

# Porting and Packaging Seattle Testbed for OpenWrt

Raymond Fok, Albert Rafetseder, Justin Cappos

## Abstract

In this technological era nearly every home is equipped with Internet access, usually forwarded through a wireless router. Though it wields less processing power than a desktop computer or even a smartphone, routers can be customized with Linux system functionalities like an SSH server, VPN, and network monitoring tools. This project aims to deploy Seattle Testbed on wireless routers running OpenWrt by modifying an existing Linux distribution and using Seattle’s Custom Installer Builder to package its base installers into a tarball for shipment alongside other platforms, with its dependencies managed separately.

## Introduction

Seattle Testbed is an open-source, peer-to-peer platform designed to enhance networking and distributed systems research. Established on the resource donation of users and institutions worldwide, Seattle’s global distribution makes it ideal for applications in cloud computing, networking, and ubiquitous computing. Users can install Seattle for code experimentation in a safe and contained sandboxed environment, limiting the consumption of resources such as CPU, memory, storage space, and network bandwidth, and ensuring that the tested programs are isolated from other files or programs. These characteristics allow distributed code testing without compromising the machine’s performance or security.

Wireless routers represent the most important piece of hardware connecting users and the global network of the Internet. OpenWrt gives users the flexibility of customizing the software on their routers via a writable filesystem, package management system, cross-compilation tool-chain, and minimal Linux shell (ash). Currently, this project deploys Seattle on a TP-LINK TL-WDR3600 router running the latest version of OpenWrt (Chaos Calmer 15.05.1).

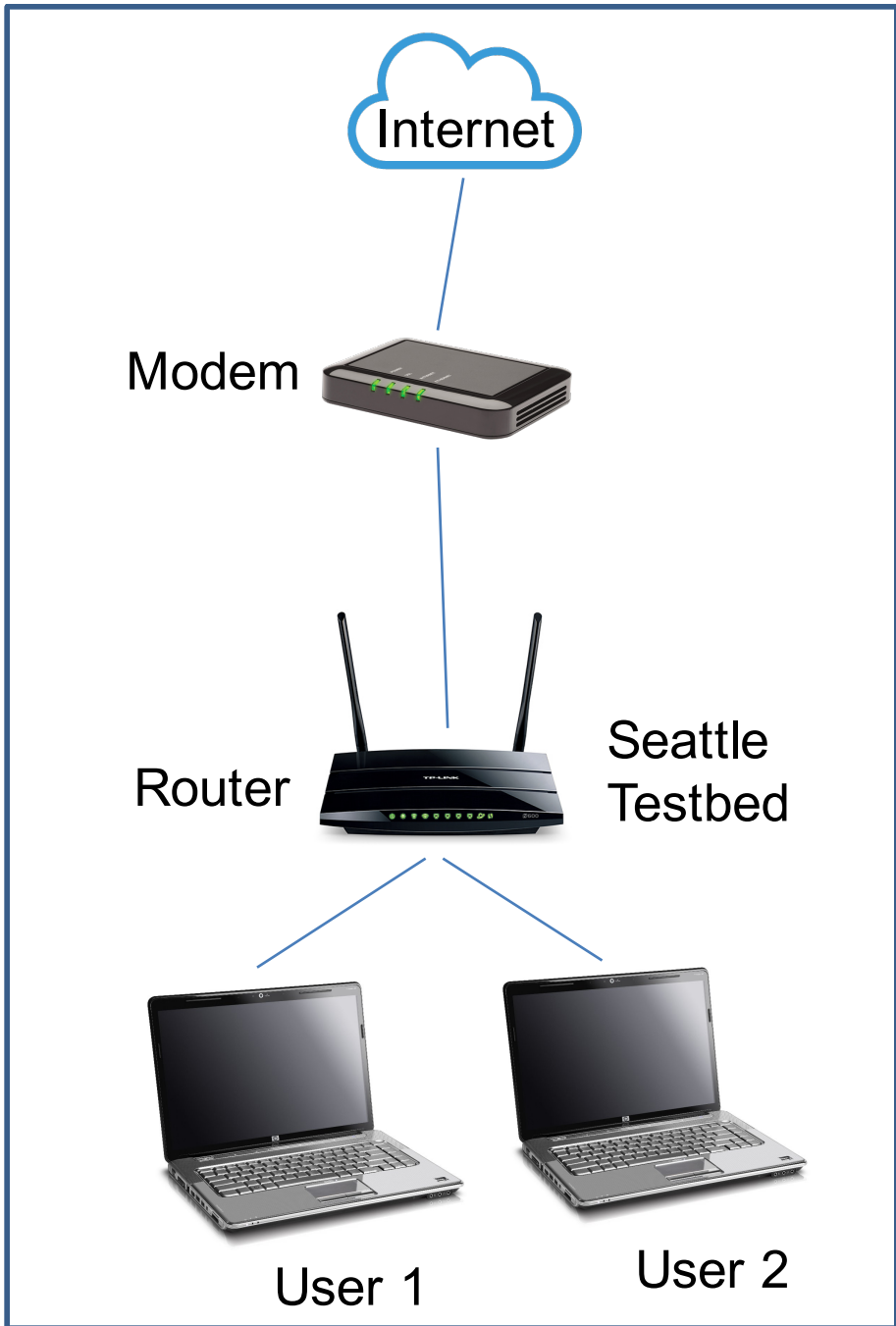
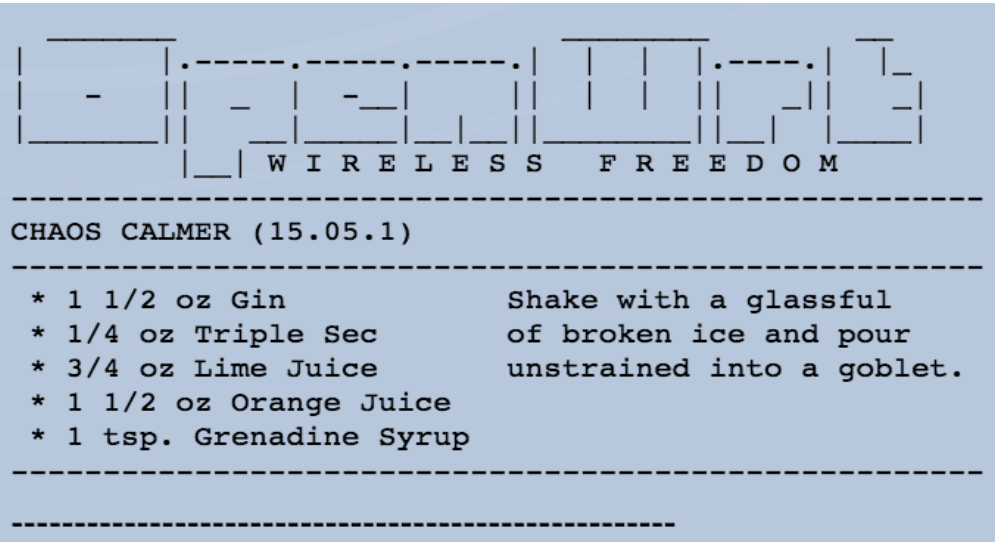


Figure 1: A depiction of Seattle Testbed deployment on a wireless router.

## Deployment

A previous deployment option<sup>1</sup> involved generating an IPK for Seattle files, and installing onto the router via OpenWrt’s built-in Opkg Package Manager<sup>2</sup>. Seattle’s dependencies and install location could then be defined in the Makefile and automatically configured at installation.

```
cd ~
wget https://downloads.openwrt.org/chaos_calmer/15.05.1/ar71xx/generic/OpenWrt-SDK-15.05.1-ar71xx-
tar -xvf OpenWrt-SDK-15.05.1-ar71xx-generic_gcc-4.8-linaro_uClibc-0.9.33.2.Linux-x86_64.tar.bz2
cd OpenWrt-SDK-15.05.1-ar71xx-generic_gcc-4.8-linaro_uClibc-0.9.33.2.Linux-x86_64/package
git clone https://github.com/rf1591/Seattle-OpenWrt
mv Seattle-OpenWrt seattle
cd ..
make clean && make world
scp seattle_1.0-1_ar71xx.ipk root@192.168.1.1:/tmp
opkg install /tmp/seattle_1.0-1_ar71xx.ipk
```

Figure 2: Commands issued to create and install a Seattle IPK using OpenWrt’s SDK.

This alternate approach directly constructs and downloads a tarball to the user’s computer. One advantage is that the installation instructions are simpler for the end-user because they do not have to go through the process of OpenWrt’s custom packaging, which updates frequently. A drawback of this method is that users are assumed to either have the necessary dependency packages installed already on their routers, or have the additional flash storage required for a post-installation.


- Download the Seattle tarball.
- Copy the tarball to OpenWrt with `scp seattle_openwrt.tar.gz root@192.168.1.1:/`
- Ssh into the router and extract the tarball with `tar -xvzf seattle_openwrt.tar.gz`
- Install or update the OpenWrt packages required for the Seattle installation.
 

```
opkg install python-light python-compiler python-ctypes python-logging python-openssl
```
- Navigate to the directory named `seattle` and run `./install.sh`
- To check that Seattle is running, try running: `ps | grep nmmain.py | grep -v grep`

Figure 3: Steps to download Seattle Testbed from Seattle’s Custom Installer Builder. Note additional packages (like `git-http`) are recommended for unit testing

## Challenges

- Limited storage resources on router after installing a light-weight Python framework and other necessary dependencies for unit testing
- OpenWrt is a minimal Linux-based operating system which is great for small-capacity embedded devices like routers, but lacks the traditional functionality of `bash`, `git`, and `wget`



Architecture	MIPS 74Kc
System	AR9344 (MIPS)
CPU	560 MHz
Flash Storage	8 MB
RAM	128 MB

Figure 4: Specifications of a TP-LINK TL-WDR 3600 wireless router.

## Results

Since OpenWrt closely resembles a classic Linux kernel, Seattle’s Linux distribution could be used as a framework for deploying onto OpenWrt. Several minor tweaks were made to adapt for the minimized operating system functionality typically provided on embedded devices.

- Increased timeout for code safety evaluation to compensate for router’s slower processing speed
- Addressed a GETTID syscall issue<sup>3</sup> by hardcoding in the syscall value for retrieving the thread id on a Linux MIPS 32-bit system
- Replaced non-portable shell script commands from Seattle’s startup and shutdown files with OpenWrt compatible calls

Advantages of this deployment are that users can avoid the hassle of building their own IPK files and follow a traditional installation of Seattle Testbed similar to other platforms. However, the responsibility of dependency installation and maintenance then shifts onto the user. During unit testing, several parts of an established unit testing framework for Seattle had to be updated and adapted for OpenWrt. Additionally, problems arose during Seattle’s resource allocation phase because portions of the router’s already limited computing power were donated. It became hard to judge if the router was capable of running Seattle and actually donating resources because it timed out of several unit tests due to insufficient processing power.

## Future Work

Although the foundation for deploying Seattle Testbed on OpenWrt has been built through this project and the work of several other researchers, it remains to be seen what impactful measurements can be gathered from testing with this relatively new platform. As computational and storage resources are still a critical limitation of embedded devices, improvements to testing and resource allocation should be expected as processing speed and flash storage space increase.

- Install Seattle Testbed on multiple routers in locations around the world to monitor aspects such as network traffic statistics, packet transfer rates, and signal strength
- Improve unit testing and optimize resource-heavy operations

### References

- Xuefeng Huang. *An Experimental Platform for Home Wireless Routers*. 2016.
- OPKG Package Manager. <https://wiki.openwrt.org/doc/techref/opkg>
- Linux ‘GETTID’. [https://github.com/SeattleTestbed/rep\\_y\\_v2/issues/98](https://github.com/SeattleTestbed/rep_y_v2/issues/98)