­­Network Edge Workflow Tool (Newt)

The Network Edge Workflow Tool (Newt) is a Python-based software tool designed to support the creation and orchestration of decentralized network workflows. Newt was initially developed to extend previous work completed on the Workflow ZeroRPC ([WZeroRPC](https://www.nrl.navy.mil/itd/ncs/products/wzerorpc)) software toolkit. WZeroRPC is built around remote procedure call (RPC) concepts and is limited to supporting the design and execution of network workflows using a basic Coordinator-Worker model. The Coordinator-Worker model execution in a network is driven from a centralized coordinator which manages the overall progression of the workflow. The Newt design goes beyond this design in several ways and supports the following additional capabilities:

* Decentralized execution and logic
* Group-based network communications
* Multiple data transport approaches

Newt allows decentralized workflows to be defined and acted upon using local logic decisions at a node within the workflow. Newt is also designed to address network edge workflow models that may span combinations of one-to-one, one-to-many, and many-to-one network communications, and may also require the support of complex distributed decision making (e.g., causal interaction models). This tool is network-transport “agnostic” in its design, in that Newt supports different data transport protocols to facilitate the encapsulation of group-based configurations (e.g., multicast data flows) and to support different levels of reliability and persistence within the workflow hierarchy graph.

# Newt Technical Overview

The following provides a brief technical introduction to Newt; please consult the references listed at the end of this document for more detailed information and examples. In Newt, network-based workflows are created programmatically using a Python language application programming interface (API) and can be serialized to and deserialized from a JavaScript Object Notation (JSON) representation. This means that Newt workflows can be orchestrated in other ways, as long as the editor can generate a valid Newt JSON workflow format. The general use of network-based scientific and business workflow software systems has increased over the past decade and presently addresses a wide range of distributed computing and data dependency needs across a multitude of application domains. However, the state-of-the-art of present workflow systems is primarily based on Directed Acyclic Graph (DAG) designs, which inhibit the use of loop constructs or dynamic data routing decisions based on local information. Newt supports a variety of workflow topologies ranging from typical DAG-based models to fully decentralized models. For applications in more dynamic network environments, such as mobile ad hoc networks (MANET), this architectural flexibility is needed since strict trees and DAGs are often difficult to maintain reliably. A basic architecture overview of Newt is shown below in Figure 1.

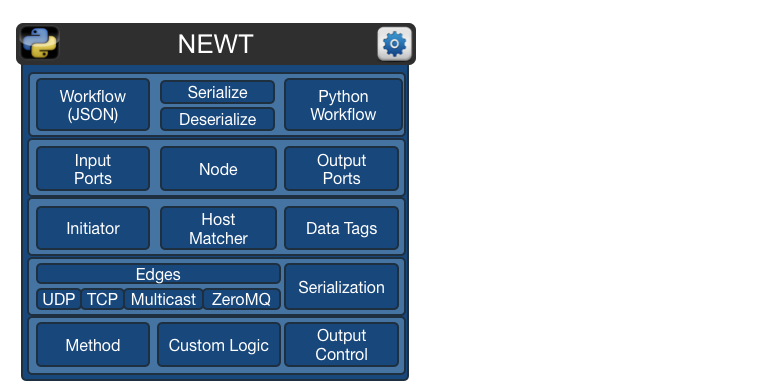


Figure : Newt Basic Architecture Overview

The Newt workflow representation contains a list of nodes and edges in a directed graph structure. Newt nodes represent a named entity in the workflow and the nodes act as containers for adding ports, which connect edges to other ports on other Newt nodes in a workflow communication graph representation. Nodes can also be specified to auto-initiate, allowing certain nodes (initiators) in the workflow to initiate the network workflow at start-up. An output port contains (optionally) default arguments and specifies a bind address and port. An input node has a physical bind address/port and a target process method, which is called when a message is received. Since methods are bound to Newt “input ports,” there can be multiple method invocations per node. A “method” is a Python method process that could be called on a node, and is not restricted to a specific object (like in WZeroRPC), but can live in any Python class module or package and can have any argument structure or data type. Methods can contain both input and output parameters. The input arguments are specified using the input port and the output parameters are the return tuples from a method.

When a workflow runs, each host acts completely autonomously. A special component (host matcher) determines if this particular host is responsible for running a particular workflow node by matching the bind addresses of the input or output ports for each node in the workflow to the local bind addresses on the host. If these match, the host executes that part of the workflow; if not, the host ignores the workflow node. Consequently, to run a workflow, each host needs access to the workflow graph to determine what to process. The workflow graph can be calculated statically (before the workflow is executed) or dynamically loaded on each iteration. However, even in the static case, nodes make decisions upon each received event on how to process the data and which node they will send the processed data to. In this way, the local logic of the methods in the workflow dictate the actual flow of the workflow at runtime. From a Newt target method, a programmer can route data and enable or disable output ports through the use of a context object, which provides the edge information (i.e., Newt node connections). By allowing this level of control, a node can define what links of the downstream workflow to process and when.

Finally, an edge defines the connection between an output port on one node and an input port on another. The edge also defines the network transport that should be used, along with the mechanism for serializing the data. In Newt, transport links are unidirectional by design, but can be orchestrated to implement logical bidirectional links by using the workflow map.

# Related References

For further Newt technical details and examples please see:

1. Macker J. P., Taylor I. 2017. [Orchestration and Analysis of Decentralized Workflows within Heterogeneous Networking Infrastructures](https://www.nrl.navy.mil/itd/ncs/node/324). *Future Generation Computer Systems*. Volume 75, October 2017, pages 388-401.
2. Macker J. P., Taylor I. 2015. [Orchestrating Workflows Over Heterogeneous Networking Infrastructures: Newt: A Network Edge Workflow Tool](https://www.nrl.navy.mil/itd/ncs/node/320). *WORKS '15: Proceedings of the 10th Workshop on Workflows in Support of Large-Scale Science*. November 2015 Article No.: 2.