Contiki Notes

<https://www.youtube.com/watch?v=6xJ1zbTVOnM&t=39s>

Interview with Adam dunkels - founder of Contiki.

Video description:

*Contiki connects low-power microcontrollers to the internet and supports standards like IPv6, 6lowpan, RPL and CoAP. Other key features include highly efficient memory allocation, full IP networking, very low power consumption, dynamic module loading and more. Supported hardware platforms include Redwire Econotags, Zolertia z1 motes, ST Microelectronics development kits and Texas Instruments chips and boards.*

**GENERAL**

* Contiki is an operating system for internet of things devices.
* For wireless microprocessors
* Makes it possible for devices to connect to the internet.
* Uses a condensed version of the TCP/IP protocol stack.
* Uses IPv6 - of course for the HUGE number of IOT devices in existence
* IOT is an extension to the internet - and so will use the internet protocol (IPv6).
* MQTT has been added in.
* Contiki runs on a large number of different platforms and microprocessor types.
* Low end to high end such as ARM Cortex. Ported to many platforms. Worth checking security against each other - are the implementations consistently secure? How do the implementations differ?
* Prides itself on portability.

**TRANSPORT LAYER**

* Internet protocols themselves - extended to IOT for wireless application.
* Extend networks by added new nodes
* Mesh networking
* UDP and TCP support, CO-AP, HTTP, Web sockets.
* “An extension of the Internet”.
* Mesh network standards - “Ripple” - IETF standard protocol for mesh networking IPv6.
  + We might have a lossy environment - so the protocol can deal with this by adapting how to move packets through the network.
  + Ripple protocol works well here.
  + Regression test suite used with Contiki - every time code is changed in the codebase.
  + Simulations are run to test that the mesh is working.
  + Very hard to test in the real world, so relies on the simulation.

**DISTANCE AND DENSITY**

* Depends on what radio you are using
* You can easily have hundreds / thousands of nodes.
* In larger numbers, performance decreases.
* For larger numbers, they have a data collection scheme - for those networks. Rarely collect data from all the nodes, and more often from some of the nodes.
* Moving to larger networks / long range radios which cover several KM - e.g. Streetlight network.
* Density of a metropolitan area.

**WIRELESS COMMUNICATIONS PROTOCOLS AND STANDARDS**

* Connect the world to you in fascinating ways.

**DEVELOPER COMMUNITY WITHIN CONTIKI**

* Hierarchical - small team of dev merging patches - 10 people
* 50 or 60 actively contributing code back to the codebase.
* Moving steadily
* Comparable OS - embedded systems. (RIOT) - very similar to CONTIKI as to how RIPPLE is used.
* TinyOS - focused on the wireless part.
* Systems like Linux are typically aimed at much larger microprocessors.
* Contiki might typically have 100kb of flash ROM and 30/20 kb of RAM.
* Run a full mesh network stack on 20kb of RAM.

**QUALITY CONTROL**

* Contiki is commercially used.
* Every time a change to the code base is made, a testing system is used (regression testing) a simulation system called Cooja.
* This is used to set up a scenario of a wireless network with mesh nodes. They add an application to that, run the application and make sure it still works - 20 or 30 scenarios for Ripple alone. Typically this application will get a node to request data from the internet, making sure the data is received into the simulator network.
* Tests that compile the system for a large number of platforms. 8-9 different platforms, 4-5 different CPU architectures.
* Uncovered a great number of bugs. So, how good is this simulation system?

**NEW FEATURES**

* Contiki 3.0
* Connectivity from the mesh network to the internet (IPv6/4) through proxies and firewalls.
* Test port 80, HTTP protocols etc.
* Add proxy support.
* System on a chip platforms - 2.4 GHz support + BTLE.
* BTLE and mesh network connected to the internet.
* Device discovery from a smart phone
* Bluetooth beacons etc.
* Push towards new platforms, internet connectivity.

<https://www.youtube.com/watch?v=xE0zUwclgAo>

“What is Contiki OS and why is uIP special?”

**Features**

* Designed specifically to work with IoT
* Devices which are low power, constrained devices - microcontrollers, wireless sensors
* Highly portable, multi tasking
* New features - MicroIP (uIP) and 6LoWPAN.

**uIP**

* Micro IP
* Implements a skeleton TCP/IP
* Designed for 8 and 16 bit micro controllers
* Stack includes TCP, UDP, ICMP protocols.
* Size of uIP is a few Kilobytes, ram requirements are a few hundred bytes.
* uIPv6 - the world’s smallest certified IPv6 stack - designed by CISCO for sensors and actuators.

**6LoWPAN**

* IPv6 - low power and wireless personal area network.
* Enables the use of IPv6 over the MAC and PHY layer of 802.15.4 (LR-WPAN).
* Challenges - each MAC layer has a limitation for data e.g. MTU - allowing in this case only 127 bytes.
* Networking layer - the IPv6 MTU was 280 bytes.

**Transition from IPv4 to IPv6**

* IPv4 is a 32 bit IP address - 2^32 = 4 billion or so available addresses, which is soon to run out.
* IPv6 is 128 bit IP addresses - which is considerably more.
* How does this transition? Slowly, and gradually.
* Enabled from three popular technologies: Dual stack, tunnelling and header translation.

**Dual Stack**

* TCP/IP protocol stack where we contain bother IPv4 and IPv6 parts of the protocol in the stack.
* If we need to communicate with an IPv4 node, use that part of the stack. Or with IPv6 address, use that part of the stack.
* Source queries the DNS - the DNS responds back with IPv4 or IPv6 address - based on which one, the source may respond with an IPv4 packet or an IPv6 packet which it sends to the destination node.

**Tunnelling**

* Implemented when different IP versions exist on path.
* E.G. if two IPv6 machines communicate through IPv4 routers, the IPv6 packets are placed inside IPv4 packets and then forwarded towards the destination machine.

**Header Translation**

* Necessary when most of the Internet has moved to IPv6, but some systems still use IPv4.
* Sender uses IPv6, receiver uses IPv4. Tunnelling doesn’t work here, as destination machine only understands IPv4.
* So the header format is changed on route.
* It is converted from IPv6 to IPv4 on route.
* Done by mapped address.

**RPL**

* Routing protocol for Low power and Lossy networks (RPL)
* A distance vector routing protocol.
* The routing is based on Destination oriented Acyclic graphs or DODAGs.

**Terminology related to RPL**

* Direct acyclic graph (DAG) - a spanning tree graph when we don’t have any cycles
* Root - destination of the nodes in the DAG.
* Up - any edge that is directed towards the root.
* Down - any edge that is directed away from the root.

**Destination Oriented Direct acyclic graph (DODAG)**

* Special DAG where all nodes are trying to reach a single destination.
* Objective function - helps us determine how close we are to the root - e.g. number of intermediary nodes.
* Rank - distance from the root.
* RPL instance - When we have one or more DODAGs, then each DODAG is an instance.
* DODAG ID - Each DODAG has an IPv6 ID (128 bit). This ID is given to the root only. As long as the root doesn’t change, the ID doesn’t change.
* DODAG version - each different shape of a DODAG has a different version. E.g. if nodes sleep or turn off for a couple of weeks, the shapes will change to fill in the gaps.
* Goal - where a DODAG wants to reach.
* Grounded - when a DODAG reaches its goal.
* Floating - when DODAG hasn’t reached its goal.
* Parent - the parent node - the node to which the arrow is pointing
* Child - the child node - the node from which the arrow is extending.
* Sub-DODAG - any subtree of a given DODAG
* Storing nodes - keeps the whole routing table. They know how to go from one node to the other.
* Non storing nodes - only know about their parents.

**RPL Control Messages**

* 5 control messages that form the spanning tree:
* **DODAG Information object (DIO)** - Message sent by a node present in a DODAG. Multicasted from the node, message contains information about whether the node is storing/non storing, floating/grounded.
* **DODAG Information Solicitation (DIS)** - Message sent by a new node when it doesn’t receive announcements - “Is there any DODAG out there”.
* **DODAG Advertisement Object (DAO)** - Request made by a new node to a parent or a root to request to be a part of their DODAG.
* **DAO-ACK** - an acknowledgement - yes or no
* **Consistency check**

