# Plan de Expansión de Simulaciones NS-3 para Tesis Doctoral en IoT

Este plan detalla una hoja de ruta por etapas (MODs) para expandir gradualmente las simulaciones existentes (“simulaciones 9 MOD1”) en NS-3. El objetivo es incorporar nuevas métricas, protocolos y escenarios, haciendo que los resultados sean robustos y relevantes para una tesis doctoral sobre la evaluación de protocolos de ruteo en redes IoT, con miras a la integración de redes neuronales y SDN.

## MOD2: Métricas de Protocolos de Ruteo Avanzadas

**Objetivo:** Integrar la recolección de métricas clave específicas de los protocolos de ruteo: Overhead de Ruteo y Tiempo de Convergencia.

**Impacto en “simulacioniot.cc”:** Alto. Se requiere modificar el código C++ para instrumentar la captura de estos datos.

**Impacto en “run\_simulations.sh”:** Bajo. Principalmente asegurar que los nuevos archivos/datos se muevan y se mantenga la estructura.

**Acciones Detalladas:**

1. **Instrumentación para Overhead de Ruteo:**
   * **En simulacioniot.cc:**
     + **Contadores de Mensajes de Control:** Para cada protocolo (AODV, OLSR, DSDV, DSR), identifica los tipos de paquetes de control (e.g., RREQ, RREP, RERR para AODV; HELLO, TC para OLSR; actualizaciones de tabla para DSDV/DSR).
     + Implementa contadores estáticos o un std::map<std::string, uint64\_t> para almacenar el número de paquetes de control enviados y/o recibidos por cada nodo o globalmente.
     + Engancha estos contadores a los “trace sources” apropiados de cada módulo de enrutamiento (ej. ns3::aodv::RoutingProtocol::Rx o Tx para paquetes de control, si están disponibles y son lo suficientemente específicos). Esto puede requerir bucear en la documentación del módulo de enrutamiento de NS-3 para encontrar los trace sources exactos.
     + Al final de la simulación (CalculateMetrics o una nueva función de resumen), calcula el overhead total (número de paquetes de control) y el overhead por paquete de datos enviado (relación).
     + **Ejemplo de pseudo-código para AODV:** ```cpp // Global counter or map static uint64\_t g\_aodvRreqSent = 0; static uint64\_t g\_aodvRrepSent = 0; // … add other control message types
     + // Trace callback function for AODV RREQ Tx void AodvRreqTxTrace(Ptr packet) { g\_aodvRreqSent++; // Optionally log time and node ID }
     + // In your main simulation setup: // Assuming ‘routingHelper’ is configured for AODV // Iterate over nodes and connect to trace sources for AODV protocol for (uint32\_t i = 0; i < allNodes.GetN(); ++i) { Ptr node = allNodes.Get(i); Ptr aodv = node->GetObject(); if (aodv) { aodv->TraceConnectWithoutContext(“TxRreq”, MakeCallback(&AodvRreqTxTrace)); // Connect to other relevant trace sources (Rrep, Rerr, etc.) } } ```
   * **En run\_simulations.sh:**
     + Asegura que los nuevos datos de overhead se añadan al metrics.csv existente o a un nuevo archivo de log (routing\_overhead.csv). Si es un nuevo archivo, asegúrate de moverlo a la carpeta metrics/ de cada corrida.
2. **Instrumentación para Tiempo de Convergencia:**
   * **En simulacioniot.cc:**
     + **Eventos de Disrupción:** Programa eventos en la simulación para simular cambios de topología. Esto podría ser:
       - Apagar un nodo aleatorio después de X segundos (Simulator::Schedule(Seconds(X), &DisableNode, nodeToDisable);).
       - Añadir un nuevo nodo dinámicamente.
       - Mover un nodo existente a una nueva ubicación para forzar cambios de ruta.
     + **Monitoreo de Tabla de Ruteo:** Para medir la convergencia, necesitas acceso al estado de la tabla de ruteo de los nodos. NS-3 a menudo expone trace sources para cambios en la tabla de ruteo.
       - Conéctate a estos trace sources (ej. ns3::aodv::RoutingProtocol::RouteAdded, RouteRemoved, RouteUpdated).
       - Registra el simTime cuando se produce un cambio significativo en las rutas después del evento de disrupción.
       - El “tiempo de convergencia” se puede definir como el tiempo desde el evento de disrupción hasta que un porcentaje de nodos (ej. 90-95%) ha estabilizado sus rutas hacia un destino de interés, o hasta que no se detectan más cambios en la tabla de ruteo para un período determinado.
     + **Función de Registro de Convergencia:** Crea una función que se llame periódicamente después del evento de disrupción para verificar la estabilidad de las tablas de ruteo y registrar el tiempo de convergencia.
   * **En run\_simulations.sh:**
     + No se requieren cambios mayores, ya que la lógica de tiempo de convergencia estará dentro del C++ y los resultados se añadirán a los archivos de métricas.

**Archivos de Salida (Modificados/Nuevos):**

* metrics/metrics.csv: Deberá incluir nuevas columnas como overhead\_ruteo\_total\_bytes, overhead\_ruteo\_total\_paquetes, overhead\_por\_paquete\_datos, tiempo\_convergencia\_segundos. (Se recomienda agregar el protocolo al nombre de la métrica, ej. aodv\_overhead\_bytes).
* Opcional: routing\_table\_changes.csv (para un análisis más granular de la convergencia).

## MOD3: Robustez y Resiliencia (Fallo de Nodos/Enlaces)

**Objetivo:** Evaluar la capacidad de los protocolos para mantener el rendimiento y la conectividad cuando hay fallos en la red.

**Impacto en “simulacioniot.cc”:** Alto. Se añadirán funciones para simular fallos y se adaptará la recolección de métricas.

**Impacto en “run\_simulations.sh”:** Medio. Nuevos argumentos para controlar los fallos y posiblemente nuevas subcarpetas para organizar estos escenarios.

**Acciones Detalladas:**

1. **Implementación de Fallos:**
   * **En simulacioniot.cc:**
     + **Función de Fallo de Nodo:** Crea una función DisableNode(Ptr<Node> node) que apague un nodo (ej. desactive su dispositivo Wi-Fi, detenga sus aplicaciones).
     + **Función de Fallo de Enlace (Opcional, más complejo):** Podrías simular la degradación de un enlace o la desconexión entre un par de nodos específicos, aunque apagar nodos suele ser suficiente para evaluar el impacto en el ruteo.
     + **Programación de Fallos:**
       - Introduce nuevos argumentos de línea de comandos para controlar la simulación de fallos (ej., --numFaultyNodes, --faultStartTime, --faultDuration, --faultPattern).
       - Programa Simulator::Schedule para llamar a DisableNode en nodos aleatorios o específicos en momentos determinados.
       - Considera patrones de fallo: un solo fallo, fallos múltiples simultáneos, fallos en secuencia, o fallos permanentes vs. intermitentes.
   * **Recolección de Métricas de Robustez:**
     + Durante el periodo de fallo y después, monitorea PDR, Throughput y Latencia. Observa la caída en el PDR y el tiempo que tarda en recuperarse.
     + **Conectividad de la Red:** Implementa una forma de verificar la conectividad global o la conectividad entre pares de nodos clave después de un fallo. Puedes usar ns3::Ipv4RoutingHelper::GetRoutingTable() (si está disponible y tiene el detalle necesario) o enviar pings periódicos entre nodos y registrar sus respuestas.
2. **Automatización de Escenarios de Fallo:**
   * **En run\_simulations.sh:**
     + Añade un nuevo bucle para iterar sobre diferentes escenarios de fallos (ej. FAULT\_SCENARIOS=("no\_fault" "single\_node\_fail" "multiple\_nodes\_fail")).
     + Pasa los argumentos de fallo relevantes a simulacioniot.cc.
     + Considera crear subcarpetas adicionales en la estructura de resultados, como SIMULATION\_DIR/CONFIG\_NAME/PROTOCOL/FAULT\_SCENARIO/runX/.

**Archivos de Salida (Modificados/Nuevos):**

* metrics/metrics.csv: Nuevas columnas o filas para reflejar el PDR\_con\_fallo, throughput\_durante\_fallo, tiempo\_recuperacion\_segundos, conectividad\_post\_fallo.
* Opcional: fault\_log.csv que registre cuándo y qué nodos fallaron.

## MOD4: Consumo de Energía Detallado

**Objetivo:** Obtener un análisis más granular del consumo de energía, desglosado por estados de la radio y componentes.

**Impacto en “simulacioniot.cc”:** Medio. Requiere la configuración detallada del módulo de energía.

**Impacto en “run\_simulations.sh”:** Bajo. Asegurar la recolección y movimiento de los nuevos archivos de log.

**Acciones Detalladas:**

1. **Configuración Detallada del Módulo de Energía:**
   * **En simulacioniot.cc:**
     + Asegúrate de que ns3/energy-module.h esté incluido.
     + En lugar de solo BasicEnergySource, configura un WifiRadioEnergyModel para cada dispositivo Wi-Fi.
     + Configura los parámetros del modelo de radio (ej. TxCurrent, RxCurrent, IdleCurrent, SleepCurrent en miliamperios) basándose en hojas de datos de dispositivos IoT reales si es posible, o valores comunes en la literatura.
     + Conecta “trace sources” específicos del WifiRadioEnergyModel (ej. EnergyConsumedTx, EnergyConsumedRx, EnergyConsumedIdle) para registrar el consumo en cada estado.
     + **Función de Registro Periódico:** La función LogEnergyConsumption ya existente puede ser mejorada para registrar estos detalles periódicamente.
   * **Calculo de Métricas:**
     + Al final de la simulación, calcula el consumo total de energía por nodo, y también el consumo promedio en cada estado (Tx, Rx, Idle, Sleep).

**Archivos de Salida (Modificados/Nuevos):**

* energy\_consumption.csv: Columnas adicionales como tx\_energy\_mJ, rx\_energy\_mJ, idle\_energy\_mJ, sleep\_energy\_mJ, total\_energy\_mJ por cada nodo a lo largo del tiempo.
* metrics/node\_metrics.csv: Nuevas columnas como avg\_tx\_power\_consumed, avg\_rx\_power\_consumed para resumir el consumo promedio por nodo.

## Próximos MODs (Breve Resumen):

* **MOD5: Protocolos Adicionales (RPL):** Introducir el protocolo RPL en las simulaciones y comparar su rendimiento con los protocolos ad-hoc existentes utilizando todas las métricas recolectadas hasta el MOD4.
* **MOD6: Escenarios Más Complejos:** Implementar modelos de movilidad más realistas (ej. Manhattan Grid, Group Mobility) y topologías de red no uniformes (con agujeros, clústeres de nodos).
* **MOD7: Integración con SDN (Inicial):** Comenzar a explorar el módulo SDN de NS-3, diseñando un escenario simple donde un controlador SDN básico tome algunas decisiones de ruteo, comparando con el ruteo tradicional.
* **MOD8: Ataques Avanzados:** Implementar patrones de ataques de seguridad más sofisticados (blackhole, wormhole) para evaluar la robustez y la capacidad de detección de los protocolos de ruteo.
* **MOD9: Calidad de Servicio (QoS):** Introducir diferentes clases de tráfico con requisitos de QoS y evaluar cómo los protocolos de ruteo manejan estas prioridades.
* **MOD10 y Siguientes: Redes Neuronales y Optimización:** Esta etapa más avanzada implicaría integrar la lógica de redes neuronales para la toma de decisiones de ruteo, o para el análisis de datos de simulación para identificar patrones de rendimiento. Esto podría requerir la exportación de datos a herramientas externas (Python con TensorFlow/PyTorch) o la integración de bibliotecas de ML en NS-3 (más complejo).

**Consideraciones Generales para cada MOD:**

* **Documentación:** Después de cada MOD, actualiza tu documentación (el archivo Markdown que ya estamos construyendo) con los detalles de la implementación, las nuevas métricas y cómo interpretar los resultados.
* **Control de Versiones:** Utiliza un sistema de control de versiones (Git es lo ideal) para gestionar los cambios en simulacioniot.cc y run\_simulations.sh. Esto te permitirá volver a versiones anteriores y colaborar si trabajas con otros.
* **Validación:** Siempre valida que tus nuevas métricas y configuraciones estén funcionando correctamente. Haz pequeñas pruebas iniciales antes de lanzar corridas completas.
* **Recursos Computacionales:** A medida que los MODs avancen, especialmente con más nodos y tiempos de simulación más largos, los requisitos computacionales aumentarán. Planifica el acceso a recursos adecuados.

Este plan iterativo te permitirá construir tu base de simulaciones de forma metódica, asegurando que cada nueva adición sea bien comprendida y validada antes de pasar a la siguiente etapa.