SWiG Network Simulator

February 2022

Neil Judell

The SWiG network simulator is an attempt to provide a flexible modeling tool to evaluate performance of undersea networks implementing the in-development SWiG protocol. Multiple modulation types are simulated (at a timing and performance level, not a physical level). Modulators and transmitters can be configured to be self-cancelling or non-self-cancelling, to employ CSMA access protocol or to simply transmit whenever a message is present. Signal interference levels are modeled and computed on-the fly. Store-and-forward protocols and mesh protocols are implemented. The network implements the FD/HD two-channel protocol as proposed. Building blocks are Matlab classes, with methods exposed to permit network configuration and operation. Sample scripts are provided to demonstrate use of the class objects as well as to perform useful computations of network performance.

We begin with the classes defined and their methods:

Modulator classes

- modulatorClass – the base class for setting up different types of modulators. This is an abstract class – no actual modulators will be constructed directly from this class. But real modulators will inherit from this class
    - constructor modulatorClass(topBitrate,packetLength,preambleCollisionFatal, fullDuplex,CSMA,centerFrequency,maxInterferenceIn_dB,nominalPreambleDuration,maxBandwidth)
        - topBitrate is payload bit rate in bits per second at full bandwidth
        - packetLength is the payload packet length in bits
        - preambleCollisionFatal is a Boolean – if two preambles of the same modulation type collide, both packets are lost if set. Generally true for SWiG FHSS modulation, untrue for generic DSSS
        - fullduplex is a Boolean – true if equipped with self-noise cancellation, false otherwise
        - CSMA is a Boolean – true if the modem must obey Carrier Sense Multiple Access to begin transmission
        - centerFrequency in Hz
        - maxInterferenceIn_dB is the maximum other-packet interference permitted. Interference levels above this value result in failure to decode
        - nominalPreambleDuration – duration of the preamble in seconds at full-bandwidth
        - maxBandwidth – nominal bandwidth in Hz when bandwidth is full
    - resetModulator – resets the modulator to nominal bandwidth and center frequency
    - setBandwidthFraction(fraction) uses a value of 0 to 1 to set actual bandwidth
    - getBandwidthFraction
    - setCenterFrequency(frequency) sets the center frequency in Hz

  - getCenterFrequency
  - getDuplex
  - getCSMA
  - getBandOverlap(otherModulator) determines fraction of this modulator's band that overlaps with otherModulator. Used for computing interference levels
  - getPacketLength – get packet length in bits
  - getPacketDuration – in seconds
  - getModulatorType – returns a string describing the modulator. This is an abstract function – real modulators must define this
  - packetValid(delay) – delay in seconds is propagation delay for this modulator. A Boolean that gives an inherent probabilistic physical layer Boolean. I.e. will stochastically give a value of true or false based on distance. This is an abstract function to be defined in the real modulator. Generally, most modulators lose 5% of all packets, and then a lot more as distance goes up to a threshold range based on distance.
  - attenuation(delay) – delay is propagation time in seconds, returns attenuation in dB based on r^2 loss and absorption at the center frequency
  - isHDModulator – returns a Boolean indicating whether the modulator is used for the HD high-speed channel. Base class returns false
- midbandModulatorClass – a class inheriting from modulatorClass, used in other modulators
  - constructor midBandModulatorClass(topBitrate,packetLength,preambleCollisionFatal. fullDuplex,CSMA,centerFrequency,maxInterferenceIn_dB,nominalPreambleDuration,maxBandwidth) – see modulatorClass constructor
  - packetValid(delay) – overrides base class for center frequency of 21 kHz
  - attenuation(delay) – overrides base class for center frequency of 21 kHz
- SWIGPrimitiveModulator – inherits from midBandModulatorClass. Implements a SWiG FHSS modulator and demodulator. Configures the specifics for a SWiG modulator that has bandwidth and center frequency flexibility, but cannot decode multiple received signals at once
  - Constructor SWIGPrimitiveModulator(fullDuplex,CSMA) – fullduplex is a boolean indicating whether the modem can self-cancel. CSMA is a Boolean indicating whether the modem must obey CSMA to access the channel
  - getModulatorType – returns 'SWiGPrimitive'
- SWIGModulator – inherits from midBandModulatorClass. Implements a SWiG FHSS modulator and demodulator. Configures the specifics for a SWiG modulator that has bandwidth and center frequency flexibility, and has the ability to decode multiple received signals at once
  - Constructor SWIGModulator(fullDuplex,CSMA) – fullduplex is a boolean indicating whether the modem can self-cancel. CSMA is a Boolean indicating whether the modem must obey CSMA to access the channel
  - getModulatorType – returns 'SWiG'
- genericDSSSModulator – inherits from midBandModulatorClass. Implements a generic DSSS modulation scheme. Primary differences from the SWiG variants – more susceptible to interference, collision of preambles is not fatal. Default bandwidth and center frequency are different.

  - Constructor genericDSSSModulator(fullduplex,CSMA) – fullduplex is a boolean indicating whether the modem can self-cancel. CSMA is a Boolean indicating whether the modem must obey CSMA to access the channel
  - getModulatorType – returns 'DSSS'
  - packetValid(delay) – overrides base class for this specific modulator
  - attenuation(delay) – overrides base class for this specific modulator
- QPSKModulator – implements a high-speed QPSK modulator to demonstrate use of the HD high-speed channel
  - Constructor QPSKModulator(fullduplex,CSMA) – fullduplex is a boolean indicating whether the modem can self-cancel. CSMA is a Boolean indicating whether the modem must obey CSMA to access the channel
  - getModulatorType – returns 'QPSK'
  - packetValid(delay) – overrides base class for this specific modulator
  - attenuation(delay) – overrides base class for this specific modulator
  - isHDModulator – Boolean returning true for this modulator

Packet classes

- packetClass – a class that represents a data packet. Contains internal functions used for routing and evaluation of performance, as well as source, destination, message codes, etc.
  - constructor (mod,src,dest,respReq,IDsend,IDack,data) –
    - mod is a modulator-type object, needed to compute performance
    - src is the source node number
    - dest is the destination node number
    - respReq a boolean indicating whether this message is critical and requires acknowledgment
    - IDsend – a unique code for the sent message for tracking
    - IDack – if this is an acknowledgment packet, this is the IDsend value for the original message
    - Data – the binary message itself
  - setModulator(mod) set the modulator for the packet
  - setPacketDelay(delay) delay in seconds
  - getPacketDelay
  - getPacketDuration – returns total acoustic packet duration in seconds
  - setHop – set the next hop for a mesh network
  - getHop – get the destination hop for a mesh network
- packetFIFOClass – implements a FIFO for packets. Useful for queueing packets to be sent or processed
  - constructor packetFIFOClass(maxDepth) – maxDepth is the maximum depth of the FIFO. Items pushed on the FIFO that would exceed this depth are thrown away
  - isEmpty – Boolean indicating the FIFO is empty
  - push(packet) – pushes a packet on the FIFO
  - head – returns the packet at the head of the FIFO
  - pop – returns the head packet and removes from the FIFO

- packetDequeClass – a variant of a deque that is useful for packets. Packets can be added arbitrarily, removed arbitrarily. Also includes a packetValid Boolean for each packet in the deque
    - constructor packetDequeClass(maxQueued) – maxQueued is the initial size of the empty deque. The deque will automatically expand as needed
    - add(packet) – adds a packet to the queue. Sets the valid bit for that packet to true
    - [packets, validities, indices] = packets – returns all packets present in the deque as an array. Validities is the Boolean array specifying which packets are valid. Indices is the array of indices within the deque where the packets are located
    - Invalidate(index) – invalidate the packet residing at the specified index
    - Remove(index) – remove the packet residing at the specified index

Node class

- nodeClass – a class for describing and implementing a node in an acoustic network. The node can be an ordinary node, a store-and-repeat node or a mesh infrastructure node. A node can be configured as both store-and-repeat as well as mesh, though the store-and-repeat table takes precedence.
    - constructor nodeClass(location,modulators,ID,sendQueueDepth) – location is an (x,y,z) triple in meters describing the location of the node. Modulators is a cell array consisting of the supported modulators at the node. ID is a unique node ID – typically a small integer. sendQueueDepth is the depth of the packet FIFO to be used for queueing messages to be sent
    - setStoreAndForwardTable(table) – table is an array of node IDs. Setting this function causes the node to act as a store-and-forward node. Any messages the node receives that have a destination within the table will be forwarded.
    - setMeshRouteTable(table) – table is a struct. Table.clientList is an array containing the node IDs serviced by this infrastructure node. Table.nextHop is the node ID of the nearest mesh infrastructure node for forwarding. Setting this table causes the node to act as a mesh infrastructure node. Messages received from either a client or another infrastructure node will be forwarded – to a client if it is attached to this node, or to the next infrastructure node otherwise.
    - pushPacketsToSend(packets) – packets is an array of packets to be sent by the node. These are pushed on the packet FIFO to await transmission. Transmission will occur according to the appropriate rules for the active modulator
    - setModulator(modulatorIndex) – sets the currently active modulator at the node. This is the index into the original modulators array used during construction.
    - setBandwidthFraction(fraction) – sets the bandwidth of the active modulator. Fraction is a value between 0 and 1, with 1 being the full nominal bandwidth
    - setCenterFrequency(frequency) – sets the center frequency of the active modulator. Frequency is in Hz
    - scheduleHDChannelEvent(time,duration,fullNodeList,newFDcenterFrequency, newFDbandwidthFraction,HDDestination,HDCenterFrequency,HDBandwidthFraction,HD modulatorIndex,HDMessageList) – prepare a node to transition to be a source for the HD high-speed channel. Once configured, the node sends out configuration methods to the rest of the network and forces acknowledgment. Once all are acknowledged, or a

timeout occurs, the node assumes the rest of the network has reconfigured to give this node access to the high-speed channel. The node will then send its messages on the high-speed channel

  - time is the time in absolute seconds to begin the high-speed operation
  - duration is the time in seconds that the node will need the high-speed channel
  - fullNodeList is an array of node IDs to notify of the new configuration
  - newFDcenterFrequency is the new center frequency for the FD channel. This and the bandwidth fraction as well as the HD parameters must be set to ensure little or no overlap between the FD and HD channels
  - newFDbandwidthFraction is the new bandwidth fraction for the FD channel
  - HDDestination is the destination node for the HD communications (this node is the source)
  - HDCenterFrequency is the center frequency for the HD channel modulator
  - HDBandwidthFraction is the bandwidth fraction of nominal to be used by the HD modulator
  - HDmodulatorIndex is the index into the node modulators array for the HD modulator
  - HDMessageList is a cell array of the packets to be sent of the HD high-speed channel

- [receivedPackets,localSendingPacket] = run(time) – run the node. Does the following order of operations:
    - Gets valid received packets from the modem
    - Handles timed-out ACKs (acknowledgments) – if the node is waiting for an ACK and hasn't seen it for too long, it will queue up to send the message requiring ACK again
    - Handles received ACKs – if it's waiting for an ACK, it removes the message from the waiting list
    - ACKasNeeded – for any messages this node has received that require acknowledgment, it queues those ACKs for transmission
    - Forward stored messages – handles forwarding any store-and-forward messages that require transmission
    - Handle mesh forwarding – if this is a mesh infrastructure node, looks at received messages to see if they need forwarding
    - Handle any FD configuration messages the node has received – queueing activity for later, if necessary
    - Handle any FD configuration changes – based on queued FD configuration messages
  - time is the absolute time in seconds of the simulation
  - receivedPackets is an array of packets that have been received by this node (if this node is the destination node only)
  - localSendingPacket is either void or a single packet representing the packet this node has just finished transmitting

Utility Functions

Various functions that are useful for running or evaluating networks

- modulatorIndex(modulator) – utility function for converting a modulator type to an index integer. Returns 1 for SWiGModualtor, 2 for generic DSSS, 3 for SWiG primitive, 4 for QPSK
- results = analyzeSimulationResults(sentPacketInfo,receivedPacketInfo) – compute statistics from a run of a network simulation
  - sentPacketInfo is an n x 3 array of packets that were externally queued for transmission. Column 1 is the sent ID number, column 2 is 0 or 1 for whether the message required ACK, and column 3 is the sent time for the message. Message IDs greater than 100,000 are HD channel messages, message IDs below 100,000 are FD channel messages
  - receivedPacketInfo is an n x 3 array of packets that were determined to have been received. Column 1 is message ID for the sent message, column 2 is the ACK ID, column 3 is the time received
  - results is a struct with a statistical breakdown of the network performance
    - results. FDnumMessagesSent is the number of messages sent on the FD channel
    - results. FDnumMessagesLost is the number of FD messages that were lost
    - results. FDnumAckRequiredMessages is the number of FD messages sent that required ACK
    - results. FDnumAckRequiredMessagesLost is the number of FD channel messages requiring ACK that never were successful acknowledged
    - results. FDmeanLatency is the mean time from end of transmission to end of reception for the FD channel
    - results. FDsigmaLatency is the standard deviation of transmission latency for the FD channel
    - results.FDmedianLatency is the maximum latency measured for the FD channel
    - results. FDminLatency is the minimum latency for the FD channel
    - results.HDnumMessagesSent is the number of HD channel messages sent
    - results. HDnumMessagesSent is the number of FD channel messages sent
    - results. HDnumMessagesLost is the number of FD channel messages lost
- y = sigmoid(x,c,a) – sigmoid function – utility for validating physical layer
- [sentPacketInfo,receivedPacketInfo] = runSimulation(nodes,timeToRun, timeToFinish,timeIncrement,poissonSendInterval,pAckNeeded) – run a simulation of a network without use of HD channel.
  - Nodes is a cell array of nodeClass objects describing the networm
  - timeToRun is how long (in seconds) to run the network
  - timeToFinish is how long before the end of simulation to stop adding new messages to allow the network to finish processing
  - timeIncrement is the increment time in seconds for the network simulation
  - poissonSendInterval – the mean time in seconds between single node transmissions on the FD channel
  - pAckNeeded is the probability that any individual message will require and ACK (0 to 1)
  - sentPacketInfo is as described in the statistical analysis function
  - receivedPacketInfo is as described in the statistical analysis function

- [sentPacketInfo,receivedPacketInfo] = runSimulationWithHDFDchangeover(nodes,timeToRun, timeToFinish,timeIncrement,poissonSendInterval,pAckNeeded,sendHDnodeNumber,receiveHDnodeNumber, timeToDoHD, durationForHD, messagesForHD,modulatorForHD) – runs a network simulation, but schedules temporary use of the HD high-speed channel
  - Nodes is a cell array of nodeClass objects describing the networm
  - timeToRun is how long (in seconds) to run the network
  - timeToFinish is how long before the end of simulation to stop adding new messages to allow the network to finish processing
  - timeIncrement is the increment time in seconds for the network simulation
  - poissonSendInterval – the mean time in seconds between single node transmissions on the FD channel
  - pAckNeeded is the probability that any individual message will require and ACK (0 to 1)
  - sendHDnodeNumber is the node to be used as source for the HD channel event
  - receiveHDnodeNumber is the node to be used as the destination for the HD channel event
  - timeToDoHD is the simulation time, in seconds, to begin the HD channel request
  - durationForHD is the time, in seconds, that the HD channel will be needed
  - messagesForHD is a cell array of just the message data to be sent over the HD high-speed channel
  - modulatorForHD is index into the modulator array for the modulator to be used for the HD high-speed channel
  - sentPacketInfo is as described in the statistical analysis function
  - receivedPacketInfo is as described in the statistical analysis function

Example and useful scripts

- runTestStoreAndForward – tests a set of networks and displays results in the store-and-forward configuration.
  - 15 nodes in a 6km x 6km x 600m uniform distribution
  - DSSS configuration is 511 bit packets at 1024 bits per second, 10 kHz bandwidth, requires 6 dB ${E_b}/{(N_0 + I_0)}$, with 30 dB serial/parallel simultaneous message cancellation.
  - No HD channel operation
  - Trials are run at 0, 1, 2 and 4 store-and-forward nodes with near-optimum placement
  - Simulated runtime is one hour
  - Approximately 0.06 messages per node per second
  - Approximately 10% of messages require ACK
  - Statistics are gathered and displayed for each condition
- runTestMesh – tests a set of networks and displays results in the mesh configuration
  - 15 nodes in a 6km x 6km x 600m uniform distribution
  - DSSS configuration is 511 bit packets at 1024 bits per second, 10 kHz bandwidth, requires 6 dB ${E_b}/{(N_0 + I_0)}$, with 30 dB serial/parallel simultaneous message cancellation.

  - No HD channel operation
  - Trials are run at 2 and 4 mesh infrastructure nodes with near-optimum placement
  - Simulated runtime is one hour
  - Approximately 0.06 messages per node per second
  - Approximately 10% of messages require ACK
  - Statistics are gathered and displayed for each condition
- runFDHDtest – tests a network that transitions from FD-only to both FD and HD back to FD only
  - 8 nodes in a 2km x 2km x 200m uniform distribution
  - FD nodes tested as SWiG level 1 and as DSSS, self-cancelling, no CSMA.
  - Simulated runtime is 20 minutes
  - HD channel operation is slated to begin at 800 seconds into the simulation, and to last for 50 seconds
  - QPSK full bandwidth is to be used for the HD channel, while 0.25 fractional bandwidth SWiG is to be used for the FD channel during HD operation
  - Approximately one message per node per 0.06 seconds on the FD channel (except for HD source and destination during HD operation)
  - Approximately 10% of FD messages require ACK
  - Network is point-to-point

Simulation Results and Comments

The tables below summarize results of running simulations with no message forwarding, store-and-forward and mesh networking. (Theoretical max throughput 460 bits per second for DSSS, 53.1 for SWiG level 1). (Values with a / denote values for non-critical messages/critical messages)

DSSS, SC, no CSMA	0 repeaters	1 repeater	2 store & forward	4 store & forward	2 Mesh	4 Mesh
Fraction messages lost	0.3/0.2	0.03/0	0.02/0	0.006/0	0.06/0.02	0.3/0.2
Mean latency	2.6/5.0	3.5/4.6	3.6/4.3	3.7/4.2	3.5/4.1	2.8/4.2
Sigma latency	0.9/9.6	1.7/6.2	1.8/4.6	1.8/4.6	1.8/5.7	1.2/7.8
Median latency	2.7/2.8	3.2/3.2	3.2/3.4	3.3/3.3	3.1/3.1	2,7/2.7
Max latency	4.5/92	11.3/38	13/37	11/37	14/39.9	10/64
FD throughput (bps)	328	448	442	453	413	337

Figure 1: Simulation results - DSSS modulator, self-cancelling, no CSMA

DSSS, NSC, no CSMA	0 repeaters	1 repeater	2 store & forward	4 store & forward	2 Mesh	4 Mesh
Fraction messages lost	0.4/0.3	0.3/0	0.3/0	0.2/0	0.3/0.03	0.4/0.2
Mean latency	2.7/47.5	3.1/29.2	3.2/33	3.5/22	3.1/37.9	2.7/56.8
Sigma latency	0.9/200	1.3/74	1.5/89	1.9/57	1.3/99.1	1.1/191
Median latency	2.8/2.9	2.9/3.6	2.9/3.6	3.1/3.8	2.9/3.8	2.7/3.1
Max latency	5/2197	6.8/577	10.1/757	12/607	9.6/997	8.5/1566
FD throughput (bps)	271	345	342	348	325	277

Figure 2: DSSs results, not self-cancelling , no CSMA

DSSS, NSC, CSMA	0 repeaters	1 repeater	2 store & forward	4 store & forward	2 Mesh	4 Mesh
Fraction messages lost	0.4/0.3	0.4/0.3	0.4/0.4	0.4/0.4	0.4/0.4	0.4/0.4
Mean latency	172/203	482/488	414/431	516/584	429/446	199/199
Sigma latency	211/243	537/514	423/443	487/485	623/651	245/225
Median latency	87/88	278/295	263/261	372.436	170/166	113/138
Max latency	1199/1154	2558/2432	2361/2075	2325/2258	3174/2534	1406/1427
FD throughput (bps)	268	282	266	263	261	271

Figure 3: DSSS results, not self-cancelling, CSMA

Primitive SWiG, NSC, CSMA	0 repeaters	1 repeater	2 store & forward	4 store & forward	2 Mesh	4 Mesh
Fraction messages lost	0.5/0.5	0.5/0.4	0.5/0.4	0.5/0.5	0.4/0.4	0.6/0.6
Mean latency	169/184	341/342	341/342	708/706	543/564	160/156
Sigma latency	227/230	437/400	432/400	706/750	514/527	149/155
Median latency	93.0/90.0	177/197	177/197	451/420	377/401	112/94.3
Max latency	1804/1120	2386/1896	2386/1896	3094/3111	2615/2162	700/749
FD throughput (bps)	22.5	22.9	22.9	19.8	24.1	19.5

Figure 4: SWiG primitive modem, not self-cancelling, CSMA

Discussion of simulation results

The simulation was laid out to be large enough to assure a high probability of failure for point-to-point communications for paths extending most of the distance of the field. This ensures adequate testing of various routing protocols.

Self-cancellation vs. no self-cancellation

Self-cancellation can have a significant impact on packet loss, latency and overall network throughput for networks that are fairly heavily loaded.

CSMA vs no CSMA

As noted above, for point-to-point communications, CSMA significantly improves maximum latency, and somewhat improves reliability. Once routing is included, the use of CSMA significantly worsens latency, and may even make reliability worse.

Store-and-forward vs. Mesh

In general, store-and-forward outperforms point-to-point and mesh. Latencies are lower, reliability higher. Increasing the number of store-and-forward nodes generally does not hurt performance – until we use CSMA, when the increase in latency harms both throughput and reliability. Once routing is used, CSMA is undesirable, due to dramatic increase in latency. Primitive SWiG – with no ability to decode multiple messages simultaneously, poses the biggest performance issue – and it appears that the best choice is very careful selection of number of store-and-repeat nodes and their locations.

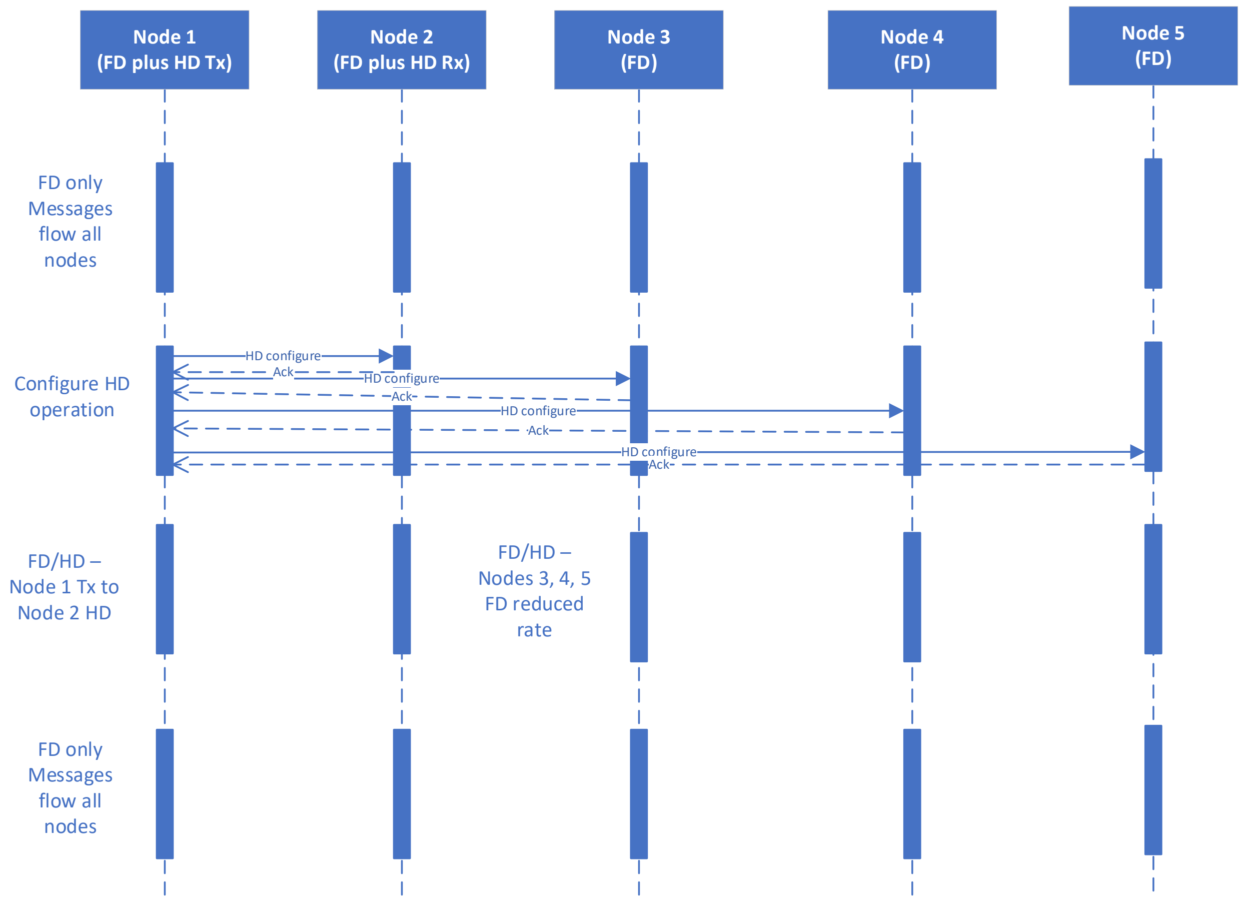
Network configuration recommendations

- Legacy SWiG hardware (HD, CSMA, no simultaneous decode)
  - Either point-to-point, or very careful design of store-and-repeat selections

- General point-to-point
    - Self-cancellation, no CSMA, multiple simultaneous decode provides best solution for reliability and latency
- General routing
    - Self-cancellation, no CSMA, multiple simultaneous decode with several store-and-forward nodes provides highest reliability and lowest latency
    - No self-cancellation, no CSMA, multiple simultaneous decode with several store-and-forward nodes provides performance nearly equal to the self-cancelling implementation

Dual-channel operation

The proposed protocol includes an important provision for allowing high-speed transfers using virtually any modulation, provided that modulation does not interfere with other network communications. The so-called FD low-speed channel is normally configured for full bandwidth. When a node requires high-speed access, it sends configuration messages to the other nodes, requesting a change in bandwidth and/or center frequency. These messages require acknowledgment. The non-high-speed nodes reconfigure themselves as requested, and the high-speed node transmits in its new modulation method on the HD high-speed channel. After a designated timeout, the network reverts to FD low-speed only.



A network of eight nodes was configured in a 2 km by 2 km by 200 meter box. Point-to-point routing was used. An advanced SWiG modulator was employed for the FD channel, using the standard center frequency and bandwidth, non-self-cancelling, no CSMA, and simultaneous multiple decoding. FD message rates of one per minute per node were employed, with a 0.1 probability that a message requires ACK. The simulation was for a 20-minute period. An HD high-speed access request was queued for 750 seconds into the simulation, lasting for 54 seconds. 400 1206-bit high-speed messages were queued for transmission at approximately 9Kbits/sec. No ACK was required for any high-speed packet. The high-speed modulator was QPSK with a center frequency of 25 kHz and a bandwidth of 10 kHz. During HD operation, the FD network was told to go to the lower 25% of the SWiG standard band, resulting in a 4x throughput decrease for the FD network packets during HD operation.

The modulator error function generally provides for about a 5% packet loss on unacknowledged messages over this type of network.

HD Simulation	SWiG Level 1	DSSS, SC, no CSMA
FD fraction of messages lost	0.07/0.06	0.03/0
FD mean latency	8.1/9.8	1.5/3.6
FD sigma latency	13/14	0.5/9.8
FD median latency	4.3/2.6	1.4/1.6
FD max latency	112/64	5.7/62
Overall FD throughput	20.7	222
FD throughput when HD unused	20.9	222
FD throughput when HD in use	16.5	219
HD throughput	8667	8532
HD fraction of packets lost	0.04	0.05

Figure 5: Results of FD/HD simulation