | - | 0.684 | - | 0.544 | - |

# Changing network topology

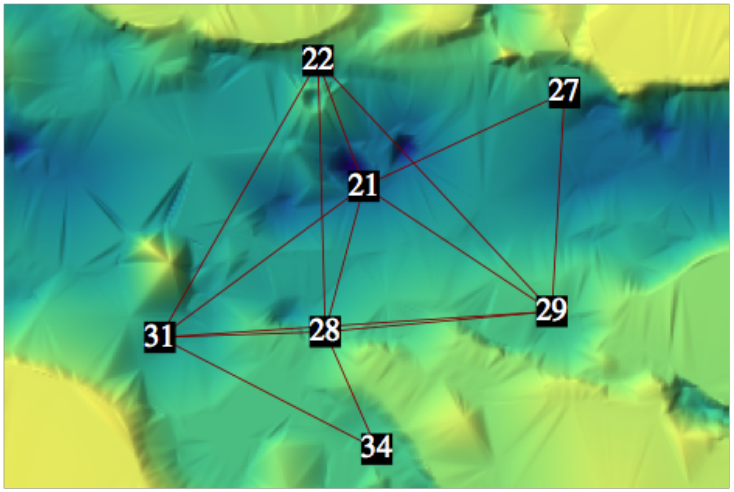


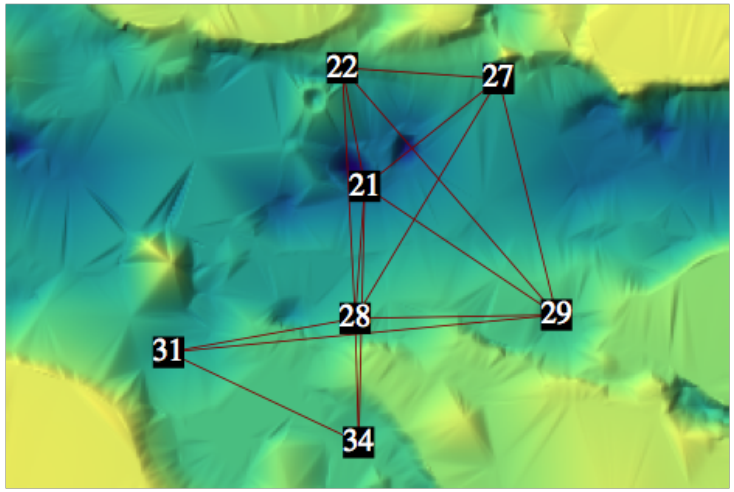



# Changing network topology

Deployment #1 vs #2:



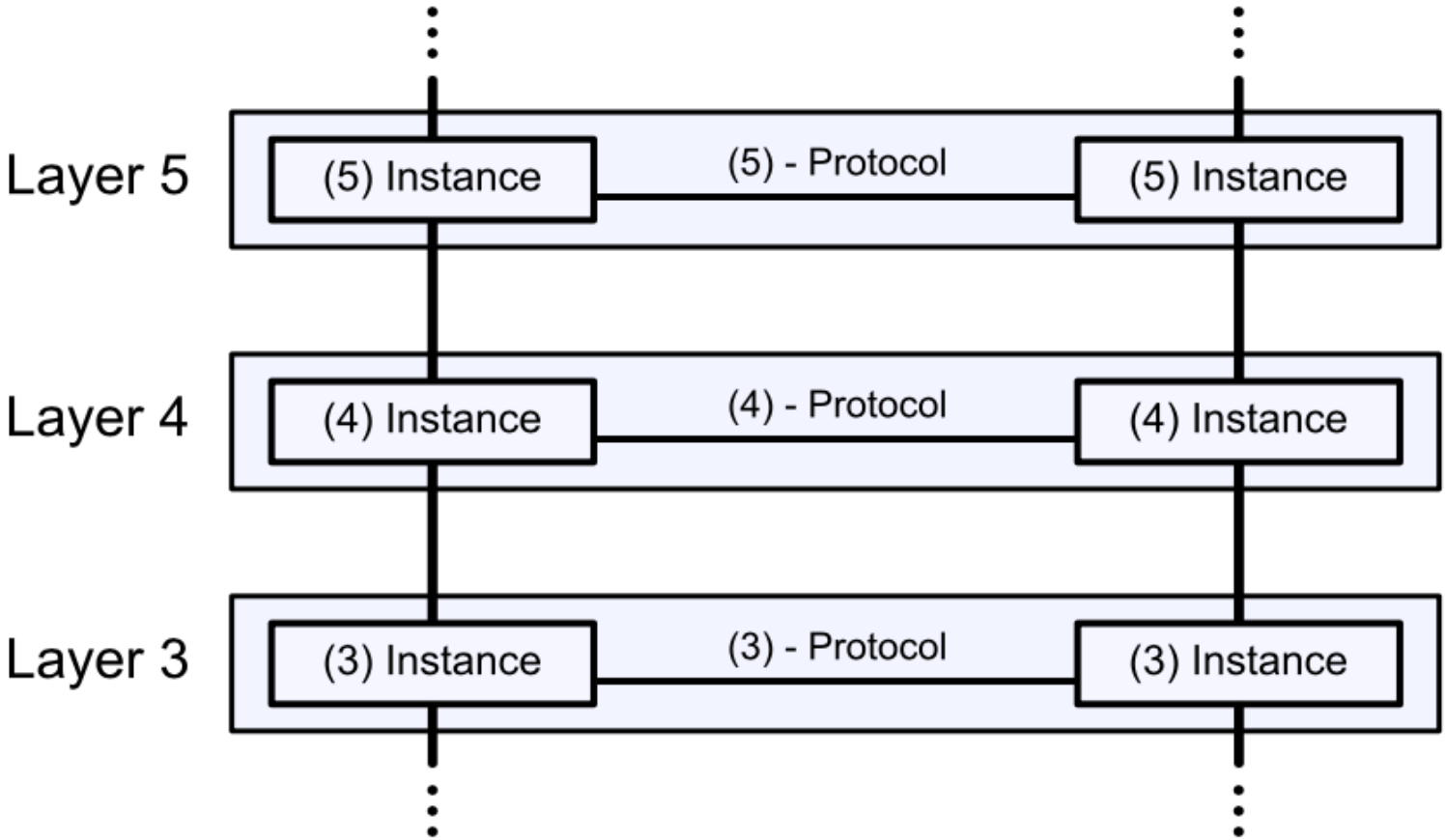




Minor changes in node positions significantly changes network topology

Section 2

# Network stack/simulators

# Layered model of network protocol stack



# OSI 7-layered model

| Layer | Data unit | Function |
|---|---|---|
| 7. Application | Data | High-level APIs, including resource sharing, remote file access, directory services and virtual terminals |
| 6. Presentation | Data | Translation of data between a networking service and an application; including character encoding, data compression and encryption/decryption |
| 5. Session | Data | Managing communication sessions, i.e., continuous exchange of information in the form of multiple back-and-forth transmissions between two nodes |
| 4. Transport | Datagram | Reliable transmission of data segments between points on a network, including segmentation, acknowledgement and multiplexing |
| 3. Network | Packet | Structuring and managing a multi-node network, including addressing, routing and traffic control |
| 2. Data link | Frame | Reliable transmission of data frames between two nodes connected by a physical layer |
| 1. Physical | Bit | Transmission and reception of raw bit streams over a physical medium |

http://en.wikipedia.org/wiki/OSI_model

# Internet does not follow the OSI 7-layer model

- TCP/IP does not use strict hierarchical encapsulation and layering

**Rough mapping**:

- Internet application layer → OSI application, presentation, session
- Internet transport layer → OSI session, transport
- Internet layer → OSI network
- Internet link layer → OSI network, data link, physical

# Underwater Networking Architecture (UNA)

- 5-layered architecture proposed, based on OSI-like layered model
- Application, Transport, Network, Data link, Physical layers
- Strict layered architecture, with no cross-layer functionality
- Implemented and used in initial versions of the UNET modems, but eventually retired in favor of an agent-based network stack (UnetStack)

---

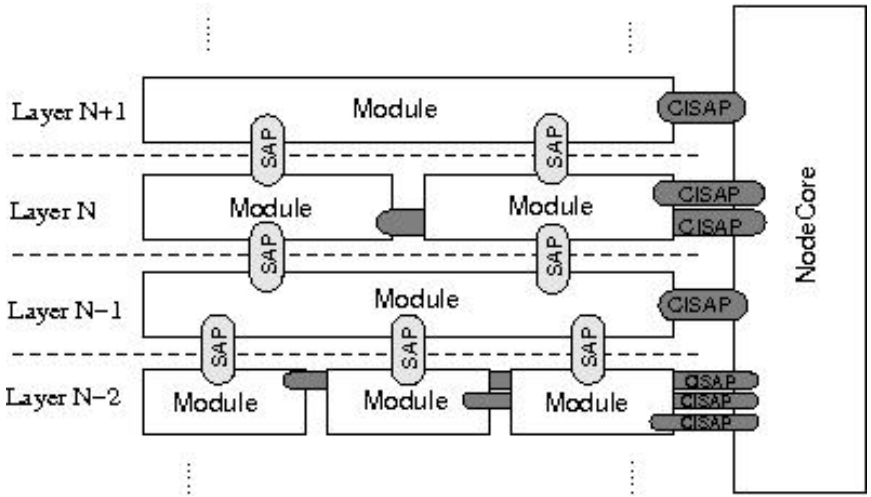
M. Chitre, L. Freitag, E. Sozer, S. Shahabudeen, M. Stojanovic, and J. Potter, "An Architecture for Underwater Networks," in OCEANS 2006 – Asia Pacific, pp. 1-5, May 2006.

# Why do we need cross-layer functionality?

- In promoting *modularity*, layering compartmentalizes information
- Protocols at one layer cannot use information available at another, and so may have sub-optimal performance
  - Example: Medium access control (data link layer function) may require knowledge of routes (network layer information) and traffic demand (application layer information) for optimal coordination of channel use
- In resource-constrained networks, the impact of layering can be severe
- Cross-layer optimization is of interest in wireless networks, but absolutely *critical* to underwater acoustic networks

# Retaining modularity, but allowing information exchange...

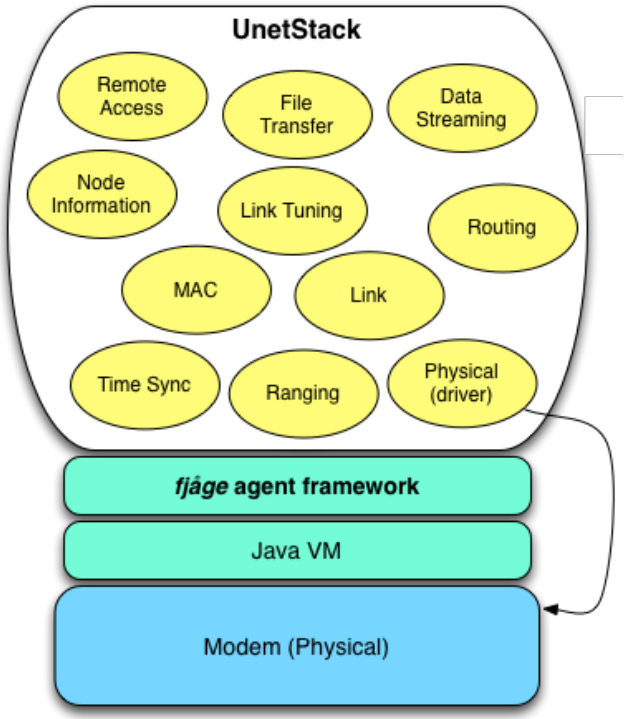
- Approach #1: Retrofit a layered protocol stack architecture with cross-layer extensions
  - Adopted by DESERT, SUNSET, etc.




NS2 MIRACLE cross-layer extension
http://telecom.dei.unipd.it/pages/read/58/

# Retaining modularity, but allowing information exchange...

- Approach #2: Adopt an agent-based architecture that naturally lends itself to information exchange between all agents ("layers")
  - Adopted by UnetStack




UnetStack
http://www.unetstack.net/

# Simulation Software Platforms

1. Ns (Network Simulator)
2. ns-miracle
3. Aqua-Sim
4. DESERT
5. SUNSET
6. Unet
7. Examples

# ns (Network Simulator)

- Began in 1989
- Variant of the REAL network simulator
- Maintained by University of Southern California, USA
- Code base in C++
- TCL scripting support
- ns-2 released in 1996

# ns-miracle

- Multi-InteRfAce Cross-Layer Extension library for the Network Simulator
- Extension for ns-2
- Engine for handling cross-layer messages
- Co-existance of multiple modules within each layer
- Maintained by University of Padova, Italy

# Aqua-Sim NG

- Simulate acoustic signal attenuation and packet collisions in underwater sensor networks
- Extends ns-3
- Maintained by University of Connecticut, USA

# DESERT

- DEsign, Simulate, Emulate and Realize Test-beds
- Underwater network protocol simulator
- Extends ns-miracle
- Maintained by University of Padova, Italy

# SUNSET

- Sapienza University Networking framework for underwater Simulation, Emulation and real-life Testing
- Simulate, emulate and test in real-life Novel (underwater) communication protocols
- Based on ns-miracle
- Maintained by University of Rome - La Sapienza

# Unet

- Agent-based network stack and simulator designed for underwater communication networks
- Code in Java and Groovy
- Groovy scripting support
- Maintained by National University of Singapore and Subnero

# Simulator Comparison

| Areas | Aqua-Sim NG | SUNSET | DESERT | Unet |
|---|---|---|---|---|
| Based On | ns-3 | ns-miracle | ns-miracle | - |
| Open Source | Yes | Yes | Yes | Partial |
| Written in | C++ | C++ | C++ | Java |
| Scripting support | TCL | TCL | TCL | Groovy/Python |
| Built-in IDE | No | No | No | Yes |
| Last Major Update | under dev | Apr 2014 | Jul 2017 | May 2018 |
| Architecture | Layered | Layered | Layered | Agent-based |
| Real-Time Scheduling | ns-3 | custom | ns-2 | custom |
| Scheduler Compensation | No[1] | Yes | No | Yes |
| Scriptable Packet Converter | No[1] | Yes | No | Yes |
| Scriptable Agents/Nodes | No[1] | No | No | Yes |

[1] Based on description in the paper, Aqua-Sim: An NS-2 Based Simulator for Underwater Sensor Networks

# Section 3

# Physical layer

# Physical layer

# Challenges

- Intersymbol interference (ISI) due to multipath (long delay spread)
- Time variability (Doppler spread)
- Wideband Doppler

# Modulation

- Incoherent modulation (e.g. FH-BFSK)
- Coherent modulation
  - Single carrier (e.g. BPSK)
  - Multi-carrier (e.g. OFDM)

# FH-BFSK

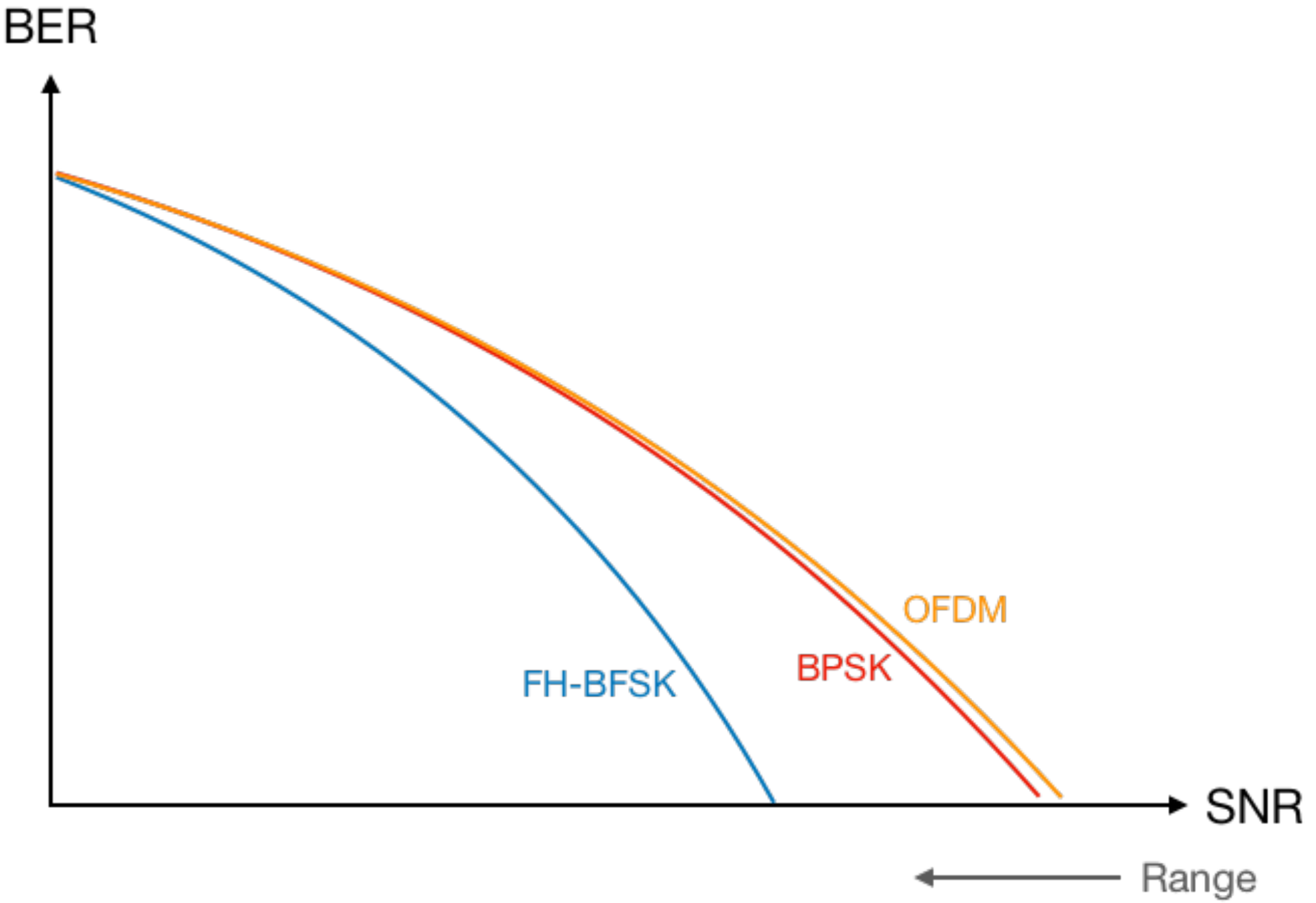
- Frequency-hopping Binary Frequency-shift Keying
- Basis of the JANUS standard
- Use one frequency to indicate a 0, another to indicate a 1
- Switch to a different pair of frequencies to avoid multipath
- Slow, but robust!

# BPSK

- Binary Phase-shift Keying
- Decision-feedback Equalizer (DFE)
- Use carrier phases to represent bits
- Feedback decisions to cancel ISI from previous symbols
- Faster, but less robust
- Sensitive to delay spread and time variability

# OFDM

- Orthogonal Frequency Division Multiplexing
- Simple to implement
- Leave a long guard period between symbols to avoid ISI
- Make up for loss in efficiency but using many carriers simultaneously
- Robust to delay spread
- Very sensitive to Doppler




For illustration only, not simulation results

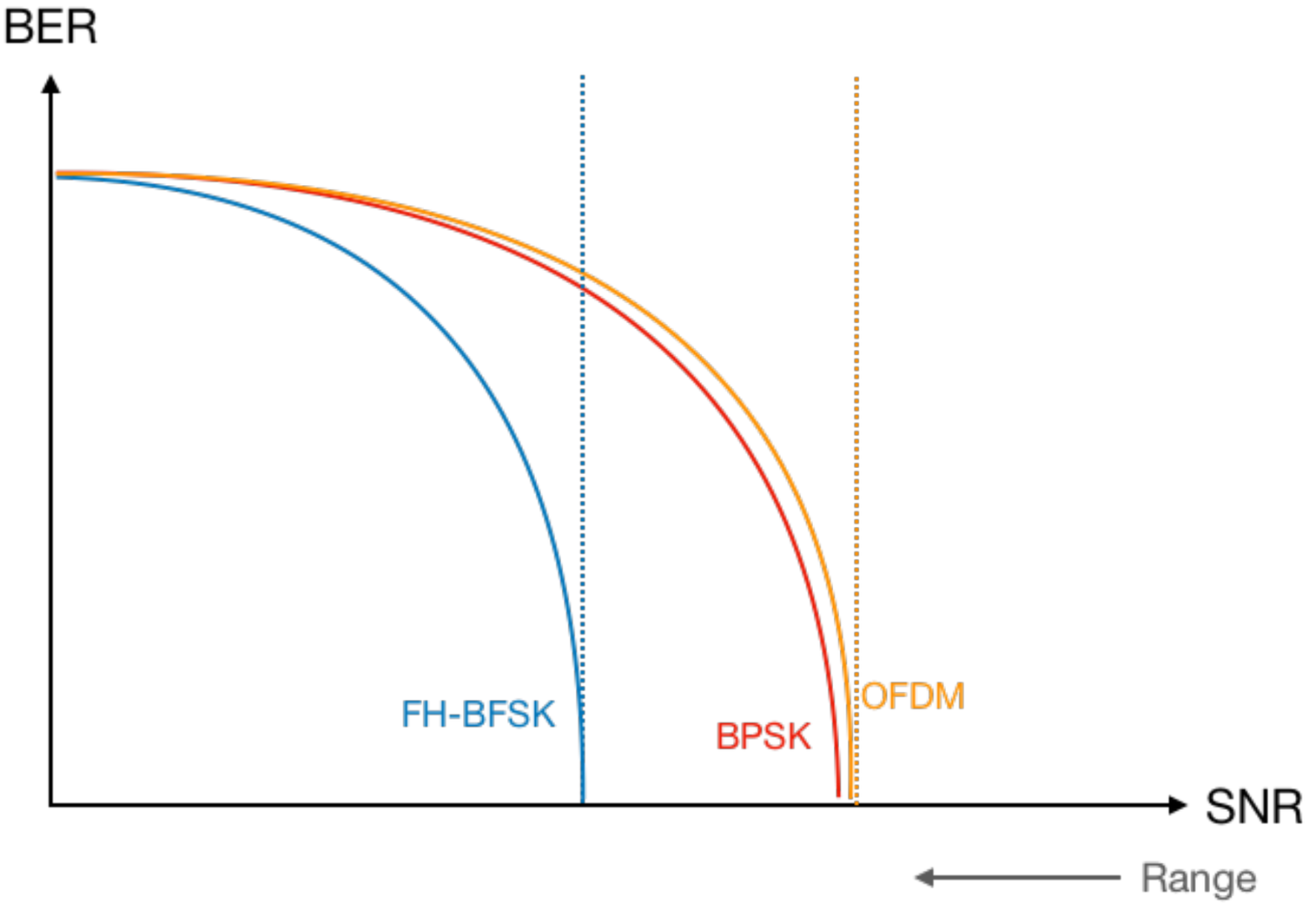
# Logical channels

No one shoe fits all!

- CONTROL: low rate, high robustness, short frames
- DATA: high rate, longer frames

# Error correction and detection

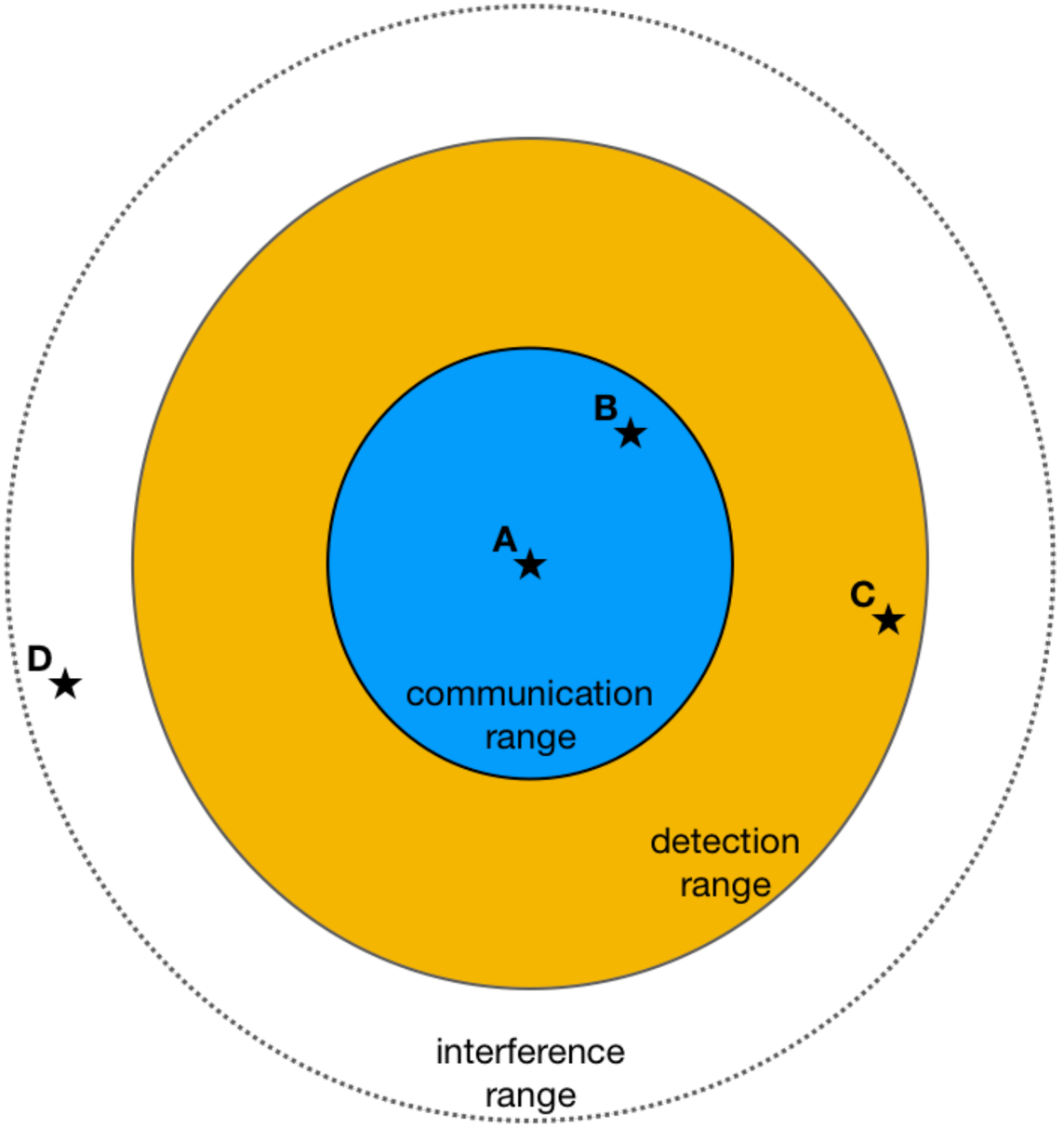
- After demodulation, some bits will be in error
- Use of forward error correction (FEC) codes for correct these (e.g. Turbo codes, LDPC, etc)
- Use of CRC to detect if all errors have been successfully corrected
- If CRC fails, drop frame




For illustration only, not simulation results

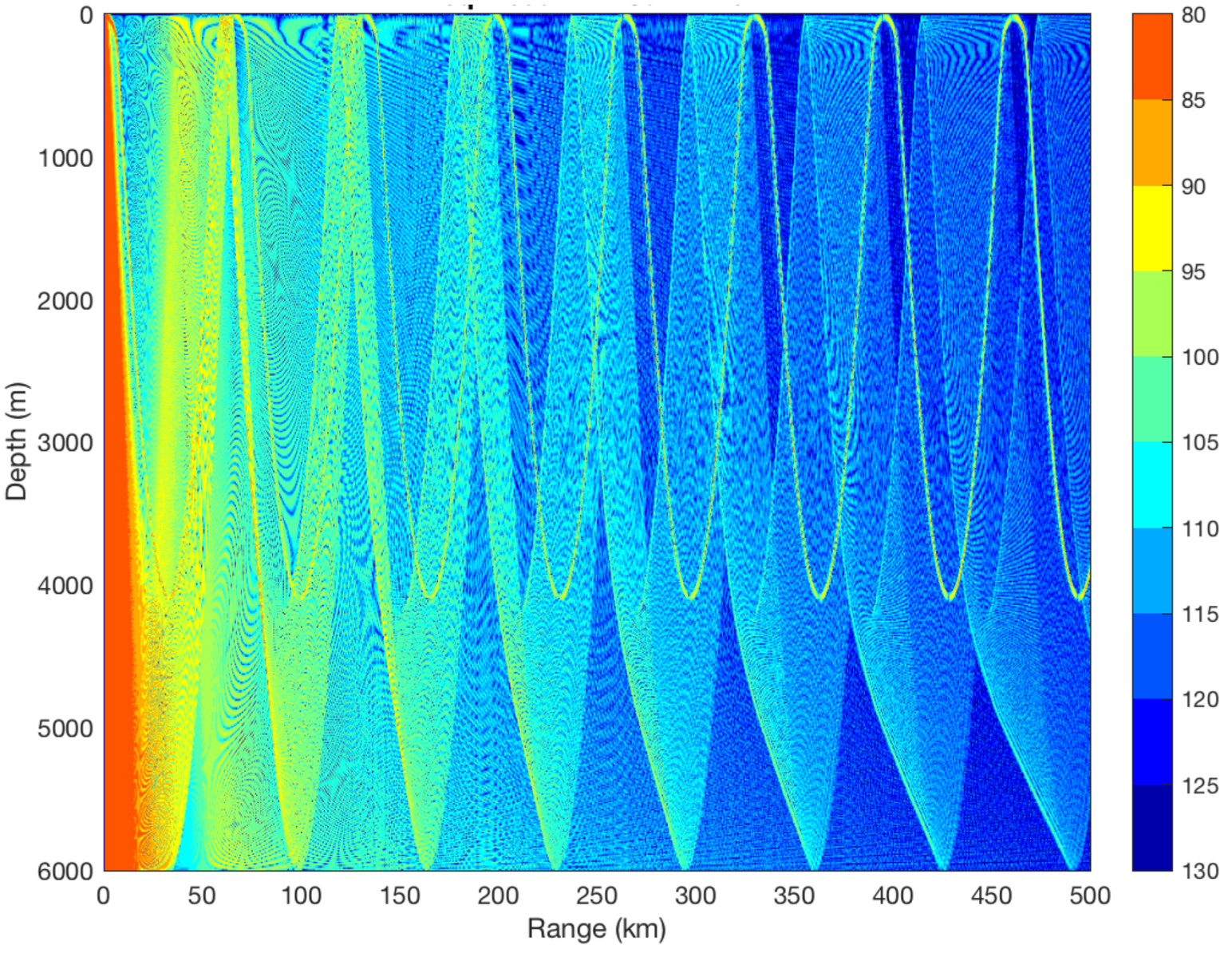
# Abstractions

- Protocol channel model
  - characterized by communication range, detection range and interference range


B
A
C
D
communication
range
detection
range
interference
range

# Abstractions

- Degrees of fidelity can be added:
  - Detection probability, decoding probability
  - Signal-to-interference-and-noise ratio (SINR)
  - Ray tracing using isovelocity channel assumption
  - Ray tracing using Bellhop to get SINR
  - Ray tracing using Bellhop for acoustic simulation


0
1000
2000
3000
4000
5000
6000
Depth (m)
0
50
100
150
200
250
300
350
400
450
500
Range (km)
80
85
90
95
100
105
110
115
120
125
130

# Abstractions

- Measured channel models
  - Measured detection, decoding probability (e.g. MISSION 2013 model, Chitre et al 2014)
  - Measured packet loss trace (network replay)
  - Measured impulse response statistics (stochastic acoustic replay, Socheleau et al 2011)
  - Measured impulse responses (acoustic replay, van Walree et al 2008, 2016)

# From simulation to deployment

- The only change between simulation and deployment should be the physical layer
  - In simulation, the physical layer abstraction is used
  - During deployments, the physical layer is provided by the modem
- UnetStack enables us to work with the same code for simulation and deployment!

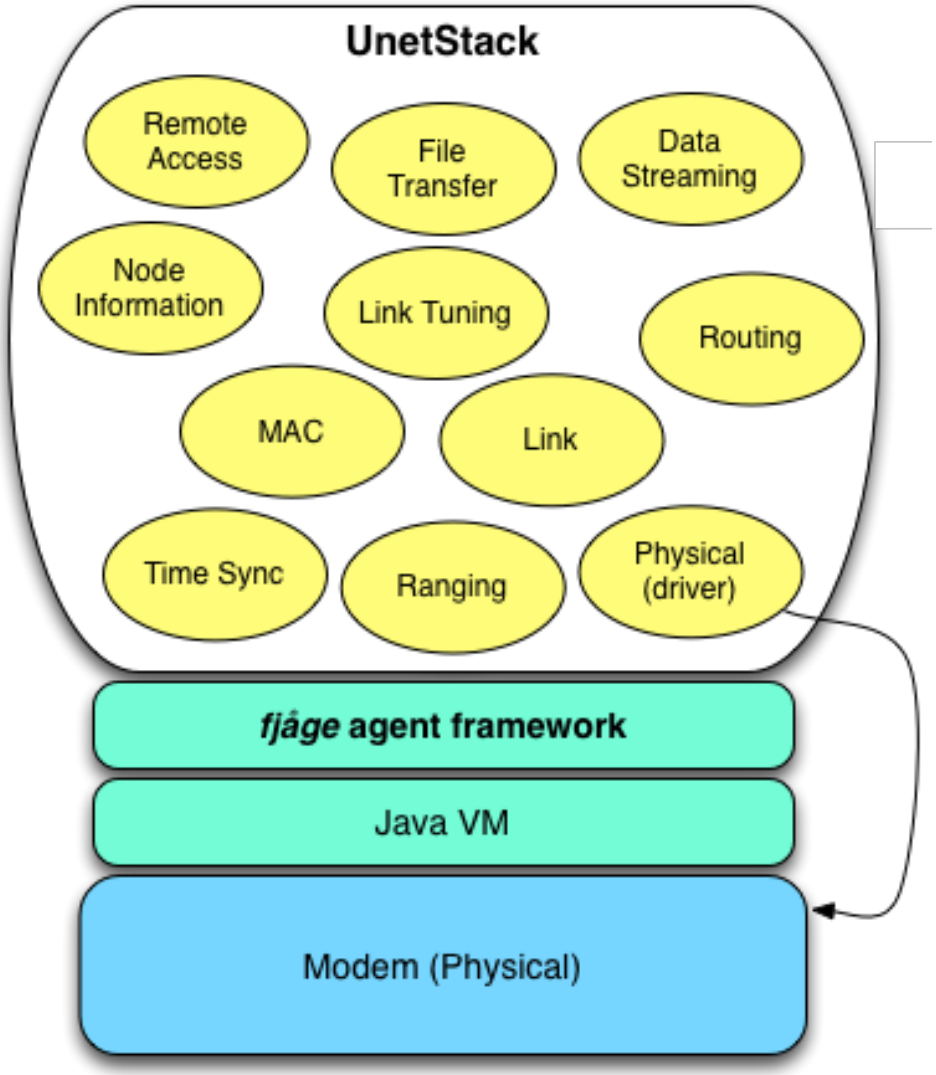
# Working with UnetStack PHY

- Sending and receiving CONTROL frames
- Sending and receiving DATA frames
- Sending and receiving arbitrary signals

Section 4

# Introduction to UnetStack

# Unet Overview



# Unet Agents

## MyAgent.groovy

```
1   import org.arl.fjage.*
2   import org.arl.unet.*
3
4   class MyAgent extends UnetAgent {
5
6     // called before other agents are ready
7     void setup() {
8       // ...
9     }
10
11    // called after other agents are ready
12    void startup() {
13      // ...
14    }
15
16  }
```

# Behaviors

- One-shot behavior – do once, as soon as possible
- Cyclic behavior – do continuously, forever
- Waker behavior – do once, after specified delay
- Back-off behavior – do once, after random delay
- Ticker behavior – do at regular intervals, forever
- Poisson behavior – do at random intervals, forever
- Message behavior – do each time a message arrives
- Finite State Machine behavior – build a FSM

# Example

## TickerBehavior

```
1   class MyAgent extends UnetAgent {
2
3     int ticks = 0
4
5     void startup() {
6       // increase ticks every one second
7       add new TickerBehavior(1000, {
8         ticks++
9       })
10    }
11
12  }
```

# Messages

```
1   // specific message sent to a named agent
2   phy = agent('phy')
3   msg = new DatagramReq(recipient: phy, data: [1,2,3,4])
4   send(msg)
5
6   // alternate syntax
7   phy = agent('phy')
8   msg = new DatagramReq(data: [1,2,3,4])
9   phy.send(msg)
10
11  // generic message sent to a topic
12  myTopic = topic('personal')
13  msg = new GenericMessage(myName: 'john', myAge: 21)
14  myTopic.send(msg)
```

# Requests and Responses

```
1   // send request to an agent, and wait for response
2   phy = agent('phy')
3   msg = new DatagramReq(to: 21, data: [1,2])
4   phy.send(msg)
5   rsp = receive()
6
7   // shorter syntax
8   phy = agent('phy')
9   rsp = phy.request(new DatagramReq(to: 21, data: [1,2]))
10
11  // even shorter syntax
12  phy = agent('phy')
13  rsp = phy << new DatagramReq(to: 21, data: [1,2])
```

# Topics and Notifications

```
1   // subscribe to notifications on a given topic
2   subscribe topic(agent('phy'))
3
4   // receive any message sent to me,
5   // or to a topic that I subscribe to,
6   // with a default timeout of 1 second
7   msg = receive()
8
9   // receive specific type of notification
10  // with a 2 second timeout
11  msg = receive(RxFrameNtf, 2000)
```

# Processing Messages

```
1   class MyAgent extends UnetAgent {
2
3     Message processRequest(Message msg) {
4       // ...
5       return null
6     }
7
8     void processMessage(Message msg) {
9       // ...
10    }
11
12  }
```

# Example

## PingDaemon.groovy

```
1   class PingDaemon extends UnetAgent {
2
3     final static int PING_PROTOCOL = Protocol.USER
4
5     void startup() {
6       def phy = agentForService Services.PHYSICAL
7       subscribe topic(phy)
8     }
9
10    void processMessage(Message msg) {
11      if (msg instanceof DatagramNtf
12          && msg.protocol == PING_PROTOCOL)
13        send new DatagramReq(recipient: msg.sender,
14          to: msg.from, protocol: Protocol.DATA)
15    }
16
17  }
```

# Services and Capabilities

```
1   class MyWonderfulMac extends UnetAgent {
2
3     def myAddress          // my node address
4     def phy                // physical layer agentID
5
6     void setup() {
7       register Services.MAC
8       addCapability MacCapability.RELIABILITY
9     }
10
11    void startup() {
12      def nodeInfo = agentForService Services.NODE_INFO
13      myAddress = nodeInfo.address
14      phy = agentForService Services.PHYSICAL
15      def rsp = request new CapabilityReq(phy,
            PhysicalCapability.TIMED_TX)
16      if (rsp.performative != Performative.CONFIRM)
17        log.warning 'TIMED_TX not supported!'
18    }
19
20  }
```

# Parameters

```
1   // define a PHY agent
2   phy = agent('phy')
3
4   // get a parameter from agent phy
5   rsp = phy << new ParameterReq().get(PhysicalParam.
        rxEnable)
6   println rsp.rxEnable
7
8   // shorter syntax
9   println phy.rxEnable
10
11  // set a parameter on agent phy
12  phy << new ParameterReq().set(PhysicalParam.rxEnable,
        false)
13
14  // shorter syntax
15  phy.rxEnable = false
```

# Example (Shell Session)

## PHY Parameters

```
> phy
[org.arl.unet.DatagramParam]
  MTU = 16
[org.arl.unet.bb.BasebandParam]
  basebandRate = 4096
  carrierFrequency = 25000
  maxPreambleID = 1
  maxSignalLength = 8192
  preambleDuration = 0.025
[org.arl.unet.phy.PhysicalParam]
  busy = false
  maxPowerLevel = 0
  minPowerLevel = -48
  propagationSpeed = 1534.4574
  refPowerLevel = 185
  rxEnable = true
  time = 5174344608
  timestampedTxDelay = 1.5
[org.arl.unet.sim.HalfDuplexModemParam]
  clockOffset = 3453.5876

>
```

# Exposing Parameters

```
1   import org.arl.unet.*
2   import com.google.gson.annotations.JsonAdapter
3
4   class MyAgent extends UnetAgent {
5
6     // parameters
7     int retryCount = 3
8     float retryTimeout = 1.0
9
10    // list of parameters
11    @JsonAdapter(JsonTypeAdapter)
12    enum Parameters implements Parameter {
13      retryCount,
14      retryTimeout
15    }
16
17    // advertise the list
18    List<Parameter> getParameterList() {
19      allOf(Parameters)
20    }
21
22  }
```

# Service Reference

- Node Information
- Address Resolution
- Datagram
- Physical
- Ranging
- Baseband
- Link
- Medium Access Control
- Routing
- Route Maintenance
- Transport
- Remote Access
- State Persistence

# Example

## Datagram Service Reference Summary

Agents offering the Datagram service support messages to transmit and receive datagrams.

- Capabilities:

  `FRAGMENTATION, RELIABILITY, PROGRESS, CANCELLATION, PRIORITY, TTL`

- Parameters:

  `MTU`

# Example

## Datagram Service Reference Summary

- Requests & responses:

  `DatagramReq` → `AGREE, REFUSE, FAILURE`

  `DatagramCancelReq` → `AGREE, REFUSE, NOT_UNDERSTOOD`

  `ParameterReq` → `ParameterRsp`

  `CapabilityReq` → `CapabilityListRsp, CONFIRM, DISCONFIRM`

- Notifications:

  `DatagramNtf, DatagramDeliveryNtf, DatagramFailureNtf, DatagramProgressNtf, ParamChangeNtf`

# Simulation DSL

## Running discrete-event simulations

```
1  //! Simulation: A very simple discrete-event simulation
2
3  simulate 1.hour, {
4    node 'A', address: 1, location: [0, 0, 0]
5    node 'B', address: 2, location: [1.km, 0, 0]
6  }
```

# Simulation DSL

## Running simulations for user interaction

```
1  //! Simulation: A real-time simulation with shell access
2
3  platform = org.arl.fjage.RealTimePlatform
4
5  simulate {
6    node 'A', address: 1, shell: true, location: [0, 0, 0]
7    node 'B', address: 2, remote: 1102, location: [1.km,
         0, 0]
8  }
```

# Simulation DSL

## Running simulations with custom agents

```
1  //! Simulation: 3-node network with ping daemons
2
3  import org.arl.fjage.*
4
5  platform = RealTimePlatform
6
7  def myStack = { container ->
8    container.add 'ping', new PingDaemon()
9  }
10
11 simulate {
12   node 'A', location: [0, 0, 0], stack: myStack, shell:
         true
13   node 'B', location: [1.km, 0, 0], stack: myStack
14   node 'C', location: [2.km, 0, 0], stack: myStack
15 }
```

# Simulation DSL

## Node mobility

```
1   n = node('AUV-4', location: [-20.m, -150.m, 0],
2            heading: 0.deg, mobility: true)
3
4   // dive to 30m before starting survey
5   n.motionModel = [
6     [duration: 5.mins, speed: 1.mps, diveRate: 0.1.mps],
7     [diveRate: 0.mps]
8   ]
9
10  // then do a lawnmower survey
11  n.motionModel += MotionModel.lawnmover(speed: 1.mps,
12    leg: 200.m, spacing: 20.m, legs: 10)
13
14  // finally, come back to the surface and stop
15  n.motionModel += [
16    [duration: 5.mins, speed: 1.mps, diveRate: -0.1.mps],
17    [diveRate: 0.mps, speed: 0.mps]
18  ]
```

# Simulation DSL

## Running simulations over a range of parameters

```
1  for (def load = 0.1; load <= 1.5; load += 0.1) {
2    simulate 1.hour, {
3      // ...
4    }
5  }
```

# Modem & Channel Settings

```
1   // modem model settings
2   modem.model = HalfDuplexModem
3   modem.dataRate = [800.bps, 2400.bps]
4   modem.frameLength = [16.bytes, 64.bytes]
5   modem.powerLevel = [0.dB, -10.dB]
6   modem.preambleDuration = 5.ms
7   modem.txDelay = 0
8
9   // channel model settings
10  channel.model = BasicAcousticChannel
11  channel.carrierFrequency = 25.kHz
12  channel.bandwidth = 4096.Hz
13  channel.temperature = 25.C
14  channel.salinity = 35.ppt
15  channel.noiseLevel = 60.dB
16  channel.waterDepth = 20.m
```

# Channel Models

- Protocol channel model
- Basic acoustic channel model

  (based on Urick and BPSK with Rayleigh/Rician fading)
- MISSION 2012a and 2013a channel models
- Custom channel models

Section 5

# Medium Access Control (MAC)

# Outline

- Background and MAC basic concepts
- Protocol models and choices
- An intuitive comparison of basic protocol models
- Mathematical analysis basics
- Adaptive MAC protocol suite
- Conclusion

# Background

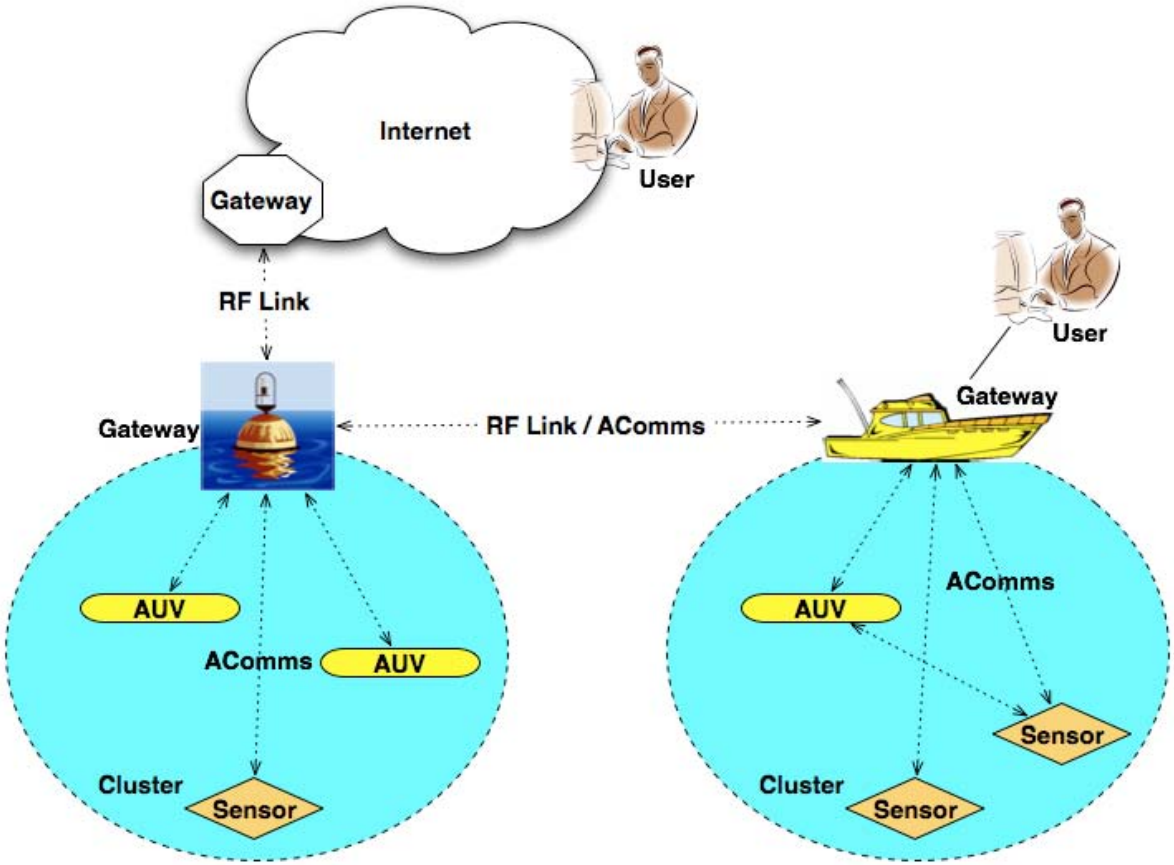
- Improve physical layer signal processing performance
- At Layer 2, MAC or Medium Access Control
  - Ensure efficient co-ordination between nodes to share limited bandwidth
  - informs who can transmit and when
- At every layer above MAC also, efficiency is critical

# MAC

- In UnetStack, agents are the users of the channel
- Ask MAC when and for how long to use the channel
- May be able to book timeslots in advance
- Call Physical services to transmit once timeslot is granted

# Underwater Acoustic Networking (UAN)

- characterized by large propagation delay, extremely low data-bandwidth and high packet errors
- requires unique, tailored solutions

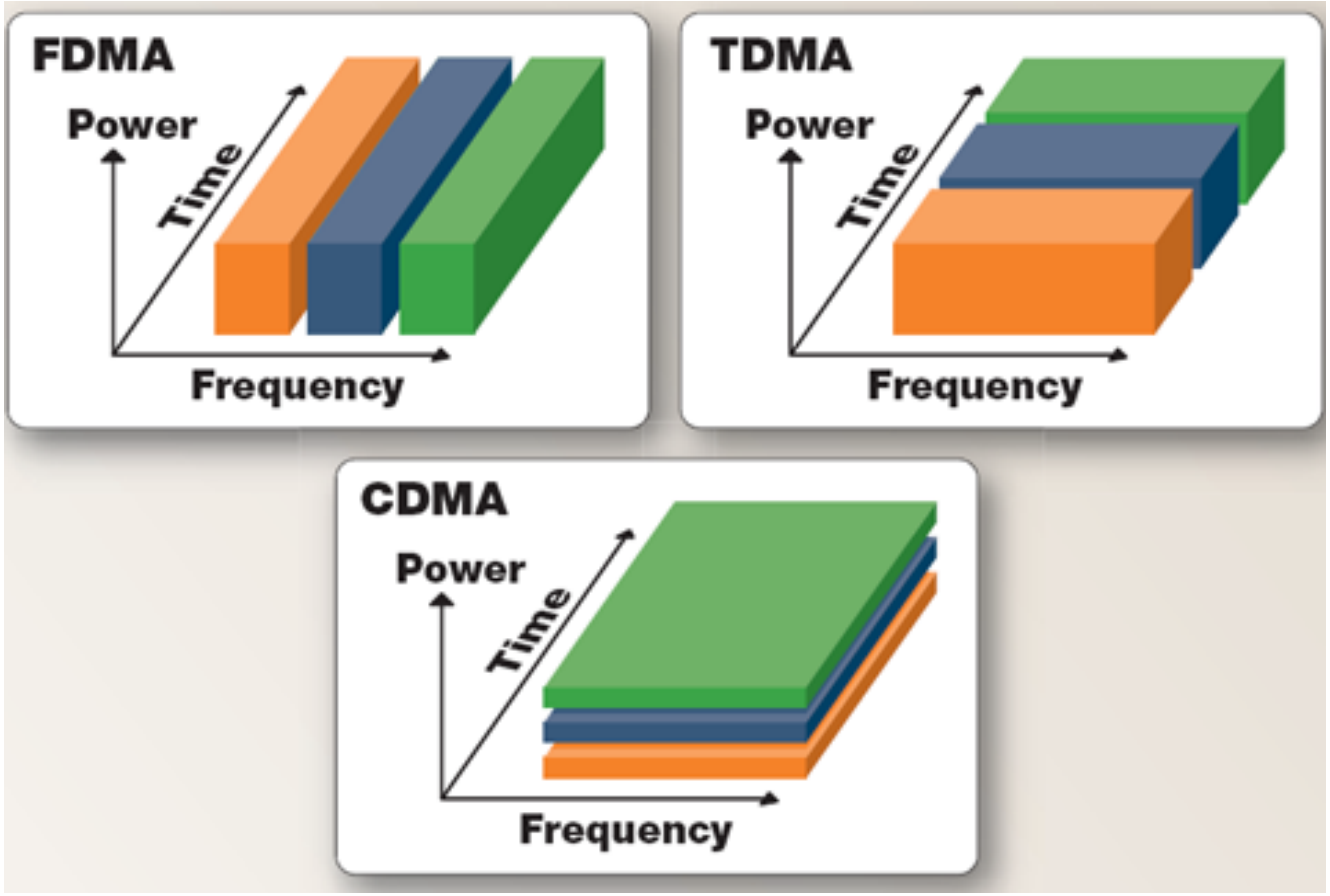


# Important Considerations

- Dynamic channelization, allocation
- Contention based vs. contention-free
- Ad hoc capability: arrivals and departures
- Spatial re-use
- Topology: distributed vs. centralized
- Dependence on time synchronization
- Scalability
- Robustness
- Fairness

# MAC Protocol Choices

Channelization: TDMA, FDMA, CDMA




[1]

[1] Image courtesy: http://wcdma3g.blogspot.in/2008/07/conclusion.html

# MAC Protocol Choices

Channelization: TDMA, FDMA, CDMA

- Equivalence in ideal case is known [BOOKCH]
- Time x Bandwidth x Code space
- CDMA in UANs affected by latency induced non-orthogonality, near-far problem

# MAC Protocol Choices

- The most significant difference!
- For equivalent capacity, CDMA and FDMA need simultaneous reception in all codes or bands while your own transmission is in progress
- i.e., modems need to be full duplex and multi-channel capable
- Such modems are not common

# MAC Protocol Choices

- TDMA is possibly the best choice for UANs
- Easy to implement over any Physical Layer
- A significant majority of proposed MAC protocols for UANs is based on time division

# Time Domain MAC Protocols

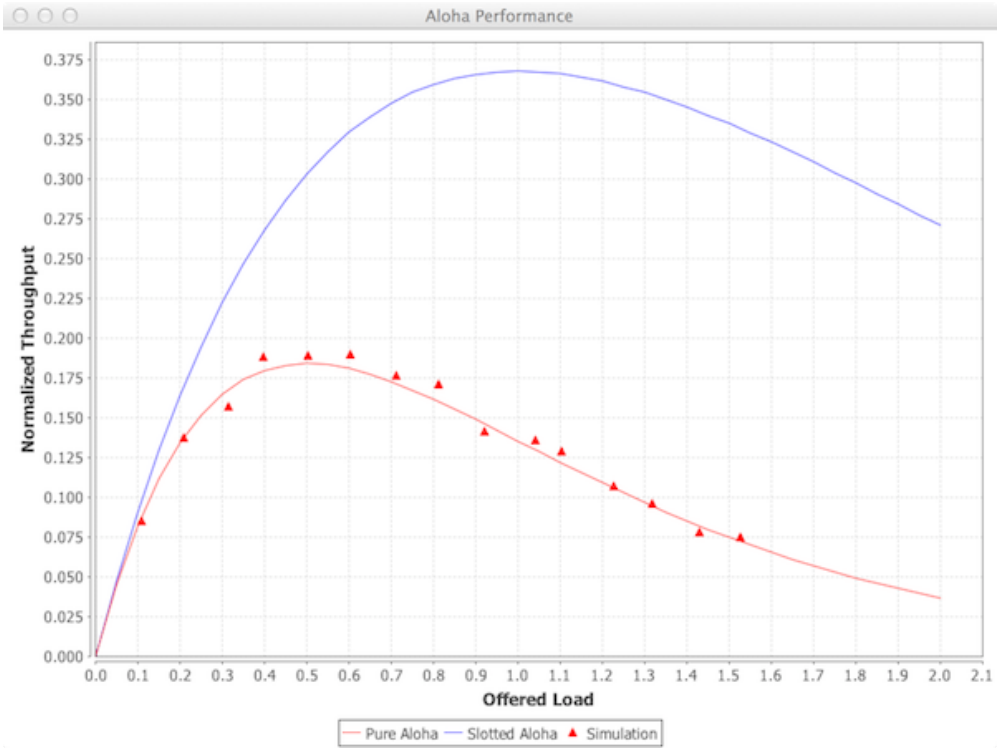
| | |
|---|---|
| TDMA | Time Domain Multiple Access |
| ALOHA | |
| DATA-ACK | |
| D-TDMA | Dynamic TDMA |
| MACA | Medium Access with Collision Avoidance |
| FAMA | Floor Acquisition Multiple Access |
| DACAP | Distance Aware Collision Avoidance Protocol |
| T-LOHI | Tone LOHI |
| MACA-EA | MACA with Early ACK |
| MACA-C | MACA-Centralized |
| MAC-AMM | Adaptive Multi-mode MAC |

# TDMA

- Easiest when there is perfect time synchronization
- Contention-free
- In radio wireless, key component of many protocols such as GSM
- Some disadvantages or challenges in UANs
  - Scalability
  - Robustness to time synchronization
  - Dynamic channelization and allocation

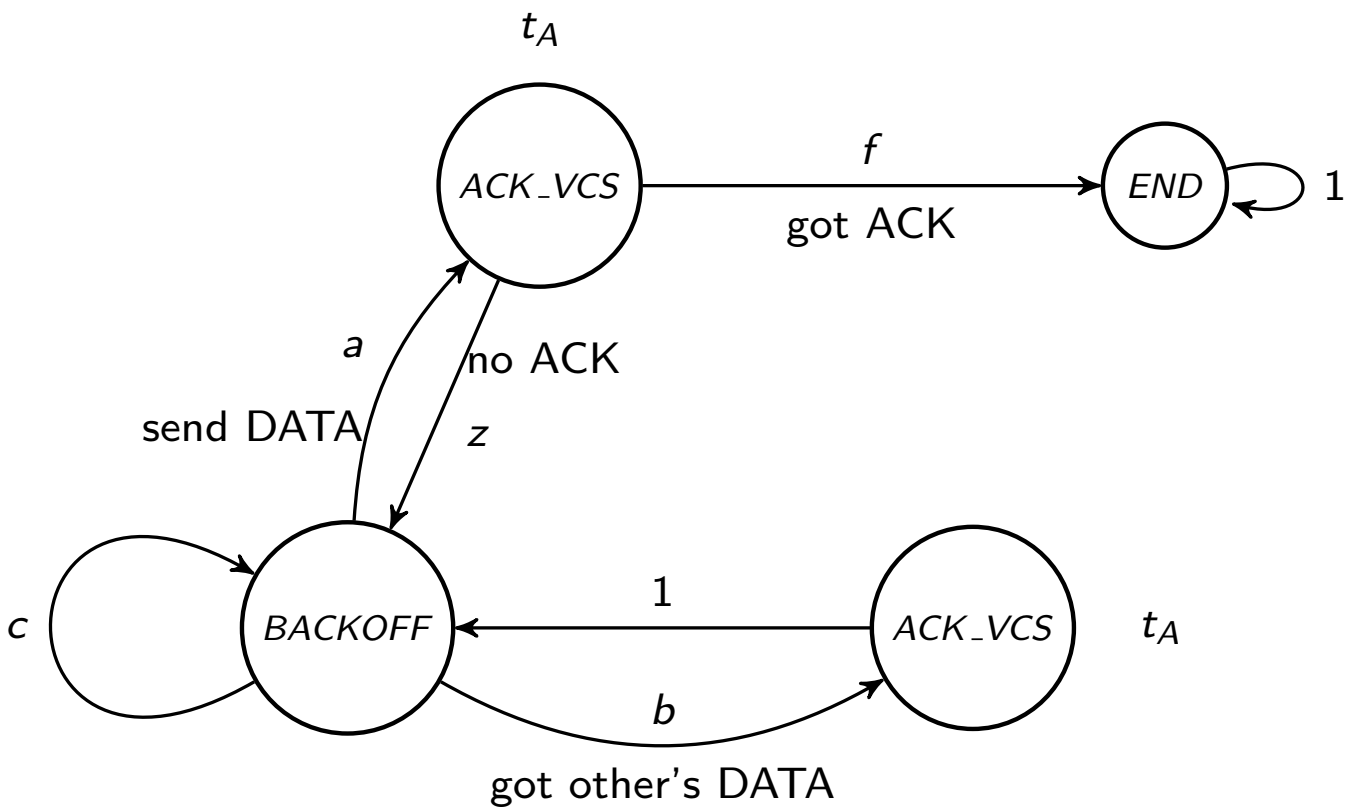
# ALOHA

- Random access, practically no MAC!
- Contention-based
- Nodes assumed to have sporadic data arrivals and transmit immediately
- Nodes do not transmit when their own transmission is in progress
- Disadvantage: low throughput, maximum 18



# DATA-ACK

- DATA-ACK with queueing and reliable communication [MAC-AMM]
- Markov chain used for mathematical analysis

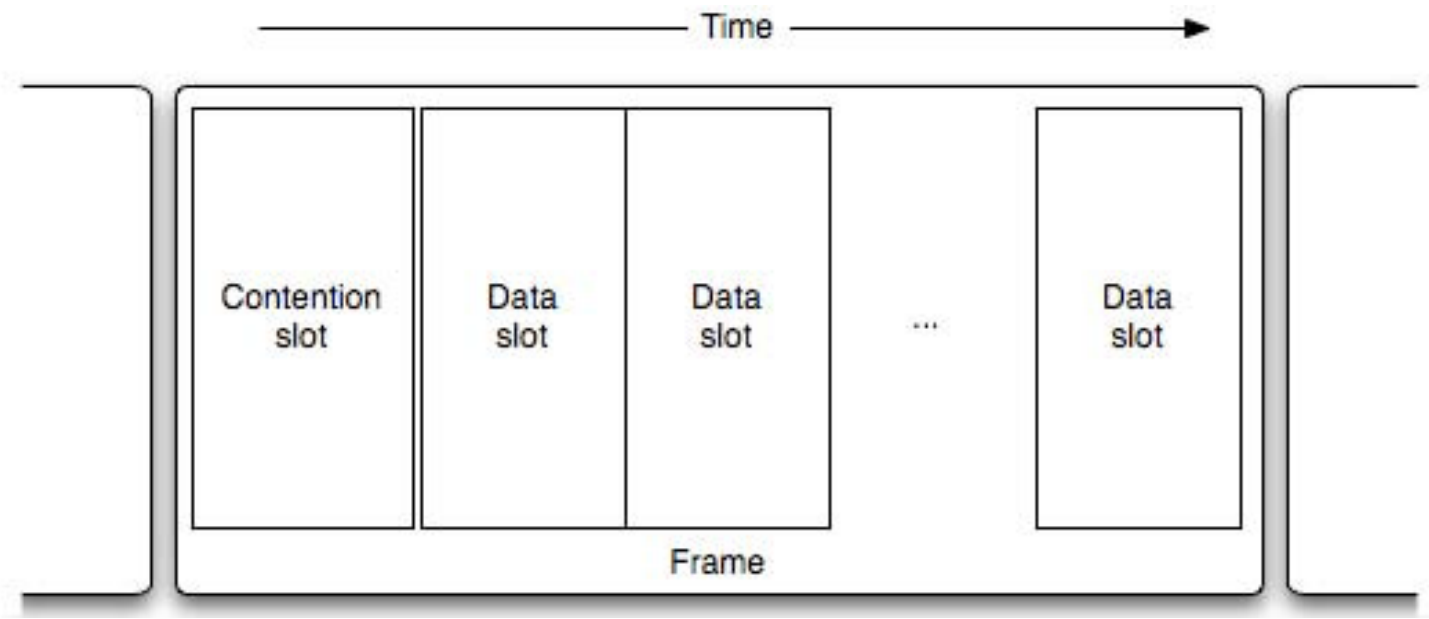



Virtual Carrier Sense (VCS)

Analysis terminating *END* state, in actual usage loops back to *BACKOFF state*

# Dynamic TDMA

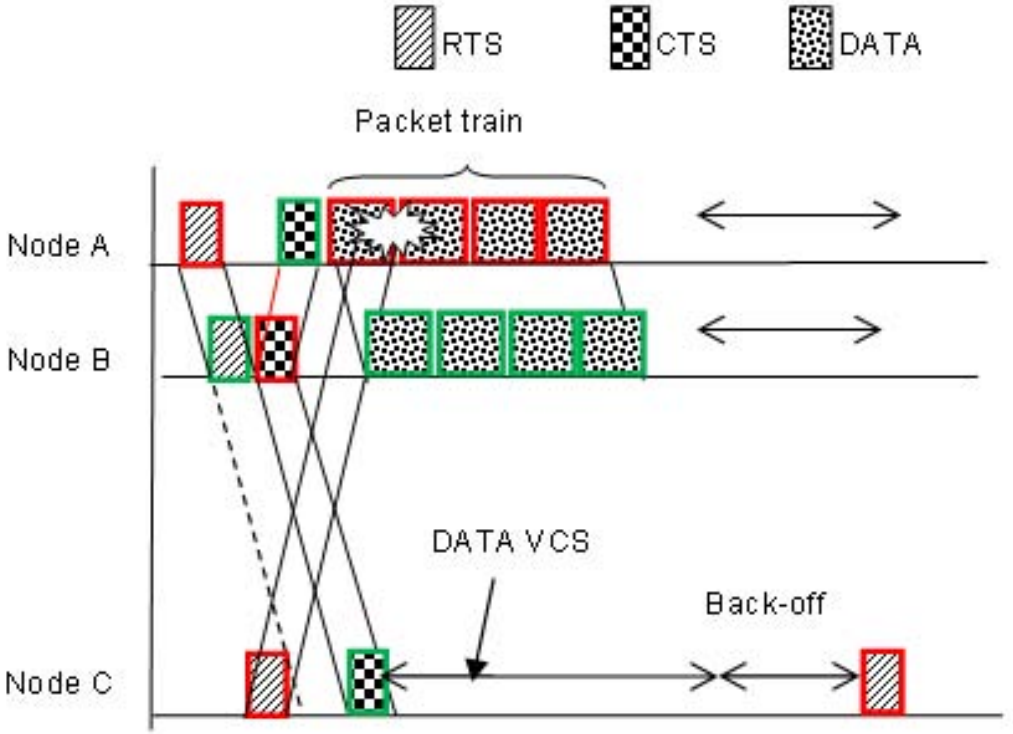
- Dynamic allocation
- Random access during the contention slots
- Contention slot takes up data-bandwidth and reduces efficiency
- Cost of ad hoc capability, dynamic allocation



# MACA

- RTS-CTS-DATA proposed by Karn in 1990 [MACA]
- Can be viewed as a logical extension of D-TDMA
- Bhargavan et al in '94 added ACK and adaptive back-off [MACAW]
- Used since 90s in terrestrial wireless and for more than a decade in UANs, e.g.[SEAWEB]
- The fundamental basis for many UAN MAC protocols proposed in the last decade

# MACA




Strictly speaking, backoff as shown is introduced in MACAW

# MACA

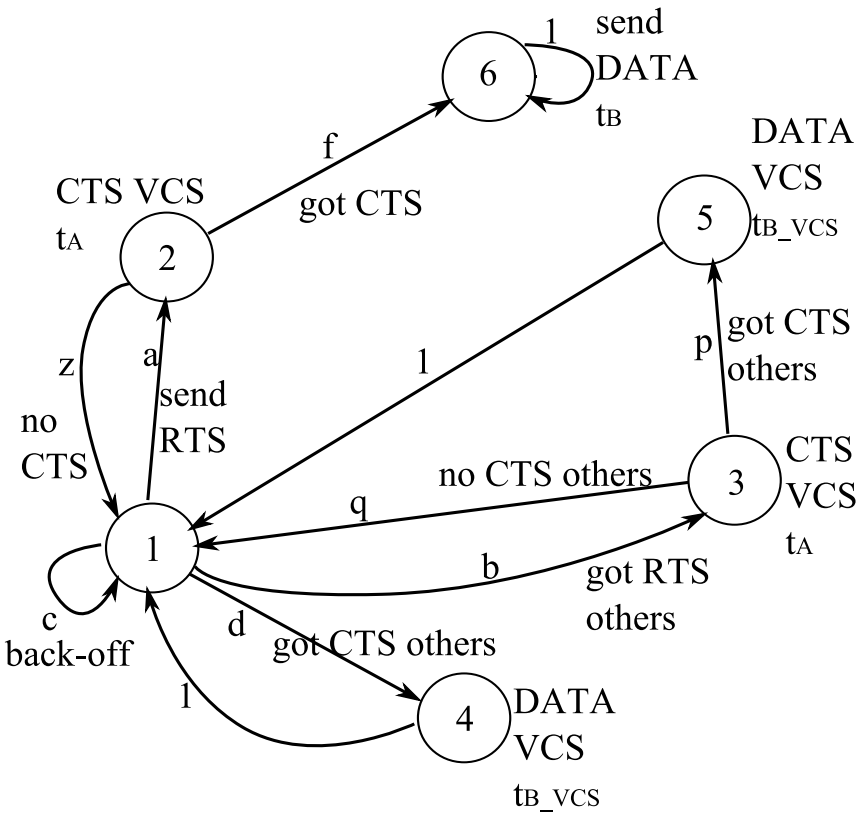
- Eliminates clock synchronization required by D-TDMA
- Dynamic channelization and allocation, automatic time domain channel-reuse
- Good for UANs with contiguous multi-hop nodes
- Note: Loss of efficiency due to latency and collisions is a fundamental limitation arising from the requirement of dynamic channelization and allocation

# MACA-EA

- A proven MAC protocol for UANs [MACA-EA]
- Mathematically analyzed, simulated and tested in sea-trials
- Based on RTS-CTS-DATA_BATCH-ACK
- Has many additional features
  - Virtual Carrier Sense (VCS) using all overheard packets including batched DATA
  - DATA packets' header shows how many packets remain in the batch. Helps nodes that missed RTS/CTS to regain VCS with a probability close to 1 after a few DATA packets are sent

# MACA-EA Markov Chain Model

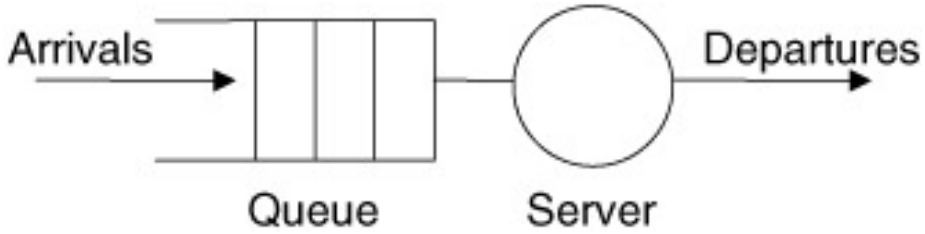
- Transition probabilities along edges ($a$, $f$, $p$ etc)
- Time in a state shown ($t_A$, $t_B$ etc)




Analysis terminating state 6, in actual usage loops back to back-off state 1

# MAC - Queueing Analysis

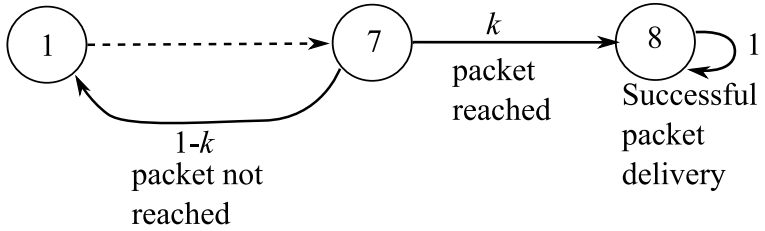
- Independence of arrivals and service




- Key factors: time duration of a packet $L$, probability that a packet is detected and decoded correctly $k$, batch size $B$, response time-out $t_A$ (related to propagation delay)
- Metrics
  - Mean batch service time and packet service time
  - Throughput (for reliable transmission)
  - Waiting time
  - Trade-off

# Mean batch and packet service times $s_b$, $s_p$

- $s_b$ - the average delay from the time a batch is intended for transmission (RTS contention starts in MACA or similar protocol) until it is successfully transmitted, i.e., until the first ACK is received for the batch
- $s_p$ - expected delay from the time a packet is intended for transmission (RTS contention starts in MACA or similar) until it is successfully delivered, i.e., until the ACK (with retries) shows successful reception of the specific packet




$$s_p = \frac{1}{k} s_b \tag{1}$$

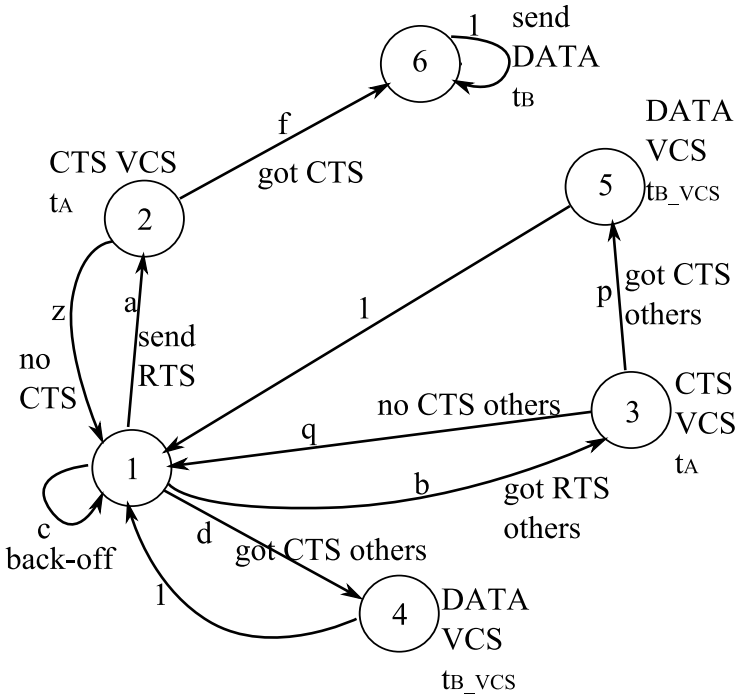
# Normalized Throughput $T$

- Defined as the number of packets successfully transferred per unit time normalized by the system capacity, (1 pkt in $L$). $B$ packets are sent as a batch in time $s_b$, and of these, only $k$ succeed due to decoding and detection losses.

$$T = \frac{kB/s_b}{(1/L)} \tag{2}$$

# The Transition Matrix

- MAC protocol state machines can be analyzed as Markov Chains [MACA-EA, MAC-AMM]




$$\mathbf{M} = \begin{bmatrix} c & a & b & d & 0 & 0 \\ z & 0 & 0 & 0 & 0 & f \\ q & 0 & 0 & 0 & p & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \quad (3)$$

# MACA-EA: Normalized Throughput $T$

$$T = \frac{kB/s_b}{(1/L)}$$

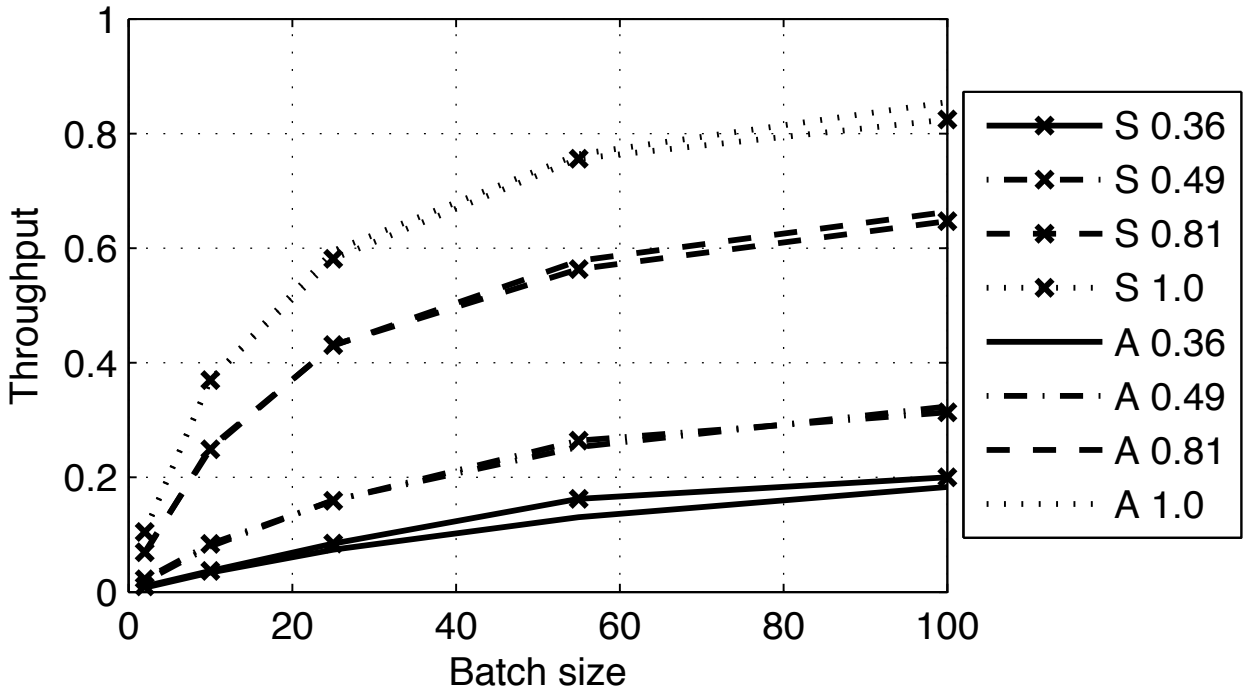
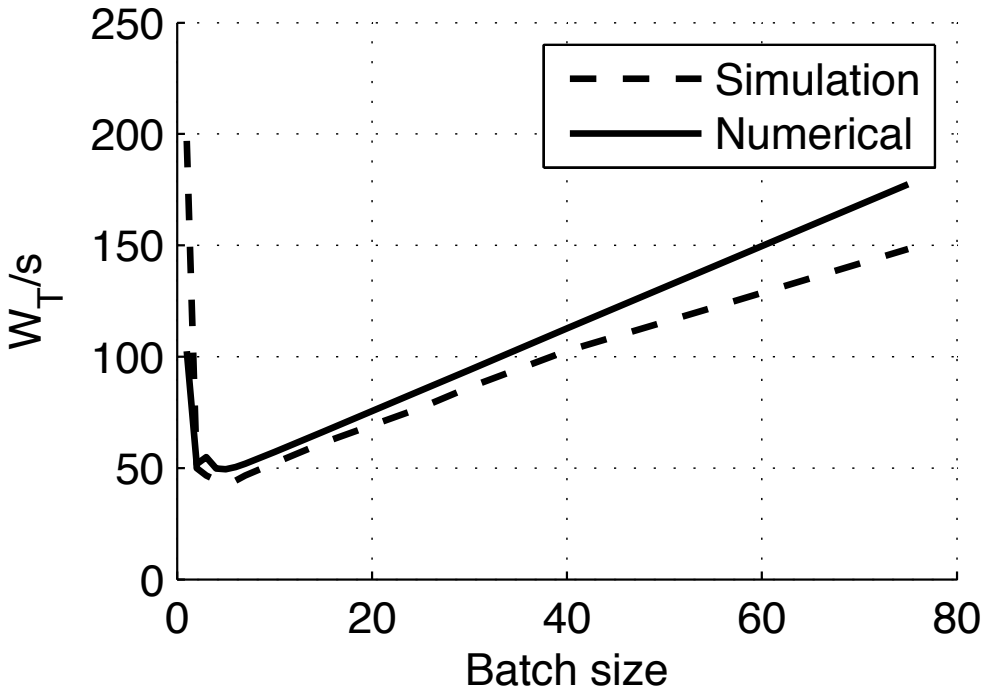



Figure: Throughput vs. batch size for $k = 0.36$, 0.49, 0.81 and 1.0, simulations (S), analysis (A). Parameters: $L = 0.5s, N = 3, D = 0.5s, W = 4, i = 3$.

# MACA-EA: Waiting Time Analysis

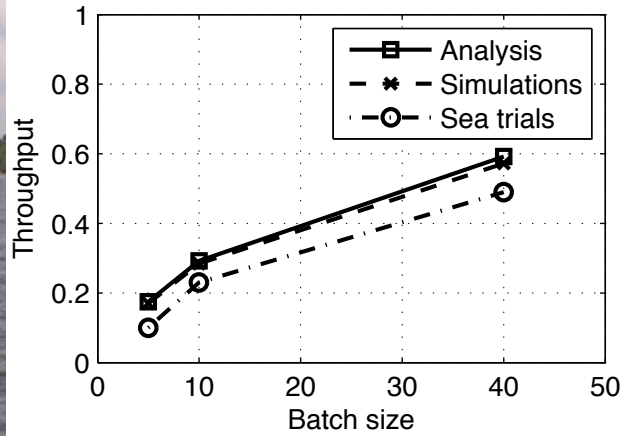
- The total waiting time $W_T = W_Q + s_p$ ($W_Q$ is waiting time in queue alone)



# MAC - Queueing Analysis Key Take Aways

- For random access protocols such as ALOHA there is a throughput maximization behaviour as widely known
- For batch mode protocols, throughput saturates as batch size increases
- An optimum batch size minimises waiting time
- There exists a maximum arrival rate for stable performance

# Validation - MACA-EA

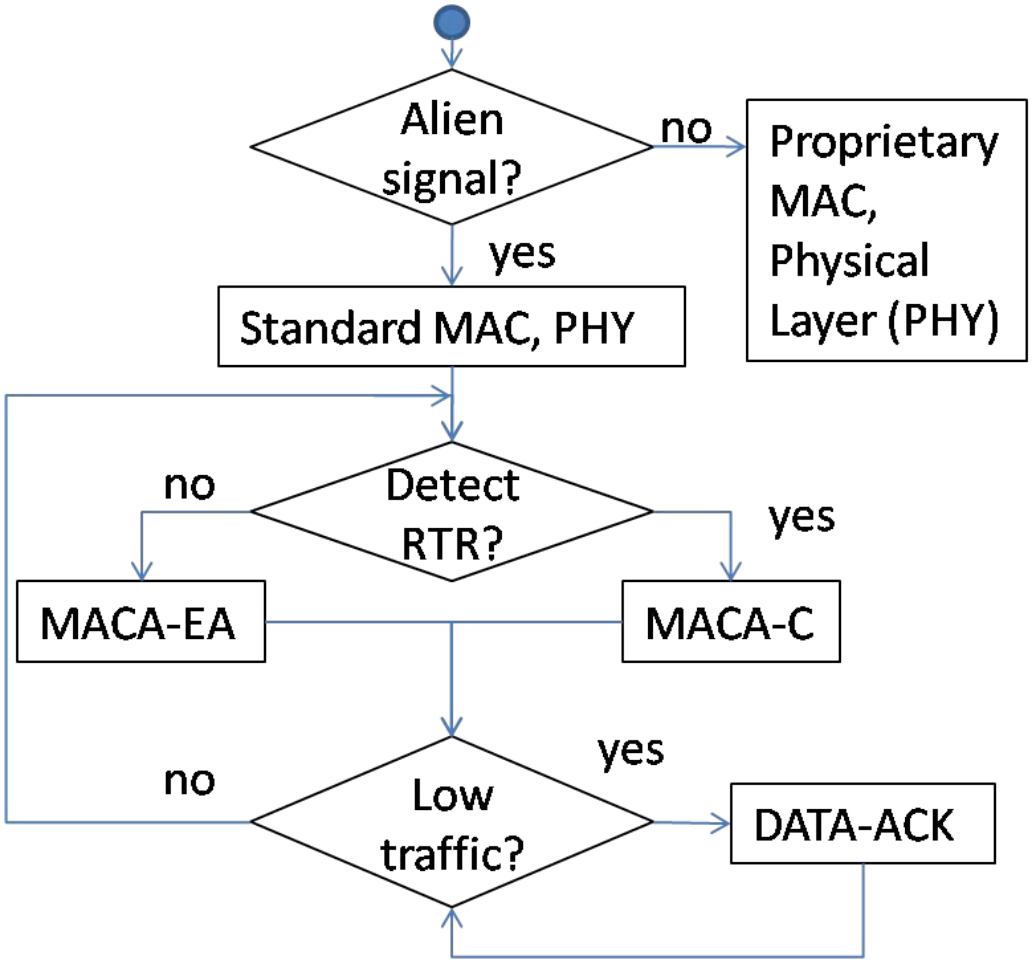



# Adaptive multi-mode MAC protocol MAC-AMM

- An adaptive suite of protocols [MAC-AMM]
- DATA-ACK mode with queuing and reliable communication
- State-dependent DATA-ACK for low traffic intensity
- Centralized MACA-C mode
- MACA-EA
- Uses a transition traffic intensity between MACA-EA and DATA-ACK modes for minimizing waiting time
- The 802.11 DCF is akin to MACA-EA and 802.11 PCF is akin to MACA-C and 802.11 has a mode similar to the DATA-ACK

# MAC-AMM

- Adaptation



# Other areas to be explored

- Neighbourhood node and traffic estimation
- Link tuning and power control
- Fairness and starvation
- Standardization of UAN MAC
- Seamless integration with Internet

# Conclusion

- Time domain protocols are often the best choice compared to CDMA or FDMA for UAN
- All proposals use one of the basic protocol structures
  - TDMA, Random Access, MACA
- MACA-based protocols well suited for UANs
- Heterogeneous UAN and adaptation to changing environment and requirements - MAC-AMM
- Protocol performance must be validated through sea-trials
- Markov chain analysis for theoretical understanding of behaviour
- Simulation results must be compared either on same or compatible platforms
- Performance comparisons must be made on equal terms in terms of metrics, environmental conditions etc

# References


| | |
|---|---|
| MACA | Phil Karn, "MACA - A New Channel Access Method for Packet Radio," Proceedings of the 9th ARRL Computer Networking Conference, Canada, 1990 |
| MACAW | Bhargavan, V., Demers, A., Shenker, S. and Zhang, L., "MACAW: A media access protocol for wireless LANs," Proceedings of ACM SIGCOMM 94, pp. 212-25, ACM, 1994 |
| SEAWEB | J. Rice et al., "Evolution of Seaweb underwater acoustic networking," OCEANS 2000 MTS/IEEE Conference and Exhibition, Providence, RI, 2000, pp. 2007-2017 vol.3 |
| S-FAMA | M. Molins and M. Stojanovic, "Slotted FAMA: a MAC protocol for underwater acoustic networks," OCEANS 2006 - Asia Pacific, Singapore, 2006, pp. 1-7 |
| DACAP | B.Peleato and M.Stojanovic, "Distance Aware Collision Avoidance Protocol for Ad-Hoc Underwater Acoustic Sensor Networks," IEEE Communication Letters, pp.1025-1027, December 2007 |
| TLOHI | A. A. Syed, W. Ye and J. Heidemann, "T-Lohi: A New Class of MAC Protocols for Underwater Acoustic Sensor Networks," INFOCOM 2008. The 27th Conference on Computer Communications. IEEE, Phoenix, AZ, 2008 |
| MACA-EA | S. Shahabudeen, M. Motani and M. Chitre, "Analysis of a High Performance MAC Protocol for Underwater Acoustic Networks," IEEE J. Oceanic Eng., vol. 39, no. 1, pp. 74-89, 2014 |
| BOOKCH | S. Shahabudeen, M. Chitre, and M. Motani, Underwater Acoustic Sensor Networks. CRC Press, 2010, ch. "Dynamic TDMA and MACA based Protocols for Distributed Topology Underwater Acoustic Networks" |
| MAC-AMM | S. Shahabudeen, M. Chitre, and M. Motani, "Adaptive multi-mode medium access control for underwater acoustic networks," IEEE J. Oceanic Eng., vol. 39, no. 3, pp. 500-514, 2014 |

# Objective

Develop two simple MAC agents using UnetStack

- A very simple MAC
- Handshake MAC - more complex protocols

---

The MAC agents are intentionally kept simple and not optimized for performance.

`http://www.unetstack.net/doc/html/mac.html` for runnable code samples

# MAC Channel Reservation

- MAC in UnetStack does channel reservation in a broader sense
- It does not directly transmit data
- It just lets the caller agent know when to START and STOP channel usage (via Physical) in reply to a ReservationReq
- This is done via ReservationStatusNtf notification and the status therein of ReservationStatus.START and ReservationStatus.END
- The caller agent can use the allocated time for data transmission or other actions such as ranging, baseband transmissions etc

# Data Transmission

- Regular data agents (RD) - shall normally use Link agent for data transmissions
- Link agent deals with MAC on their behalf
- Other agents (OA) - may directly call MAC agent to perform actions such as ranging, baseband transmissions or any custom data transmission protocols not offered by the stock data agents in UnetStack

# Send a reservation request

```
1  def mac = agentForService Services.MAC
2  if (mac) {
3    def req = new ReservationReq(recipient: mac, to:
         destination, duration: duration)
4    def rsp = request req
5    if(rsp && rsp.performative== Performative.AGREE){
6
7      // wait for a channel reservation notification
8      def ntf= receive(ReservationStatusNtf, timeout)
9      if (ntf && ntf.requestID == req.msgID && ntf.status ==
           ReservationStatus.START) {
10       // transmit data for requested duration
11       //  :
12     }
13   }
14 }
```

# A simple MAC agent [2]

- Accedes to every reservation request as soon as it is made

```
1  class MySimplestMac extends UnetAgent {
2
3    void setup() {
4      register Services.MAC
5    }
6
7    Message processRequest(Message msg) {
8      //body shown next slide
9    }
10 }
```

[2] samples/mac/MySimplestMac.groovy

# A simple MAC agent

```
1  Message processRequest(Message msg) {
2    if (msg instanceof ReservationReq) {
3      if (msg.duration <= 0)
4         return new Message(msg, Performative.REFUSE)
5
6      ReservationStatusNtf ntf1 = new ReservationStatusNtf(
7        recipient: msg.sender, requestID: msg.msgID,
8        to: msg.to, status: ReservationStatus.START)
9
10     ReservationStatusNtf ntf2 = new ReservationStatusNtf(
11       recipient: msg.sender, requestID: msg.msgID,
12       to: msg.to,  status: ReservationStatus.END)
13
14     send ntf1
15
16     add new WakerBehavior(Math.round(1000*msg.duration),
17     { send ntf2 })
18
19     return new ReservationRsp(msg)//default return AGREE
20   }
21   return null
22 }
```

# Simulation and Performance Estimation

Real-time simulation

- Test code through manual interaction via the shell
- sample with random node locations and link performance

# Real time: 3-node random network[3]

```
1  platform = RealTimePlatform
2  channel = [ model: BasicAcousticChannel ]
3  simulate {
4    // define network stack
5    def myStack = { container ->
6      container.add 'mac', new MySimplestMac()
7    }
8    // define simulation nodes
9    node "1", location: [rnd(-500.m, 500.m),
10         rnd(-500.m, 500.m), rnd(-20.m, 0)],
11         stack: myStack, shell: true
12   node "2", location: [rnd(-500.m, 500.m),
13         rnd(-500.m, 500.m), rnd(-20.m, 0)],
14         stack: myStack
15   ...
16 }
```

[3] samples/mac/mac-test-rt.groovy

# Using a console shell to run

```
> mac << new ReservationReq(duration: 1.second, to: 2)
AGREE: ReservationRsp
ReservationStatusNtf:INFORM [requestID:xxx to:2
     status:START payload:(0 bytes)]
ReservationStatusNtf:INFORM [requestID:xxx to:2
     status:END payload:(0 bytes)]
>
```

# Load Generation [4]

```groovy
class LoadGenerator extends UnetAgent {

  private List<Integer> destNodes
  private float load
  private AgentID mac, phy

  LoadGenerator(List<Integer> destNodes, float load){
    this.destNodes = destNodes
    this.load = load
  }

  void startup() {
  }

  void processMessage(Message msg) {

  }
}
```

[4] samples/mac/LoadGenerator.groovy

# Load Generation...

```
1  void startup() {
2    phy = agentForService Services.PHYSICAL
3    mac = agentForService Services.MAC
4    float dataPktDuration = get(phy, Physical.DATA,
5         PhysicalChannelParam.frameDuration)
6    float rate = load/dataPktDuration
7       // compute average packet arrival rate
8    add new PoissonBehavior(1000/rate, {
9     // create Poisson arrivals at given rate
10    mac << new ReservationReq(to:rnditem(destNodes),
11         duration: dataPktDuration)
12   })
13 }
```

# Load Generation ...

```
1 void processMessage(Message msg) {
2   if (msg instanceof ReservationStatusNtf &&
3       msg.status == ReservationStatus.START) {
4     phy << new ClearReq()
5       // stop any ongoing transmission or reception
6     phy << new TxFrameReq(to: msg.to, type: Physical.DATA)
7       // start a new transmission
8   }
9 }
```

# Performance Test [5]

```
1  def nodes = 1..10
2  for (int i: 1..10) {
3    float load = i/10.0
4    float loadPerNode = load/nodes.size()
5    simulate 1.hour, {
6      for (int me: nodes) {
7        node "$me", location: [rnd(-500.m, 500.m), rnd(-500.
             m, 500.m), rnd(-20.m, 0)], stack: {
8          container.add 'mac', new MySimplestMac()
9          container.add 'load', new LoadGenerator(nodes-me,
               loadPerNode)   // generate load to all nodes
               except me
10       }
11     }
12   }
13   println sprintf('%6d\t\t%6d\t\t%7.3f\t\t%7.3f',
14     trace.txCount, trace.rxCount, trace.offeredLoad, trace
           .throughput)
15 }
```

[5] samples/mac/mac-test-perf.groovy

# Performance Test Results

```
MySimplestMac simulation
========================

TX Count   RX Count   Offered Load   Throughput
--------   --------   ------------   ----------
   604        466        0.110          0.085
  1118        671        0.203          0.122
  1707        746        0.310          0.135
  2231        803        0.407          0.146
  2755        779        0.502          0.141
  3258        777        0.594          0.141
  3757        732        0.685          0.133
  4457        648        0.815          0.118
  4976        575        0.910          0.104
  5474        561        1.001          0.102

10 simulations completed in 101.009 seconds
```

# Enhancements to simple MAC

- The above "simple" MAC protocol may work reasonably under Poisson or sporadic traffic
- It will fail if traffic is fully loaded since the MAC grants reservation requests as soon as it is requested
- A more sophisticated version of the simple MAC that uses exponential backoff etc is shown in `samples/mac/MySimpleThrottledMac.groovy`

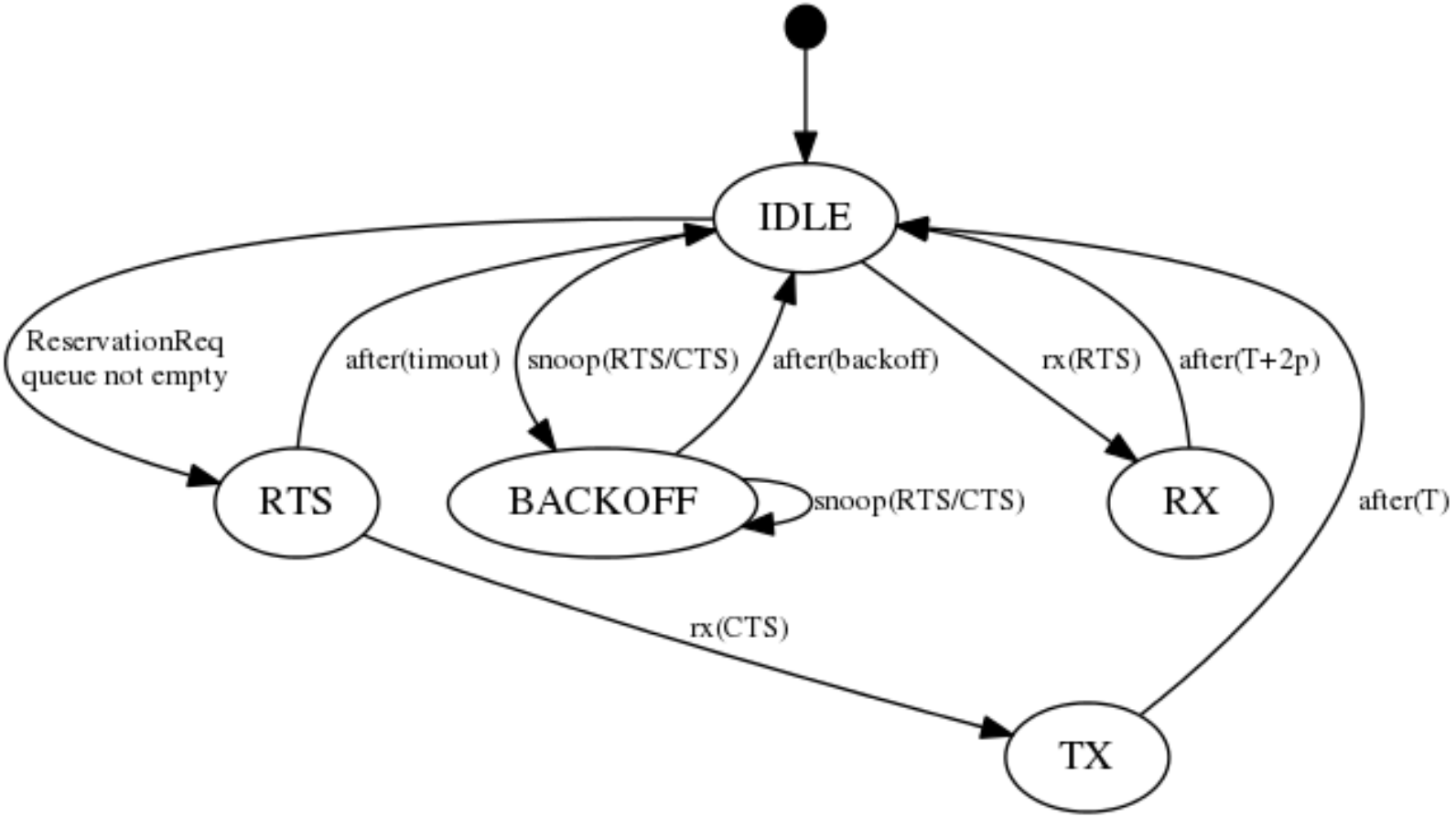
# Handshake MAC

- A simple RTS-CTS 2-way handshake-based MAC agent
- `samples/mac/MySimpleHandshakeMac.groovy`

# Handshake MAC

- Communication protocols are best described using a finite state machine (FSM)
- The FSM for a simple handshake-based MAC agent is shown next
- Models the basics of MAC protocols such as MACA or FAMA that use request-to-send (RTS) and clear-to-send (CTS) for channel reservation
- This can be treated as a Markov Chain for analysis

# Handshake MAC



# Handshake MAC - FSM declaration

- Shall focus on the heart of the implementation first
- Define the FSM states and the events that the FSM reacts to

```
1 enum State {
2   IDLE, RTS, TX, RX, BACKOFF
3 }
4
5 enum Event {
6   RX_RTS, RX_CTS, SNOOP_RTS, SNOOP_CTS
7 }
```

# FSMBuilder- MAC dev made simple and clean!

- The FSMBuilder utility class helps construct a FSMBehavior from declarative representation
- Direct mapping between the FSM diagram and FSM code
- `state(...)`: FSM states
- `onEnter/onExit`: actions upon entering/exiting a state
- `onEvent(...)`: behavior in response to events
- `after(...)`: timers that operate in a state
- `action`: continuous actions while in a state

# Handshake MAC - FSM declaration

```
1  FSMBehavior fsm = FSMBuilder.build {
2    state(State.IDLE) {
3      action { ... }
4      onEvent(Event.RX_RTS) { ... }
5      onEvent(Event.SNOOP_RTS) { ... }
6      onEvent(Event.SNOOP_CTS) { ... }
7    }
8    state(State.RTS) {
9      onEnter { ... }
10     onEvent(Event.RX_CTS) { ... }
11   }
12   state(State.TX) {
13     onEnter { ... }
14   }
15   state(State.RX) {
16     onEnter { ... }
17   }
18   state(State.BACKOFF) {
19     onEnter { ... }
20     onEvent(Event.SNOOP_RTS) { ... }
21     onEvent(Event.SNOOP_CTS) { ... }
22   }
23 }
```

# Handshake MAC - initialization

```
1  void setup() {
2    register Services.MAC
3  }
4
5  void startup() {
6    phy = agentForService Services.PHYSICAL
7    subscribe(phy)
8    subscribe(topic(phy, Physical.SNOOP))
9    add new OneShotBehavior({
10    def nodeInfo= agentForService Services.NODE_INFO
11    addr = get(nodeInfo, NodeInfoParam.address)
12   })
13   add(fsm)
14 }
```

# Handshake MAC - request handler

```
1  Message processRequest(Message msg) {
2    switch (msg) {
3      case ReservationReq:
4        if(msg.to==Address.BROADCAST || msg.to==addr)
5         return new Message(msg, Performative.REFUSE)
6        if(msg.duration<= 0 || msg.duration>
             maxReservationDuration)
7         return new Message(msg, Performative.REFUSE)
8        if(queue.size() >= MAX_QUEUE_LEN)
9         return new Message(msg, Performative.REFUSE)
10       queue.add(msg)
11       fsm.restart() // tell fsm to check queue
12       return new ReservationRsp(msg)
13     case ReservationCancelReq:
14     case ReservationAcceptReq:
15     case TxAckReq:
16       return new Message(msg, Performative.REFUSE)
17   }
18   return null
19 }
```

Queue uses: `Queue<ReservationReq> queue = new ArrayDeque<ReservationReq>()`

# Handshake MAC - message handler

```
1  void processMessage(Message msg) {
2    if (msg instanceof RxFrameNtf && msg.protocol ==
         PROTOCOL){
3      def rx = pdu.decode(msg.data)
4      def info = [from: msg.from, to: msg.to,
5           duration: rx.duration/1000.0]
6      if (rx.type== RTS_PDU) fsm.trigger(info.to== addr ?
7             Event.RX_RTS : Event.SNOOP_RTS, info)
8      else if (rx.type == CTS_PDU) fsm.trigger(info.to ==
           addr?
9             Event.RX_CTS : Event.SNOOP_CTS, info)
10   }
11 }
```

# Handshake MAC - outline done!!

- And that's the general outline of a more complex MAC implementation!! [6]
- Next up, we may look at the details of each state of the Handshake protocol

[6] Runnable code may require a few more little details that can be found at `http://www.unetstack.net/doc/html/mac.html`

# Handshake MAC - IDLE state

```
1   state(State.IDLE) {
2     action {
3       if (!queue.isEmpty()) {
4         after(rnd(0, BACKOFF_RANDOM)) {
5           setNextState(State.RTS)
6         }
7       }
8       block()
9     }
10    onEvent(Event.RX_RTS) { info ->
11      rxInfo = info
12      setNextState(State.RX)
13    }
14    onEvent(Event.SNOOP_RTS) {
15      backoff = RTS_BACKOFF
16      setNextState(State.BACKOFF)
17    }
18    onEvent(Event.SNOOP_CTS) { info ->
19      backoff = info.duration + 2*MAX_PROP_DELAY
20      setNextState(State.BACKOFF)
21    }
22  }
```

# Handshake MAC - RTS state

```
1   state(State.RTS) {
2     onEnter {
3       Message msg = queue.peek()
4       def bytes = pdu.encode(type: RTS_PDU,
5            duration: Math.ceil(msg.duration*1000))
6       phy << new TxFrameReq(to: msg.to, type:
7         Physical.CONTROL, protocol: PROTOCOL, data: bytes)
8       after(CTS_TIMEOUT) {
9         if (++retryCount >= MAX_RETRY) {
10          sendReservationStatusNtf(queue.poll(),
11              ReservationStatus.FAILURE)
12          retryCount = 0
13        }
14        setNextState(State.IDLE)
15      }
16    }
17    onEvent(Event.RX_CTS) {
18      setNextState(State.TX)
19    }
20  }
```

# Handshake MAC - TX state

```
1   state(State.TX) {
2     onEnter {
3       ReservationReq msg = queue.poll()
4       retryCount = 0
5       sendReservationStatusNtf(msg, ReservationStatus.
            START)
6       after(msg.duration) {
7         sendReservationStatusNtf(msg, ReservationStatus.
              END)
8         setNextState(State.IDLE)
9       }
10    }
11  }
```

# Handshake MAC - RX state

```
1   state(State.RX) {
2     onEnter {
3       def bytes = pdu.encode(type: CTS_PDU,
4         duration: Math.round(rxInfo.duration*1000))
5       phy << new TxFrameReq(to: rxInfo.from, type:
6          Physical.CONTROL, protocol: PROTOCOL, data: bytes
               )
7       after(rxInfo.duration + 2*MAX_PROP_DELAY) {
8         setNextState(State.IDLE)
9       }
10      rxInfo = null
11    }
12  }
```

# Handshake MAC - BACKOFF state

```
1   state(State.BACKOFF) {
2     onEnter {
3       after(backoff) {
4         setNextState(State.IDLE)
5       }
6     }
7     onEvent(Event.SNOOP_RTS) {
8       backoff = RTS_BACKOFF
9       reenterState()
10    }
11    onEvent(Event.SNOOP_CTS) { info ->
12      backoff = info.duration + 2*MAX_PROP_DELAY
13      reenterState()
14    }
15  }
```

# Conclusion

- UnetStack and its MAC framework takes away much of the chores of writing complex MAC protocols
- We hope researchers will use this to implement very sophisticated and practical MAC protocols
- Such implementations can then be tested directly in Unet or other compatible modems without any alteration or porting

# Section 6

# Higher layers

# Default Network Stack

```
> ps
node: org.arl.unet.nodeinfo.NodeInfo - IDLE
mac: org.arl.unet.mac.aloha.AlohaACS - IDLE
phy: org.arl.unet.sim.HalfDuplexModem - IDLE
state: org.arl.unet.state.StateManager - IDLE
simulator: SimulationAgent - IDLE
arp: org.arl.unet.addr.AddressResolution - IDLE
rdp: org.arl.unet.net.RouteDiscoveryProtocol - IDLE
remote: org.arl.unet.remote.RemoteControl - IDLE
shell: org.arl.fjage.shell.ShellAgent - IDLE
link: org.arl.unet.link.ReliableLink - IDLE
ranging: org.arl.unet.phy.Ranging - IDLE
transport: org.arl.unet.transport.SWTransport - IDLE
router: org.arl.unet.net.Router - IDLE

>
```

# Node Information

```
> node
node
----
address = 1
canForward = false
diveRate = 0
heading = 0
location = [0.0, 0.0, -15.0]
mobility = false
nodeName = 1
origin = []
speed = 0
time = 1459262304068
turnRate = 0

>
```

# Address Resolution

```
1  > host 'gwbuoy'
2  24
3
4  > host 'redstar-auv'
5  115
6
7  >
```

# Reliable Link

## On node 1

```
1   > link
2   link
3   ----
4   MTU = 2000
5   dataChannel = 1
6   mac = mac
7   maxPropagationDelay = 2.5
8   maxRetries = 3
9   phy = phy
10  reservationGuardTime = 0.5
11
12  > x = new byte[64]
13  > link << new DatagramReq(to: 2, data: x, reliability:
        true)
14  AGREE
15  DatagramDeliveryNtf:INFORM[to:2]
16
17  >
```

# Reliable Link

## On node 2

```
1   > subscribe phy
2   > subscribe link
3
4   RxFrameNtf:INFORM[type:DATA from:1 to:2 protocol:2
        rxTime:1362851667 (32 bytes)]
5   RxFrameNtf:INFORM[type:DATA from:1 to:2 protocol:2
        rxTime:1363088667 (32 bytes)]
6   RxFrameNtf:INFORM[type:DATA from:1 to:2 protocol:2
        rxTime:1363326667 (9 bytes)]
7   DatagramNtf:INFORM[from:1 to:2 protocol:0 (64 bytes)]
8
9   >
```

# Router & Route Discovery

```
> routes
>

> addroute 3,2
> routes
1: to 3 via link/2 [reliable, hops: 0, metric: 1.0]

> trace 3
[1, 2, 3, 1]

> delroutesto 3
> rreq 3
AGREE
> routes
1: to 3 via link/3 [reliable, hops: 1, metric: 3.0]
2: to 2 via link/2 [reliable, hops: 1, metric: 6.0]
3: to 3 via link/2 [reliable, hops: 2, metric: 2.55]
>

> trace 3
[1, 3, 1]

>
```

# Transport

```
> transport
transport
---------
MTU = 65535
link = router
maxRetries = 2
reportProgress = false
timeout = 60.0

> x = new byte[4096]
> transport << new DatagramReq(to: 2, data: x,
    reliability: true)
AGREE
DatagramDeliveryNtf:INFORM[to:2]

>
```

# Remote Access

```
> tell 2, 'hello'
AGREE

> fput 'abc.txt', 2
AGREE

> fget 1, 'abc.txt'
FILE TRANSFERRED

>
```

# Remote Access

```
> n2 = rnode 2
Remote node 2

> n2.phy.MTU
RemoteParamNtf:INFORM[MTU: 16]
> n2.link.maxRetries
RemoteParamNtf:INFORM[maxRetries: 2]
> n2.link.maxRetries = 3;
> n2.link.maxRetries
RemoteParamNtf:INFORM[maxRetries: 3]

> n2 << 'tell 1, "hello"'
OK
RemoteTextNtf:INFORM[from: 2 text: 'hello']

> n2.abc()
OK

>
```

# Section 7

# Propagation delay

# What is the impact of propagation delay on network throughput?

# Definitions

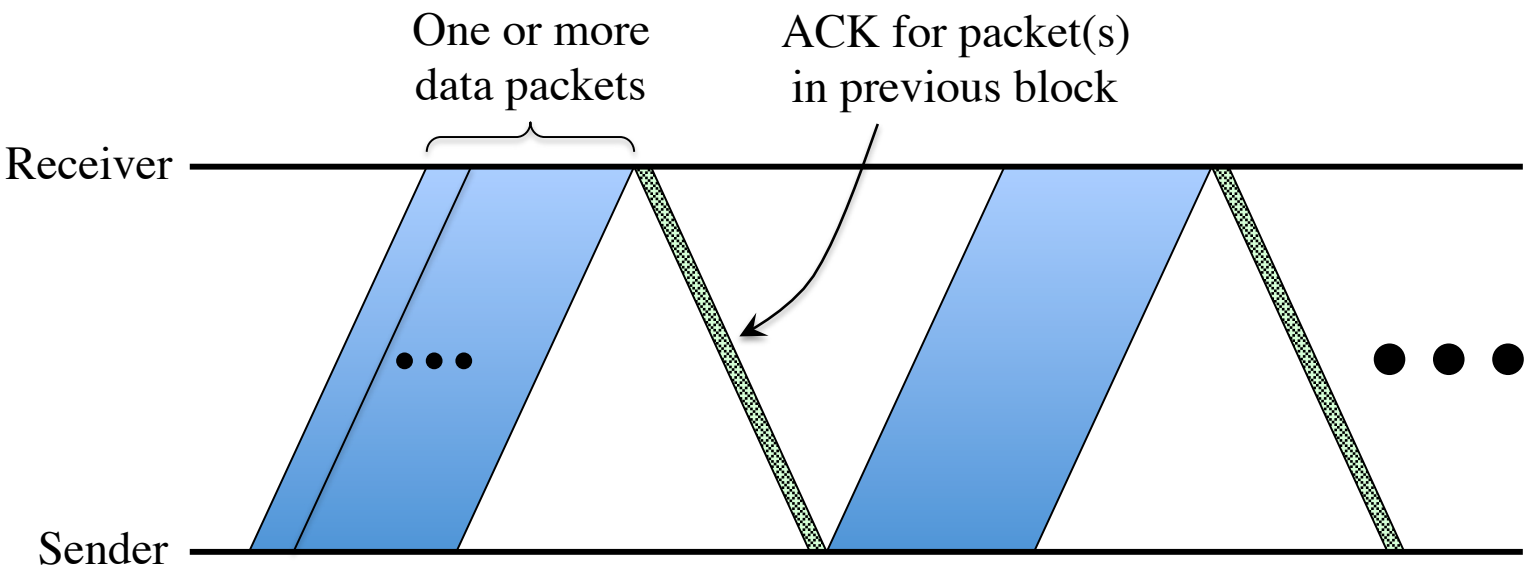
## Long propagation delay

Propagation delay comparable to symbol length, or packet length.

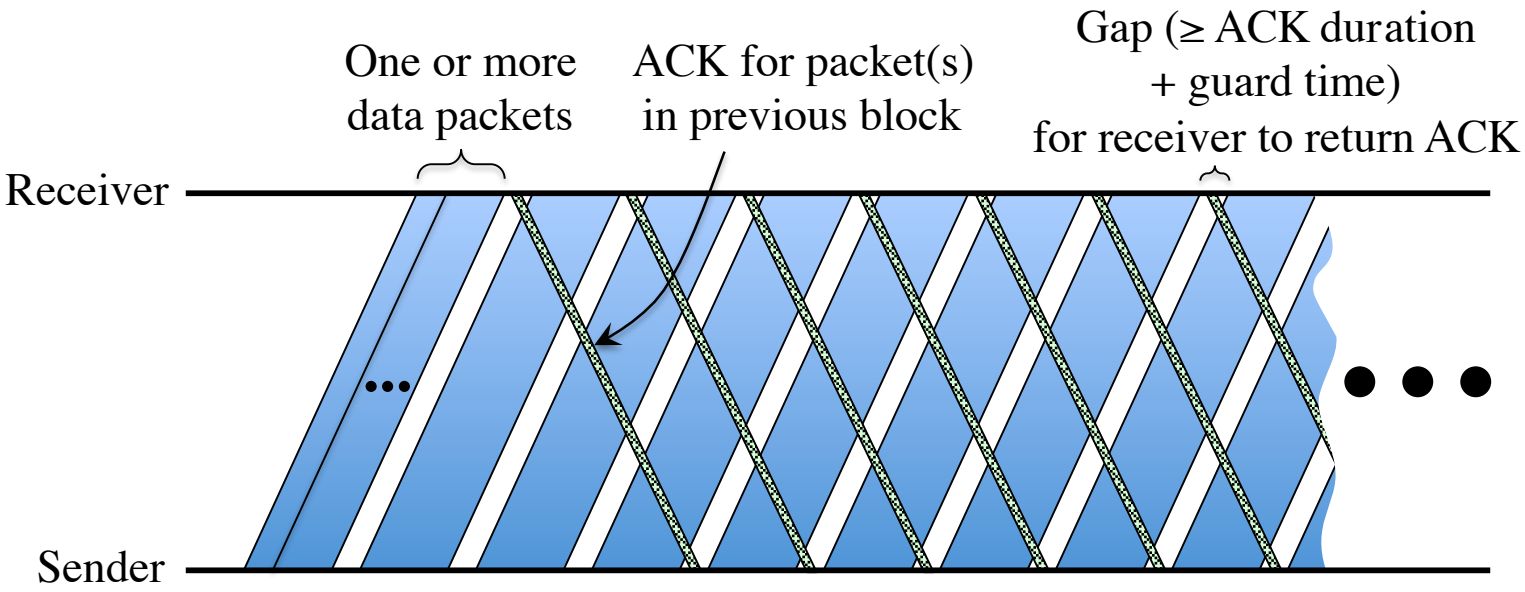
## Normalized network throughput

Network throughput normalized by average link data rate.

# Naïve adaptation of protocols



# Smarter adaptation of protocols




M. Chitre and W.-S. Soh, "Reliable point-to-point underwater acoustic data transfer: To juggle or not to juggle?," IEEE Journal of Oceanic Engineering, vol. 40, no. 1, pp. 93-103, 2014.

# More definitions . . .

## Mitigation

Mitigation suggests that propagation delay inherently reduces throughput, and we can only reduce the severity of the effect.

## Exploitation

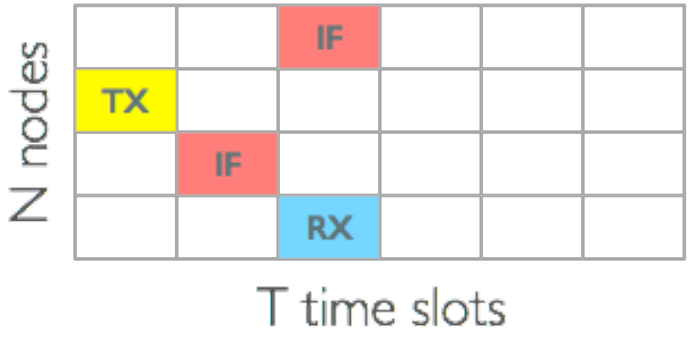
Exploitation suggests that propagation delay allows higher throughput, and appropriately design protocols may be able to gain from it.

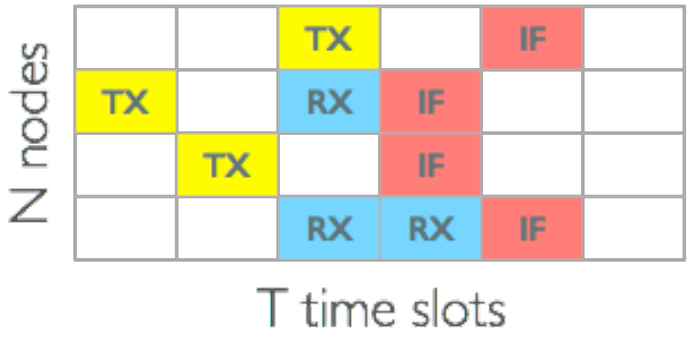
# How big is the opportunity?

- Consider a $N$-node network, and a slotted schedule with slot length $t$ (we can let $t \to 0$, if we like)
- Consider a periodic schedule with $T$ slots (again, we can let $T \to \infty$, if we like)
- There are a total of $TN$ slots available per period
- At most $TN/2$ slots can be used for transmission, as every transmission must correspond to a reception
- Hence the maximum throughput of the network is $N/2$
- This throughput is achieved when on-an-average half the nodes are transmitting simultaneously

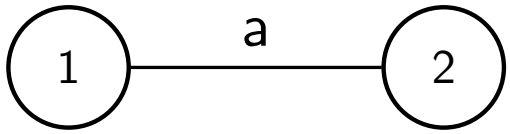
But what about interference?

# But what about interference?



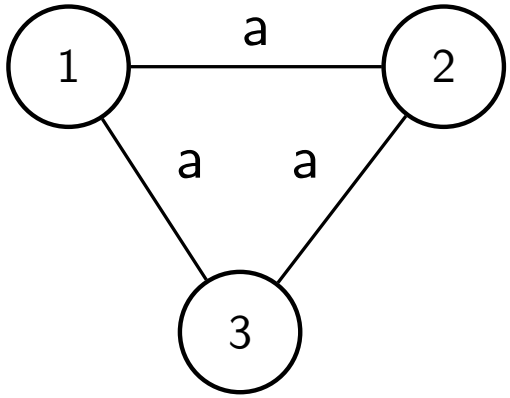

# 2-node network




- Choose time slot length $\tau = a/c$, where $c$ is the sound speed
- then distance matrix $D = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$
- Choose period $T = 2$
- Schedule $Q^{(T)} = \begin{bmatrix} 2 & -2 \\ -1 & 1 \end{bmatrix}$

  (positive numbers represent transmissions to that node, negative numbers represent receptions from that node)
- Throughput $S = 1$

# 3-node network




- $D = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix}$, $\tau = a/c$
- $Q^{(4)} = \begin{bmatrix} 2 & 3 & -3 & -2 \\ -3 & -1 & 1 & 3 \\ -2 & 1 & -1 & 2 \end{bmatrix}$
- $S = 1.5$

# Other networks

- Isosceles triangle ($N = 3$), $S = 1.5$
- Regular tetrahedron ($N = 4$), $S = 2$
- Stretched tetrahedron ($N = 4$), $S = 2$
- Linear ($N = 3$), $S = 1.33 < 1.5$

---

M. Chitre, M. Motani, and S. Shahabudeen, "Throughput of networks with large propagation delays," IEEE Journal of Oceanic Engineering, vol. 37, no. 4, pp. 645-658, 2012.

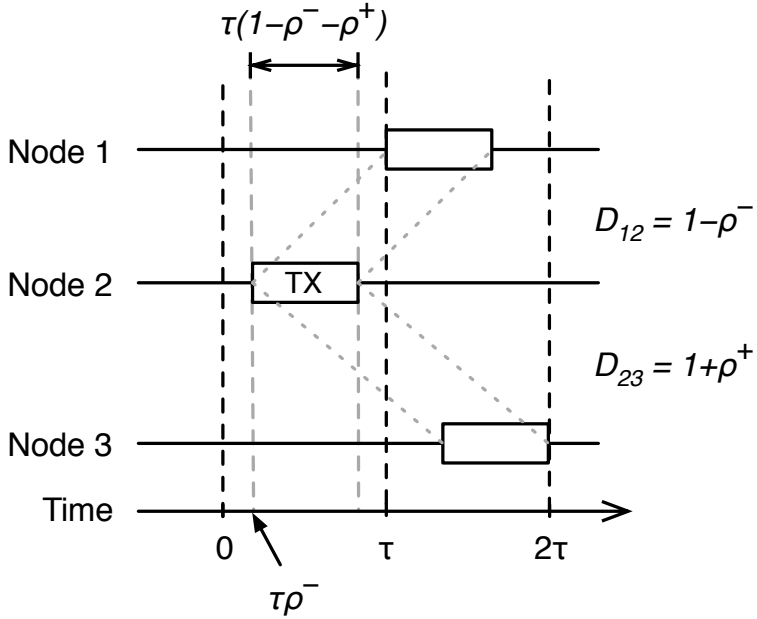
# Non-integer delay matrices

$$X = c \begin{bmatrix} 0 & -4.83 & 0.84 & -3.20 & -1.23 & -1.08 \\ 0 & 1.28 & 0.61 & -2.23 & 0.19 & 1.15 \\ 0 & -0.15 & 1.76 & 0.90 & -1.64 & 1.24 \end{bmatrix}$$

$$D = \begin{bmatrix} 0 & 5.00 & 2.04 & 4.00 & 2.06 & 2.01 \\ 5.00 & 0 & 6.02 & 4.01 & 4.05 & 4.01 \\ 2.04 & 6.02 & 0 & 5.01 & 4.00 & 2.06 \\ 4.00 & 4.01 & 5.01 & 0 & 4.02 & 4.00 \\ 2.06 & 4.05 & 4.00 & 4.02 & 0 & 3.04 \\ 2.01 & 4.01 & 2.06 & 4.00 & 3.04 & 0 \end{bmatrix}$$

# Non-integer delay matrices

- $D \approx \begin{bmatrix} 0 & 5 & 2 & 4 & 2 & 2 \\ 5 & 0 & 6 & 4 & 4 & 4 \\ 2 & 6 & 0 & 5 & 4 & 2 \\ 4 & 4 & 5 & 0 & 4 & 4 \\ 2 & 4 & 4 & 4 & 0 & 3 \\ 2 & 4 & 2 & 4 & 3 & 0 \end{bmatrix}$
- $\rho^+ = 0.05,\ \rho^- = 0$
- $Q^{(2)} = \begin{bmatrix} -2 & 2 \\ -1 & 1 \\ -4 & 4 \\ -3 & 3 \\ -6 & 6 \\ -5 & 5 \end{bmatrix}$
- $S = 2.82 < 3$



# Scheduling algorithms

- Dynamic programming based scheduling:

  M. Chitre, M. Motani, and S. Shahabudeen, "Throughput of networks with large propagation delays," IEEE Journal of Oceanic Engineering, vol. 37, no. 4, pp. 645-658, 2012.
- Scheduling with power control:

  P. Anjangi and M. Chitre, "Scheduling Algorithm with Transmission Power Control for Random Underwater Acoustic Networks," in IEEE OCEANS'15 Genoa, May 2015.
- Scheduling in multihop grid networks:

  S. Lmai, M. Chitre, C. Laot, and S. Houcke, "Throughput-maximizing Transmission Schedules for Underwater Acoustic Multihop Grid Networks," IEEE Journal of Oceanic Engineering, vol 40, no. 4, pp. 853-863 , 2015.

# Scheduling algorithms

- Scheduling in linear networks:

  S. Lmai, M. Chitre, C. Laot, and S. Houcke, "Throughput-efficient Super-TDMA MAC Transmission Schedules in Ad hoc Linear Underwater Acoustic Networks," IEEE Journal of Oceanic Engineering, 2016, (in press).

- Multihop scheduling:

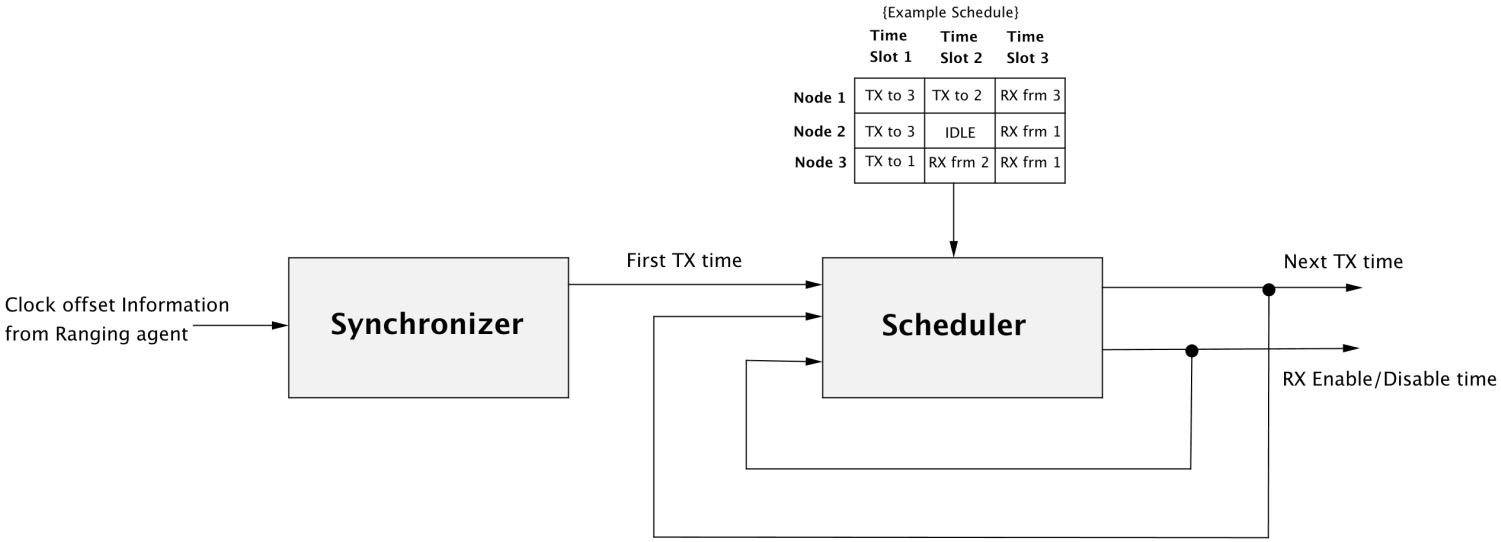
  H. Zeng, Y. T. Hou, Y. Shi, W. Lou, S. Kompella, and S. F. Midkiff, "SHARK-IA: An interference alignment algorithm for multi-hop underwater acoustic networks with large propagation delays," ACM International Conference on Underwater Networks & Systems, 2014.

- MILP based unslotted scheduling:

  C.-C.Hsu, M.-S.Kuo, C.-F.Chou, and K.C.-J.Lin, "The elimination of spatial-temporal uncertainty in underwater sensor networks," IEEE/ACM Transactions on Networking (TON), vol. 21, no. 4, pp. 12291242, 2013.

# Practical implementation

(using UnetStack and Unet-2 modem)




- Objective of Synchronizer: To achieve time synchronization among the nodes in the network
- Objective of Scheduler: To transmit packets at accurate times and prepare the modem for reception at appropriate times as per the schedule

Slide courtesy Prasad Anjangi

# Time synchronization

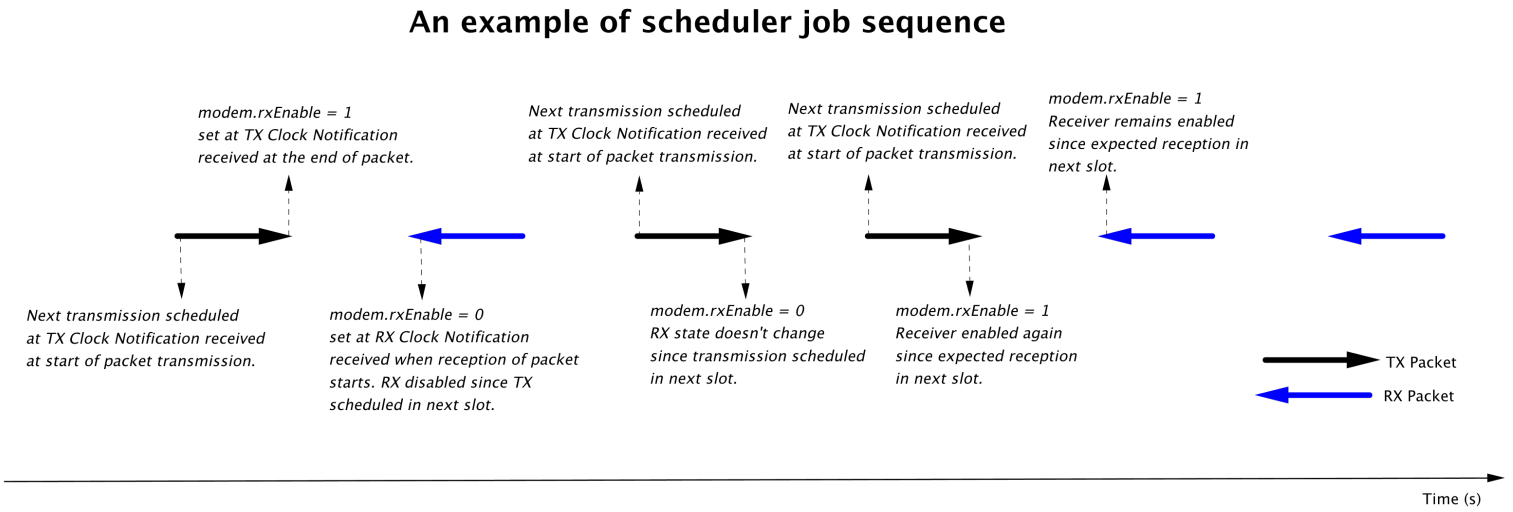
- Step 1: Get the clock offset information from `Ranging` agent
- Step 2: The most advanced node in time computes the synchronization time and transmits to all other nodes in the network

```
/*get all the offset information from ranging */
boolean getOffsetInfo() {
    def ofs = [:]
    laddr.each {
        if (it != me) {
            def rsp = ranging << new SyncInfoReq(it)
            if (rsp.getPerformative() != Performative.FAILURE) {
                log.info 'Sync Information from peer = '+rsp.getPeer()+ 'is offset = '+rsp.getOffset()
                ofs[rsp.getPeer()] = rsp.getOffset()
            }
            else {
                log.info 'SyncInfoReq Failure, Aborting...'
                return false
            }
        }
    }
    offset = 0
    if (ofs.size() != 0) {
        ofs.each {
            if (it.value > offset) {
                offset = it.value
            }
        }
    }
    ................
```

```
if (getOffsetInfo()) { // Returns TRUE if Offset Info available
    add new WakerBehavior(someDelay, {
        // invoked someDelay sec later
        if (offset == 0) {
            // offset is 0 for the most advanced node in time
            // set synctime at the most advanced node --> 1 min from now
            synctime = phy.time + 60000000 // First TX Time
            ssynctime = synctime.toString()
            // Transmit the synctime to all other nodes
            def rsp = link << new DatagramReq(data: ssynctime, protocol: Protocol.USER)
            if (rsp.getPerformative() != Performative.FAILURE) {
                log.info 'the synctime is successfully delivered'
            }
            else {
                log.info 'the synctime was not succesfully delivered'
            }
        }
    })
}
```

Slide courtesy Prasad Anjangi

# Scheduling




- Timed transmissions:
  Transmissions can be scheduled by setting an alarm
- Switching of receiver state:
  The receiver state is enabled and disabled at appropriate times according to the schedule

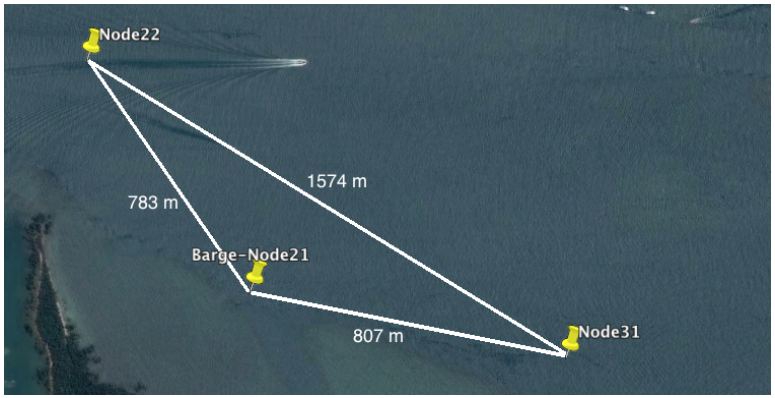
```
/*An example code to transmit test packet
at a specified time t in future*/
scheme[1].txTrigger = 4
def t = phy.time
t = phy.time + timeinFuture
modem.trigger = t
test // transmits a scheme[1] test packet
```

```
/*An example code to switch RX state*/
modem.rxEnable = 0 // Receiver disabled
modem.rxEnable = 1 // Receiver enabled
```

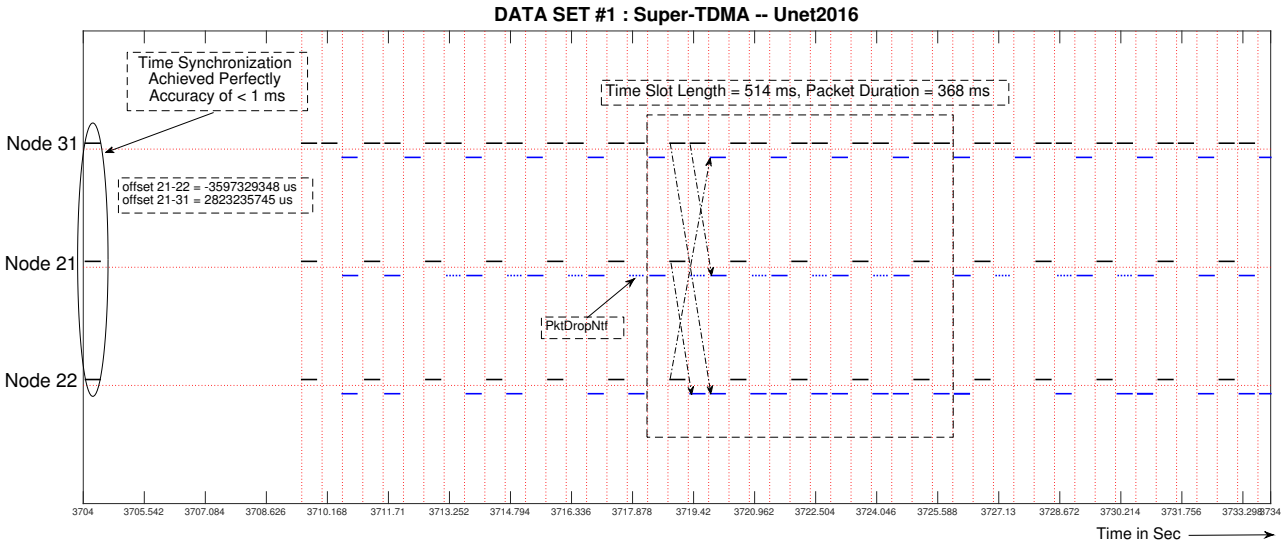
Slide courtesy Prasad Anjangi

# Experimental results

| | Time Slot 1 | Time Slot 2 | Time Slot 3 |
|---|---|---|---|
| **Node 31** | TX to 22 | TX to 21 | RX from 22 |
| **Node 21** | TX to 22 | IDLE | RX from 31 |
| **Node 22** | TX to 31 | RX from 21 | RX from 31 |




$$S = \frac{4}{3} \times \frac{368}{514} = 0.95$$




Slide courtesy Prasad Anjangi

# Section 8

# Other research topics

# Delay and disruption tolerant networks

- Delay-tolerant routing protocols

  M. S. Rahim, et al., "On the Performance of Delay-Tolerant Routing Protocols in Underwater Networks," IEEE OCEANS'11 Santander, 2011.

  Guo, Zheng, et al., "Adaptive routing in underwater delay/disruption tolerant sensor networks," Wireless on Demand Network Systems and Services, 2008. WONS 2008. Fifth Annual Conference on. IEEE, 2008.

- Underwater convergence layer implementation

  D. Merani, et al., "An Underwater Convergence Layer for Disruption Tolerant Networking," 2011 Baltic Congress on Future Internet Communications (BCFIC Riga), 2011.

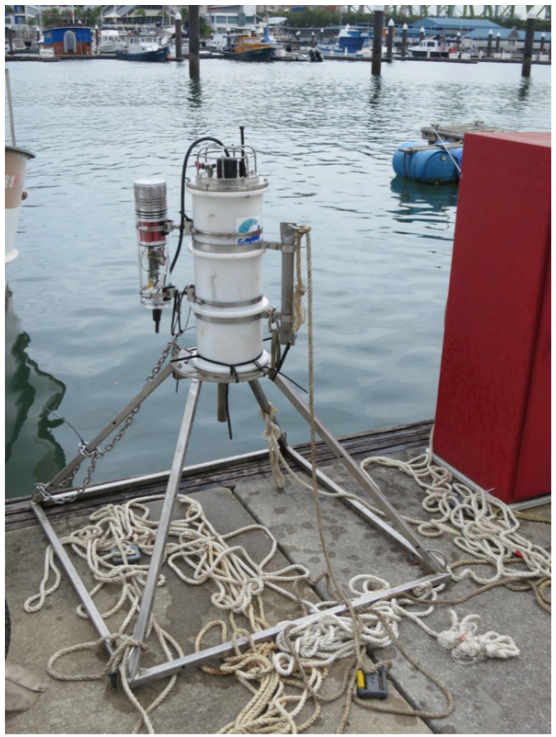
- Survey paper

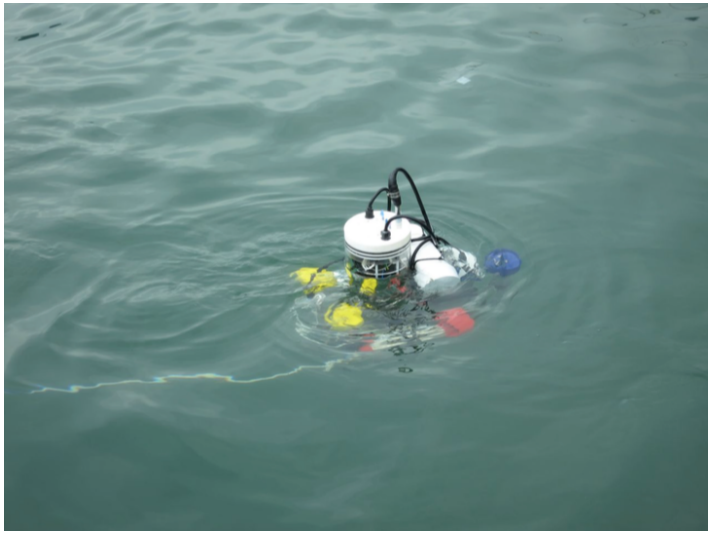
  H. H. Cho, et al., "Survey on underwater delay/disruption tolerant wireless sensor network routing," in IET Wireless Sensor Systems, vol. 4, no. 3, pp. 112-121, September 2014.

# Heterogenous networks

- Acoustic-optical handover for data muling

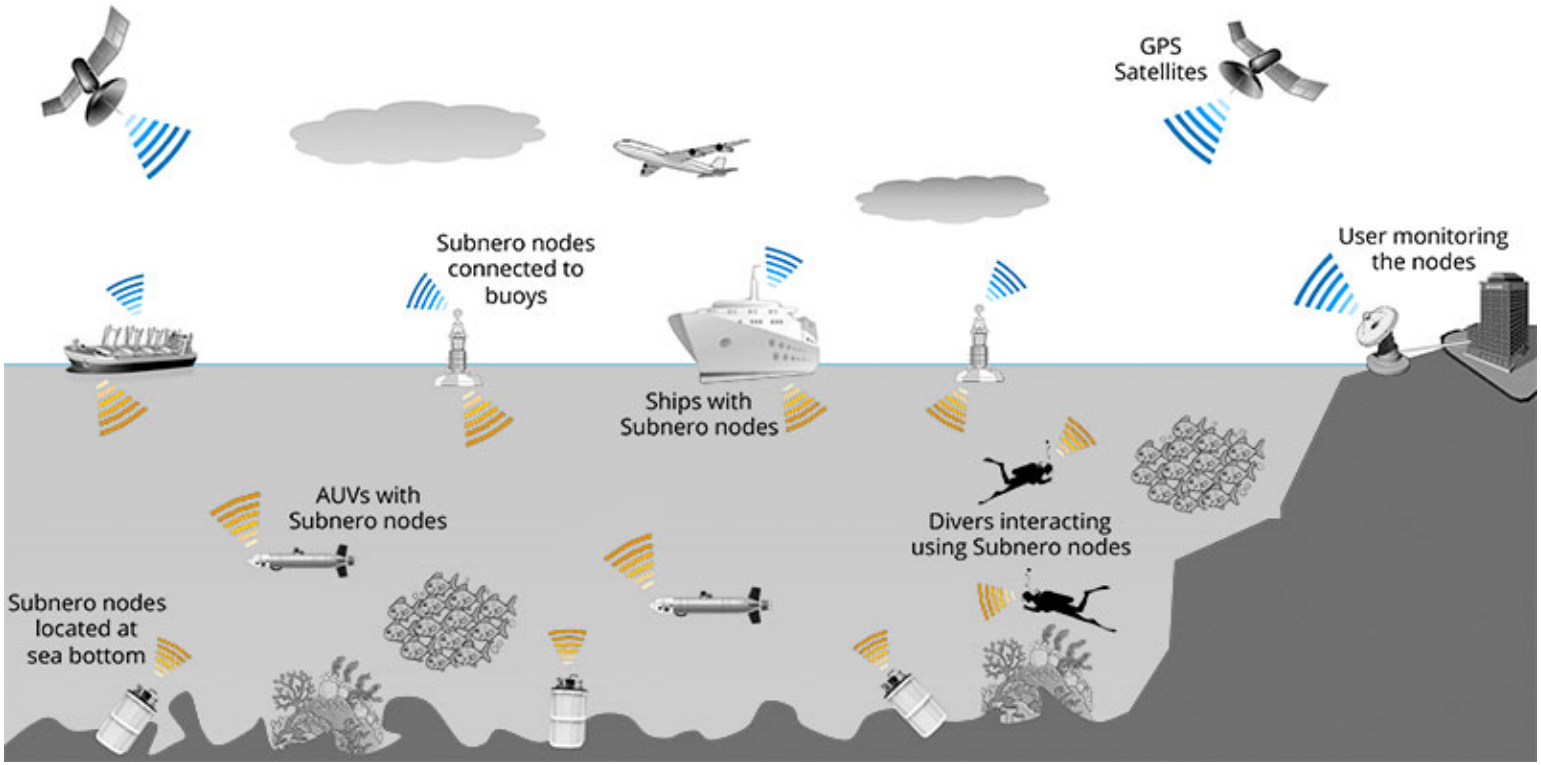
  M. Doniec, I. Topor, M. Chitre, and D. Rus, "Autonomous, localization-free underwater data muling using acoustic and optical communication," in Experimental Robotics (J. P. Desai, ed.), pp. 841-857, Switzerland: Springer International Publishing, 2013.





# Heterogenous networks

An illustration by Subnero showing what an underwater network of the future, with a mix of acoustic, optical and radio links might look like...



# Things that we didn't get to talk about...

- Routing protocols
- Address assignment and resolution protocols
- Erasure codes and ARQ-based protocols
- Network coding
- Network synchronization
- Network localization
- Mobility and network connectivity

and much more...