Argus: Modern Network Monitoring

Human Centered Tech

Thomas Doylend ◆ *Liam Collins*
humancenteredtech.us

Table of Contents

Overview

Argus is our SNMP-based network monitoring tool designed to support up to approximately 100,000 devices.

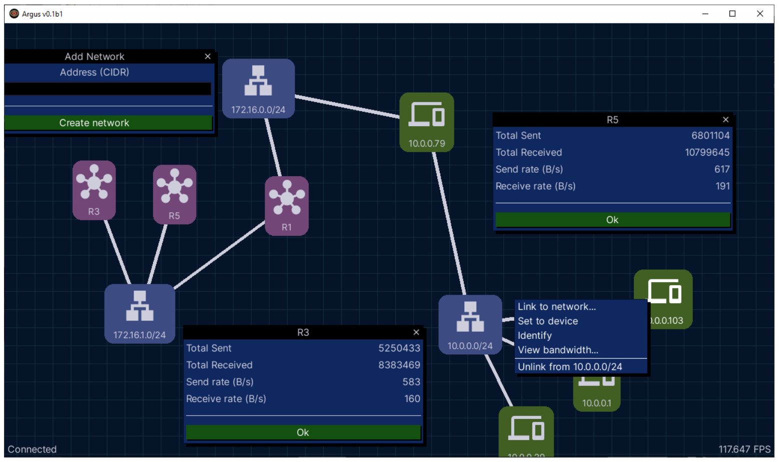
Using custom-built, highly-tuned polling and UI frameworks, **we've been able to build a prototype in one month**, gaining the experience with the problem that will enable us to build the final version in a similarly short amount of time.

Argus already supports dynamic bandwidth monitoring, ping checking, multi-user editing, and fully configurable network maps. In the coming months we will be expanding to the full version with support for several more features.

We are now developing the full version, implementing the features described below and optimizing the framework to support 100k devices.

Argus succeeds where existing solutions fail. The tools currently on the market fall into one of three categories: (a) expensive, (b) ungainly, or (c) unusable on networks larger than a few thousand devices. Argus is the smart affordable solution for networks at scale.

We are opening an investment opportunity to companies interested in using Argus and directing the course of its future development, as well as to independent investors, and also looking for partners in bringing this tool to market.



R5

Total Sent	6801104
Total Received	10799645
Send rate (B/s)	617
Receive rate (B/s)	191

R3

Total Sent	5250433
Total Received	8383469
Send rate (B/s)	583
Receive rate (B/s)	160

Features

Server-side Performance. After optimization, Argus will monitor up to 100k devices at up to 1 poll per second per device. Subnets are viewable from a single client without lag or putting unnecessary load on the network. To achieve this we are using split servers, as described in *Technical Details* below.

Custom SNMP templates. Nodes will have a configurable text display, in which the user can enter SNMP OIDs; these templates are queried from the device which the node represents, and replaced with the returned values.

Submaps. Because a network may contain an overwhelming number of devices, we use a two-level display hierarchy. At the top level, many devices are collapsed into single elements called “submaps”, each representing a family of nearby devices. Double-clicking on one of these nodes will expand the submap and show the device layout and internal connections of that map. These submaps are user-configurable, and can be displayed in either a network view or a searchable list view.

Node and Submap Status. Nodes, submaps, and links track their own statuses. A node or link is in an alert state if user-configurable conditions (e.g. excessive traffic on a link, loss of ping response from a node, etc) are met; a submap is in an alert state if any of its nodes or links are. These updates are displayed dynamically, making it possible for operators to rapidly track down existing problems in their network, anticipate upcoming problems, and manage downstream effects of overloads and outages.

Multi-user Editing. Multiple users can edit the map at once and see one another’s changes in real time. In addition we plan to implement highlighting and note-taking tools, to make it easier for operators to collaborate in virtual meetings and while out in the field.

Client-side Performance. We are targeting the following minimum client hardware and expect to achieve a consistent 60FPS or better in the client:

OS: Windows 10 or later, Debian 9 or later and its derivatives (e.g. Ubuntu, Mint)
Processor: Intel Core i5
Memory: 16GB RAM
Graphics: Intel Integrated Graphics

Timeline

The prototype took us one man-month of work. As we have now learned how to design and structure the tool, the next iterations will be completed more quickly.

We expect to have a full Early Access version of the features listed above within two months. From there, thorough optimization, bugfixing, and testing will be required before releasing the full commercial version. We’re aiming for a total of three to six months of development time to give us a sufficient window to incorporate those fixes and anticipate any delays. This will also give us an opportunity to incorporate features requested by our early-supporter clients before release.

Pricing

We plan to offer the tool on a subscription model for around $1/device per month, with a free tier for small network operators and a more expensive tier which includes support.

Technical Details

Our prototype is written in Python. Python’s performance, while not as great as a low-level language, is more than enough for both client and server in our testing; in addition, it is possible to rewrite hot-path code in C and integrate them using Cython or ctypes.

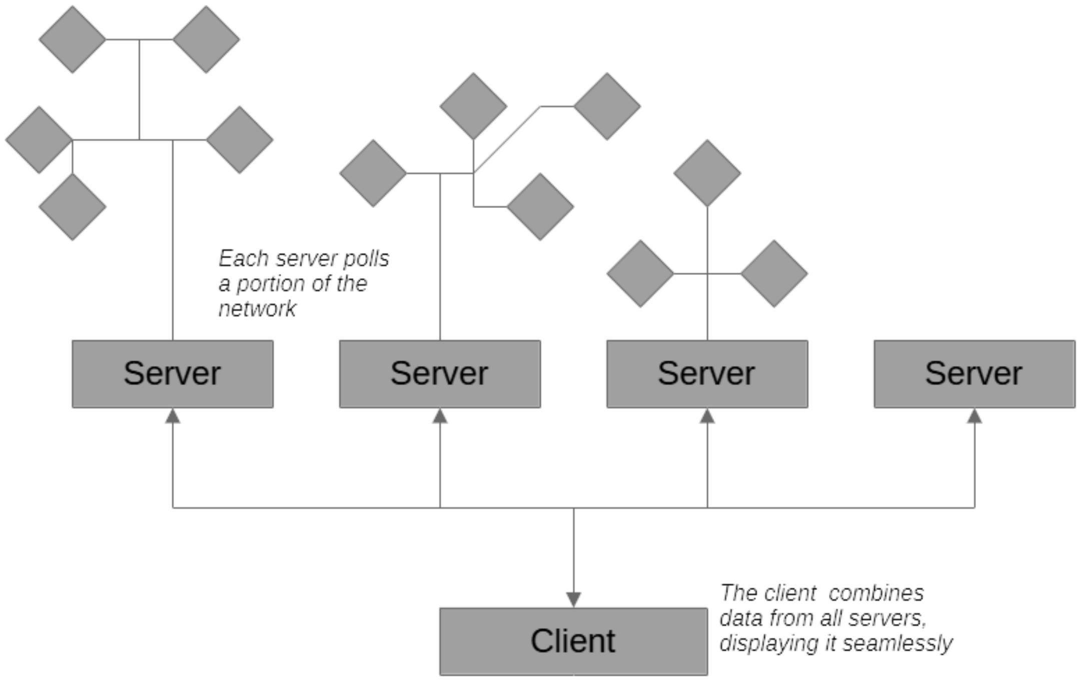
A major advantage of Python is the speed of development; we’ve been able to ship entire features in just hours thanks to the rapid turnaround Python and our custom libraries make possible. Our UI is in-house code, which makes it easy to design custom GUI features without being constrained by what primitives are available in a toolkit.

Our model uses a specialized database as the main backing store for the network information. Polling daemons make the SNMP requests of the devices on the network and report the results into the database, while the client reads from the database and displays it to the user. As the database also sends updates to subscribed clients when it receives them, data can be updated instantly in the client display as it is received from the server.

Our prototype was using PySNMP to perform the requests; we found it usable but unsatisfactory in a few respects. For the full version we will implement a custom event-based SNMP decoder, which will be more efficient. For even greater speed it can be rewritten in C very quickly.

In a network with 100,000 devices, each being polled once per second, the total volume of SNMP data could easily exceed 100Mbps. Even with gigabit links feeding the server, decoding this much network data might strain one device, so to solve this we will enable the network to “distribute” the handling of submaps across multiple polling servers.

However, from the client’s point of view, these will all integrate seamlessly and act as one device. This also opens the possibility of deploying the polling servers at various points around the network, thereby decreasing the distance that large-volume SNMP data has to travel (compared to the smaller volume of client display data). This should prove advantageous in congested low-volume networks.


Each server polls a portion of the network
Server
Server
Server
Server
Client
The client combines data from all servers, displaying it seamlessly

Operational diagram of the client/server architecture.

The database uses a publish/subscribe model, similar to MQTT or other “mesh” event protocols but with additional state storage. We can therefore distribute the database over several servers with relative ease, while still having the model cohere to a consistent state without errors.

This is also how efficient multi-user editing is accomplished. By treating each client device and each polling server both as database clients, we get two features for free. This design also makes it straighforward to implement advanced permissions management, push notifications in case of off-hours emergencies, and other features that would be difficult to add in any other model.