A Django TR-069 ACS Server

The hows and whys of building an ACS server into an existing Django based provisioning and CRM system.

May contain traces of XML.

Thomas Steen Rasmussen

DjangoCPH Day 2018

Shameless

- When not working with TR-069 I have a few projects in my sparetime
- This seems like a good time to tell you about them

CensurFriDNS / UncensoredDNS

- Open, free, uncensored DNS servers
- DNSSEC enabled
- Privacy respecting
- Since 2009
- Anycast servers
- https://blog.censurfridns.dk/
- http://blog.uncensoreddns.org/

BornHack

- Annual 7 day outdoor hacker festival
- Flyers outside
- Several organisers present
- Every August on Bornholm
- https://bornhack.dk/
- https://twitter.com/bornhax/
- Talks, workshops, fun.

Agenda

1) The Problem
   - XML? SOAP? What year is this?! I demand an explanation.
2) Broadband Forum
   - The people responsible for this mess.
3) TR-069, CWMP, ACS and other acronyms
   - A crash course - getting all the acronyms straight.
4) Fitting it all into Django
   - Applying the awesome. The real reason we are here.

1) The Problem, and why XML

ISPs need to configure equipment in a standardised way across hundreds of hardware suppliers and thousands of models.

Creating a standard to encompass all this is a **huge** undertaking, and requires a very strict and thorough approach to standardisation.

SOAP uses XML. So since everything else is XML, so are all the specs, and even the references to other literature and standards.

They even have XML explaining how to XML. Really.

2) Broadband Forum History

- “**The Frame Relay Forum**” (1991)
- “**The ATM Forum**” (1991)
- “**The ADSL Forum**” (1994)
- “**DSL Forum**” (1999) (formerly “ADSL Forum”)
- “**MPLS Forum**” (2000)
- “**MPLS and Frame Relay Alliance**” (2003) (merged “Frame Relay Forum” with “MPLS Forum”)
- “**MFA Forum**” (2005) (merged “ATM Forum” with “MPLS and Frame Relay Alliance”)
- “**IP/MPLS Forum**” (2007) (formerly “MFA Forum”)
- “**Broadband Forum**” (2008) (formerly “DSL Forum”)
- “**Broadband Forum**” (2009) (assimilated “IP/MPLS Forum”)

3) TR-069

TR-069 is the primary document. It was first released in 2004. XML and SOAP to the rescue!

TR-069 means a lot of things. RPCs, data models, and the **CWMP** protocol are defined in TR-069.

But it can also mean all the related technical reports. It can be a bit overwhelming.

Devices can get **TR-069** certified (though not many do):

https://www.broadband-forum.org/implementation/interop-certification/tr-069-certified-products

(16 total products certified!)

TR-069 continued

- **TR-069** defines the **CWMP** protocol and the **InternetGatewayDevice** root data model. First version is from 2004. Latest version is amendment 5 from November 2013.
- **CWMP** is the **CPE WAN Management Protocol** currently at version 1.4.
- **CWMP** defines: Terminology, Protocols, Encodings, Authentication, ACS discovery, Connection establishment, RPC methods, and more.
- **TR-069** been extended and improved many, many times. This happens in new **Technical Reports**.
- The next couple of slides show the important documents in play.

Some Technical Reports

Report	Name	Latest
TR-001	ADSL Forum System Reference Model	May 1996
TR-069a5	CPE WAN Management Protocol	November 2013
TR-098a2c1	Internet Gateway Device Data Model for TR-069	December 2014
TR-106a7	Data Model Template for TR-069-Enabled Devices	September 2013
TR-135a3	Data Model for a TR-069 Enabled STB	November 2012
TR-143a1c1	Enabling Network Throughput Performance Tests and Statistical Monitoring	August 2015
TR-181i2a11	Device Data Model for TR-069	August 2016

https://www.broadband-forum.org/standards-and-software/technical-specifications/technical-reports

Root Data Models

Name	Latest definitions in	Comments
InternetGatewayDevice:1	tr-098-1-8-0-full.xml September 2014	Home internet routers
Device:1	tr-181-1-7-0-full.xml November 2015	Anything but home internet routers
Device:2	tr-181-2-11-0-full.xml July 2016	Unified InternetGatewayDevice:1 and Device:1

Service Data Models

Name	Latest definitions in	Comment
FAPService:2	tr-196-2-1-0-full.xml August 2015	Femto AP
FAPService:1	tr-196-1-1-1-full.xml November 2012	Femto AP
StorageService:1	tr-140-1-3-0-full.xml May 2017	Storage
STBService:1	tr-135-1-4-0-full.xml August 2015	Device
VoiceService:2	tr-104-2-0-0-full.xml March 2014	VOIP Device
VoiceService:1	tr-104-1-0-0-full.xml July 2011	VOIP Device

Selected Schema Files

Name	Latest definitions in	Comments
TR-069 RPCs	cwmp-1-4.xsd November 2013	RPC calls
XMPP Connection Request	cwmp-xmppConnReq-1-0.xsd November 2013	Jabber! :-)
TR-069 Data Model Definition Schema (Data Model Schema)	cwmp-datamodel-1-5.xsd September 2013	Spec for the specs

Support Files

Name	Latest definitions in	Comments
TR-069 Data Model Data Types	tr-106-1-0-0-types.xml November 2013	Data Types
TR-069 Data Model Bibliographic References	tr-069-1-5-0-biblio.xml July 2016	References to RFCs and other external sources

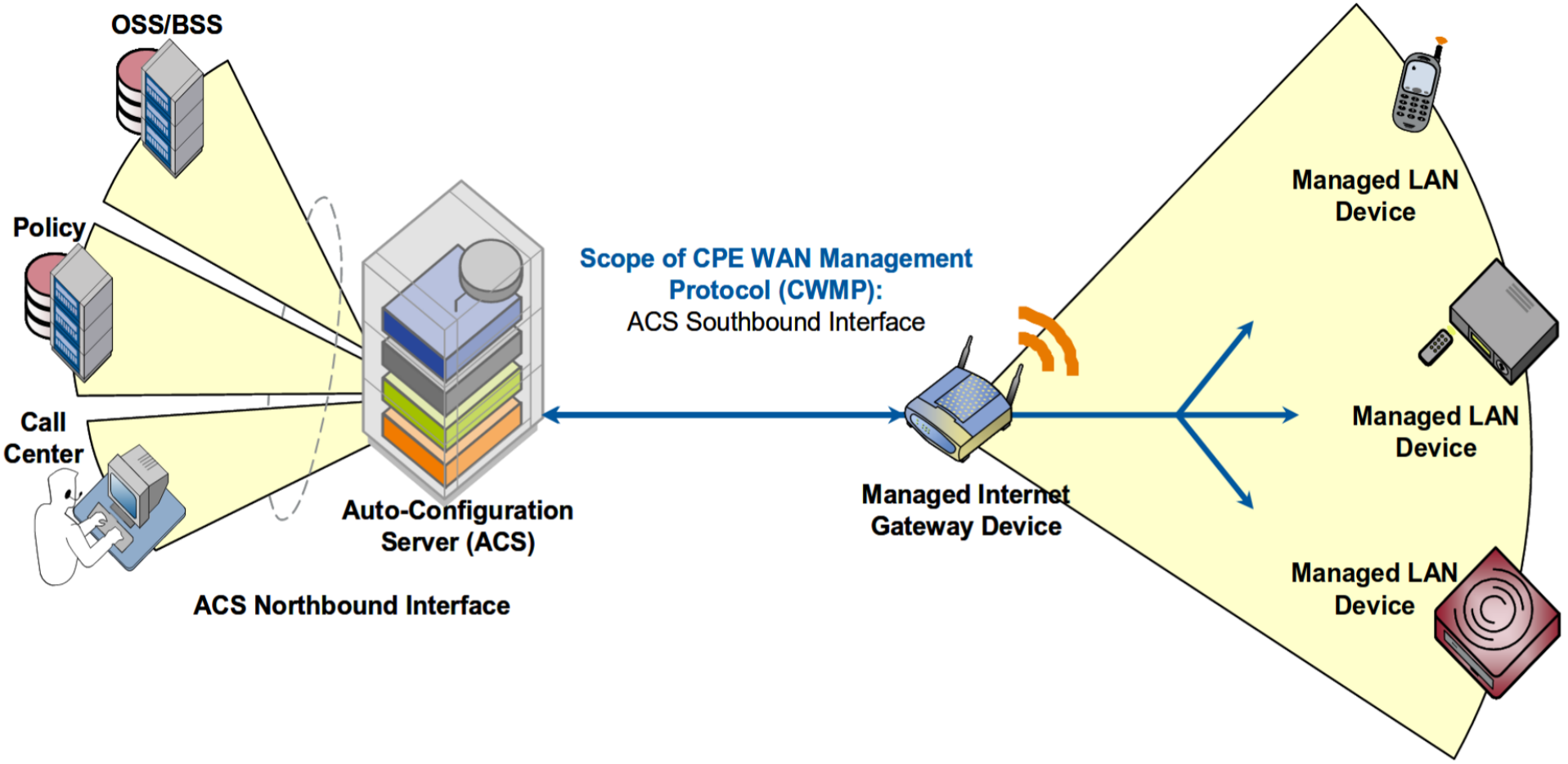
The CPE

- CPE is short for Customer Premises Equipment
- Like many other 3LA it is a bit of an overloaded term
- CPE means any piece of equipment at the customer premises:
  - The internet router
  - Settop boxes
  - VOIP phones
  - Wifi access points
  - Anything really
- CPE gets ACS url over DHCP (usually)

The ACS

- ACS is the Auto Configuration Server
- The CPE uses TR-069 to contact the ACS to get configuration
- This is called a CWMP or ACS session
  - A session is always initiated by the CPE
  - The CPE can be asked to intiate a session though
  - This is called a ConnectionRequest and mostly happens over HTTP or XMPP.
- A diagram showing ACS position in the network follows

Figure 1 – Positioning in the End-to-End Architecture



CWMP

- CWMP defines the protocol
- It specifies that we use HTTP, SOAP, it defines the RPCs and so on.
- It considers encoding, compression, authentication, encryption and more.
- It also defines ConnectionRequests over HTTP and XMPP.
- Illustration of the standard protocol stack follows.

Figure 2 – Protocol stack

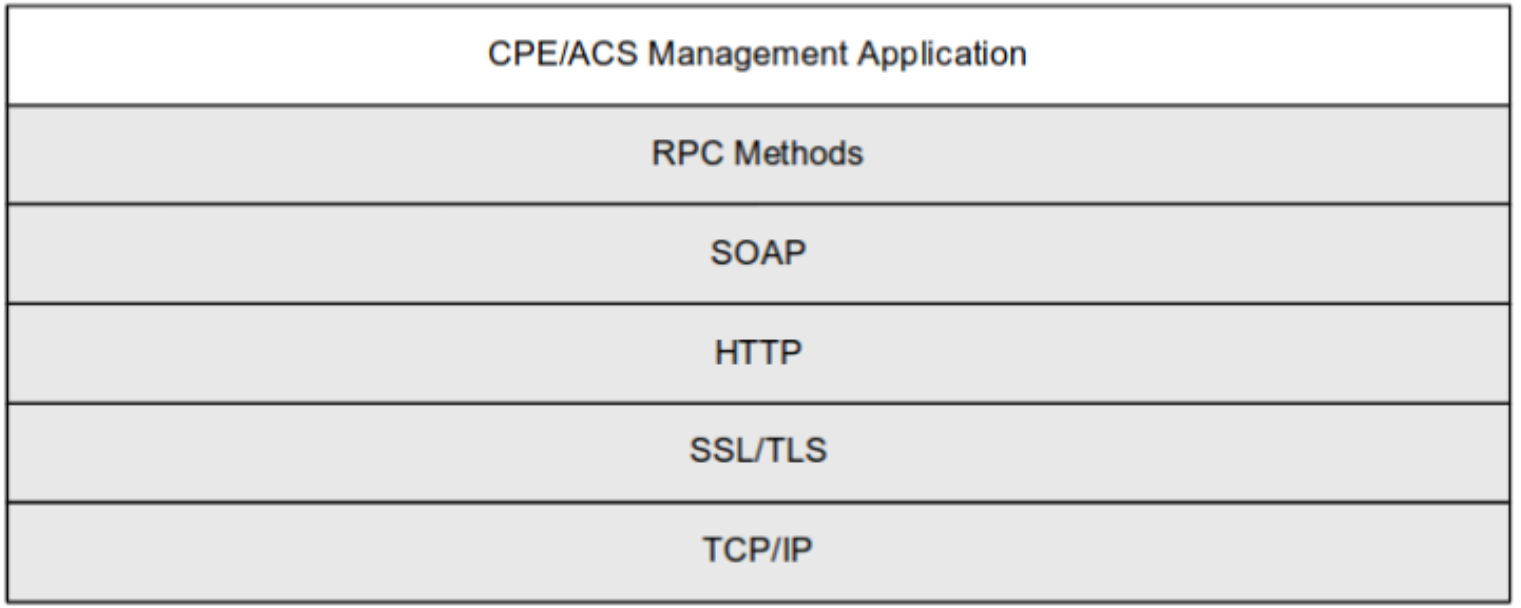
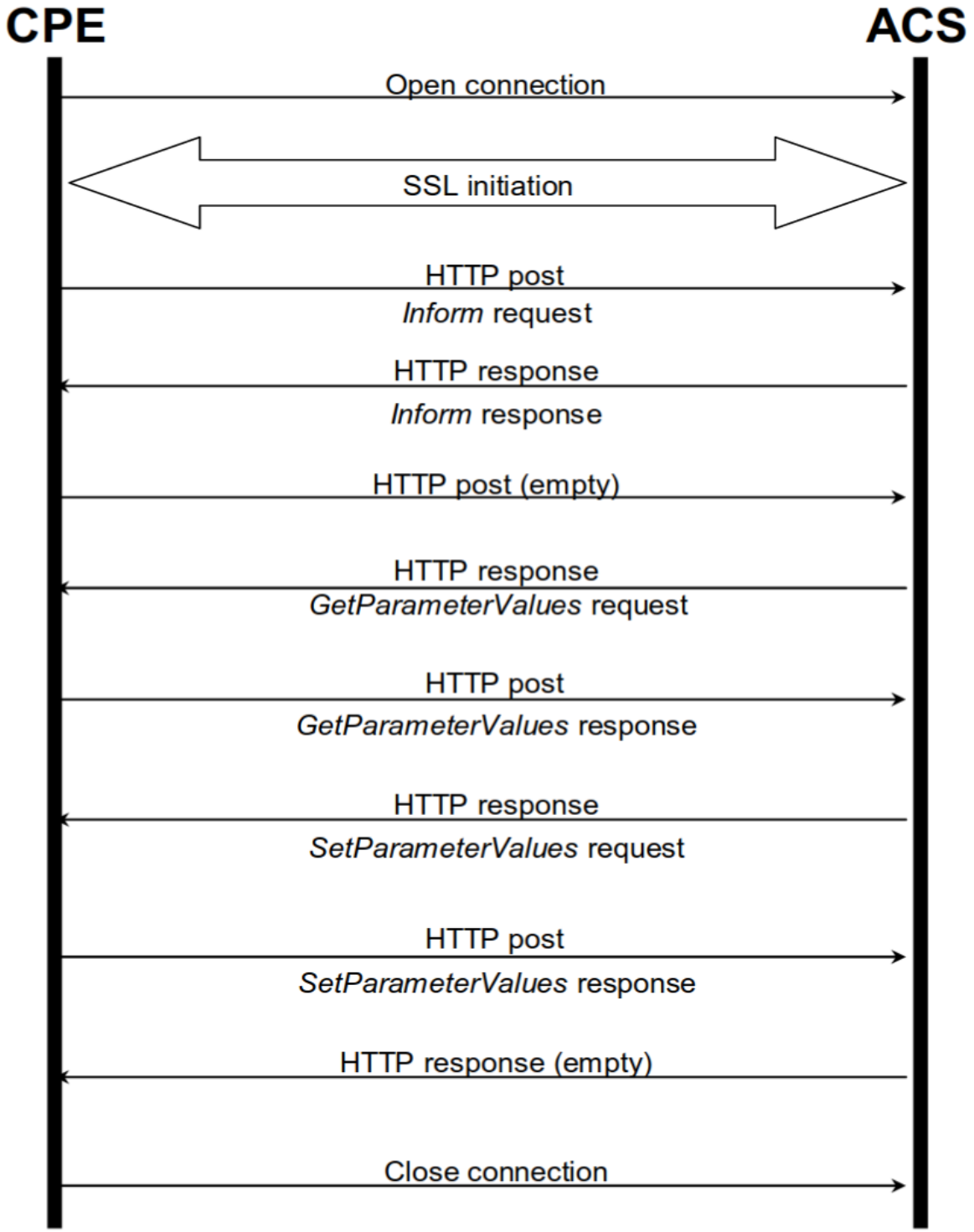


Table 1 – Protocol layer summary

Layer	Description
CPE/ACS Application	The application uses the CPE WAN Management Protocol on the CPE and ACS, respectively. The application is locally defined and not specified as part of the CPE WAN Management Protocol.
RPC Methods	The specific RPC methods that are defined by the CPE WAN Management Protocol. These methods are specified in Annex A.
SOAP	A standard XML-based syntax used here to encode remote procedure calls. Specifically SOAP 1.1, as specified in [9].
HTTP	HTTP 1.1, as specified in [6].
TLS	The standard Internet transport layer security protocol. Specifically, TLS 1.2 (Transport Layer Security) as defined in [11] (or a later version). Note that previous versions of this specification referenced SSL 3.0 and TLS 1.0.
TCP/IP	Standard TCP/IP.

An ACS Session

- The session is 2-way.
- First the CPE calls RPC methods on the ACS.
- Then the ACS calls RPC methods on the CPE.
- When the ACS is done the session is over.


CPE
ACS
Open connection
SSL initiation
HTTP post
Inform request
HTTP response
Inform response
HTTP post (empty)
HTTP response
GetParameterValues request
HTTP post
GetParameterValues response
HTTP response
SetParameterValues request
HTTP post
SetParameterValues response
HTTP response (empty)
Close connection

XML Time

- Time to look at some XML
- This will begin with a quick SOAP and XML primer
- We will look at the first RPC call in every session – the Inform.
- The Inform contains the info needed to identify and communicate with the CPE.
- The CPE considers a session successfully initiated only if it gets an InformResponse RPC reply

SOAP Primer

- SOAP (originally Simple Object Access Protocol) is a protocol specification for exchanging structured information in the implementation of web services in computer networks. Its purpose is to induce extensibility, neutrality and independence.
- It uses XML for its message format, and relies on application layer protocols, most often Hypertext Transfer Protocol (HTTP) or Simple Mail Transfer Protocol (SMTP), for message negotiation and transmission. (thanks Wikipedia!)
- WSDL files are often used to define a SOAP service.
- Note: SOAP doesn’t require a WSDL, and WSDLs can be used to define non-SOAP services.

Diving in: A TR-069 Inform 1/5

```
<?xml version='1.0' encoding='utf-8'?>
<SOAP-ENV:Envelope
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:cwmp="urn:dslforum-org:cwmp-1-0">
  <SOAP-ENV:Header>
    <cwmp:ID SOAP-ENV:mustUnderstand="1">inform</cwmp:ID>
  </SOAP-ENV:Header>
<SOAP-ENV:Body SOAP-
ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
```

Inform 2/5

```
    <cwmp:Inform>
      <DeviceId>
          …
      </DeviceId>
      <Event xsi:type="SOAP-ENC:Array" SOAP-ENC:arrayType="cwmp:EventStruct[1]">
          …
      </Event>
   <MaxEnvelopes>1</MaxEnvelopes>
      <CurrentTime>2018-03-13T23:38:03</CurrentTime>
      <RetryCount>0</RetryCount>
      <ParameterList xsi:type="SOAP-ENC:Array" SOAP-
ENC:arrayType="cwmp:ParameterValueStruct[9]">
          …
      </ParameterList>
    </cwmp:Inform>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Inform 3/5 (DeviceId)

```
<DeviceId>
  <Manufacturer>AirTies</Manufacturer>
  <OUI>001CA8</OUI>
  <ProductClass>Air4920DK-WA</ProductClass>
  <SerialNumber>AT1931620001413</SerialNumber>
</DeviceId>
```

Inform 4/5 (Event)

```
      <Event xsi:type="SOAP-ENC:Array" SOAP-
ENC:arrayType="cwmp:EventStruct[1]">
        <EventStruct>
          <EventCode>2 PERIODIC</EventCode>
          <CommandKey/>
        </EventStruct>
      </Event>
```

Inform 5/5 (ParameterList)

```
<ParameterValueStruct>
  <Name>Device.DeviceInfo.SoftwareVersion</Name>
  <Value xsi:type="xsd:string">1.23.4.11.3795</Value>
</ParameterValueStruct>
<ParameterValueStruct>
  <Name>Device.ManagementServer.ParameterKey</Name>
  <Value xsi:type="xsd:string">2018-01-06 13:16:14.145930+00:00</Value>
</ParameterValueStruct>
<ParameterValueStruct>
  <Name>Device.ManagementServer.PeriodicInformInterval</Name>
  <Value xsi:type="xsd:unsignedInt">3600</Value>
</ParameterValueStruct>
```

TR-069 RPC Methods

- TR-069 defines the RPC methods in play in CWMP
- Next slide show a table of the required RPC methods which a CPE and an ACS must support.
- There is also a list of optional RPC methods which are not required. Stuff for another talk.

The Best Laid Plans...

- TR-069 is far from perfect. For example:
- It is way more difficult than it should be to figure out which CWMP version the CPE is using.
- It is way more difficult than it should be to figure out which Data Model the CPE is using.
- In some versions of CWMP or Data Models they simply forgot to include the version in the early versions, which makes it impossible to do this perfectly.
- Next slide shows one of several tables and flowcharts from TR-069 to help figure out the maze of versions

Table 11 – CWMP Version Negotiation

	1.0 CPE	1.1-1.3 CPE	1.4 (or later) CPE
1.0 ACS	1. CPE sends 1.0 namespace 2. ACS sends a 1.0 namespace response that does not contain UseCWMPVersion header element 3. CPE chooses CWMP version 1.0 ACS chooses CWMP version 1.0	1. CPE sends 1.1-1.3 namespace 2. ACS sends a fault or a 1.0 namespace response 3. CPE chooses CWMP version 1.0 ACS chooses CWMP version 1.0	1. CPE sends SupportedCWMPVersions header element 2. ACS ignores SupportedCWMPVersion header element 3. ACS sends a fault or a v1.0 namespace response 4. CPE chooses CWMP version 1.0 ACS chooses CWMP version 1.0
1.1-1.3 ACS	1. CPE sends 1.0 namespace 2. ACS sends a 1.0 namespace response that does not contain UseCWMPVersion header element 3. CPE chooses CWMP version 1.0 ACS chooses CWMP version 1.0	1. CPE sends 1.1-1.3 namespace 2. ACS does not send UseCWMPVersion header element 3. CPE chooses `min`(CPE CWMP version, ACS CWMP version) ACS chooses `min`(CPE CWMP version, ACS CWMP version)	1. CPE sends SupportedCWMPVersions header element 2. ACS ignores SupportedCWMPVersions header element 3. ACS does not send UseCWMPVersion header element 4. CPE chooses ACS CWMP version ACS chooses ACS CWMP version
1.4 (or later) ACS	1. CPE sends 1.0 namespace 2. ACS sends a 1.0 namespace response that does not contain UseCWMPVersion header element 3. CPE chooses CWMP version 1.0 ACS chooses CWMP version 1.0	1. CPE sends 1.1-1.3 namespace 2. ACS does not send UseCWMPVersion header element 3. CPE chooses CPE CWMP version ACS chooses ACS CWMP version	1. CPE sends SupportedCWMPVersions header element 2. ACS uses unspecified logic to determine CWMP version from the supplied list 3. ACS replies with UseCWMPVersion header element 4. CPE chooses CWMP version specified by UseCWMPVersion ACS chooses CWMP version specified by UseCWMPVersion

CWMP version != CWMP namespace

A.6 RPC Method XML Schema

The XML schema, which is the normative definition for all RPC methods defined for the CPE WAN Management Protocol, is specified in the referenced files below:

Table 89 – CWMP XML Schema Versions

CWMP Version	Namespace	XSD
1.0	urn:dslforum-org:cwmp-1-0	http://www.broadband-forum.org/cwmp.php/cwmp-1-0.xsd
1.1	urn:dslforum-org:cwmp-1-1	http://www.broadband-forum.org/cwmp.php/cwmp-1-1.xsd
1.2	urn:dslforum-org:cwmp-1-2	http://www.broadband-forum.org/cwmp.php/cwmp-1-2.xsd
1.3	urn:dslforum-org:cwmp-1-2	http://www.broadband-forum.org/cwmp.php/cwmp-1-3.xsd
1.4	urn:dslforum-org:cwmp-1-2	http://www.broadband-forum.org/cwmp.php/cwmp-1-4.xsd

1

End of TR-069 Intro

- So, we needed TR-069 at BornFiber
- We wanted our ACS tightly integrated into our existing system
- We always aim to keep our stack as small as possible. Managing software takes time too.
- Time to choose an ACS!

ACS Servers

- Picking an ACS server comes down to the usual choices:
- Open Source
  - http://www.freeacs.com/
  - https://github.com/genieacs/genieacs
- Commercial
  - Many options, and the large CPE manufacturers all have their own software available for a price.

Crazy idea: Could we build an ACS?

- Pros
  - We have some experience in this area
  - Would be tightly coupled with our current system
  - Would not expand our current software stack at all
  - We don’t need the entirety of ACS. We basically need to tcpdump a few sessions with GenieACS and serve up the same XML.
- Cons
  - Big standard, much XML
  - Device quirks could (and does) make things tricky
  - People tend to look at you funny at meetings

Tools and tips

- Have a local testbed, including DHCP, NTP, TFTP, DNS server...
- Use HttpRequester or similar browser plugin for quick testing
- Use real devices and Wireshark, early and always.
- Have a local greppable repo with the files from https://www.broadband-forum.org/standards-and-software/technical-specifications/tr-069-files-tools
- Like in https://github.com/tykling/cwmpdocs
- Also https://www.qacafe.com/tr-069-training/

SOAP in Python

- SOAP clients are easy to find, especially with a WSDL. I have used
  - Suds
  - Zeep
- A good list on https://wiki.python.org/moin/WebServices#SOAP

CWMP Clients

For testing purposes it can be nice to have a software CWMP client. But mostly I use real devices.

- Open Source
  - http://www.easycwmp.org/
- Commercial (used in Icotera for example)
  - https://www.avsystem.com/products/libcwmp/

4) Django Time, yay!

- An ACS SOAP server has a simple enough job, on paper:
  - Accept a POST request with some XML
  - Parse the XML, put a response together, send it back (with the correct MIME type)
  - Remember to include a session cookie so future requests can be linked to the same session
  - Done! Standard HTTP stuff, perfect for Django. Right? Right!
- We already had a (very simple) SOAP server running for a partner API which does callbacks over SOAP.
- This made the task of building an ACS server considerably less daunting. SOAP is not as dangerous or difficult as it sounds!

A Simple Django SOAP Server 1/2

```
from defusedxml.lxml import fromstring


@csrf_exempt
@require_http_methods(['GET', 'POST'])
def soap_callback(request):
    if request.method == 'POST':
        try:
            xmlroot = fromstring(request.body)
            validxml=True
            processed=False
        except Exception as E:
            print 'got exception parsing XML: %s' % E
            validxml=False
            processed=True
```

A Simple Django SOAP Server 2/2

```
        content = render_to_string('soap_response_response.xml')
        response = HttpResponse(content, content_type='text/xml;
charset=utf-8')
    else:
        # no POST, just render and return the wsdl, hello Waoo
        content = render_to_string('soap_response.wsdl', {
            'hostname': settings.WAOO_SOAP_RESPONSE_HOSTNAME
        })
        response = HttpResponse(content, content_type='text/xml;
charset=utf-8')
    # set content-length header (or nothing works)
    response['Content-Length'] = len(content)
    return response
```

Introducing MrX

- Our central provisioning and CRM system at BornFiber
- Django project written from scratch
- Maintains records on customers, products, network equipment, circuits and much more.
- Already interfaces with the world outside of Django in many ways:
  - Partner SOAP API for provisioning tv stuff (client&server)
  - Dynamic TFTP server (no files)
  - Manages FreeRADIUS tables (DHCP and network)
  - Configures network equipment over SSH
  - Others I’ve forgotten

Introducing MrX

- Large project!
- First commit July 2015, currently at 4745 commits by 3 authors.
- 156 models across 30 apps

```
user@bornfiber-acs-dev:~/devel/mrx/src/mrx$ cloc .
    1005 text files.
     994 unique files.
     156 files ignored.

github.com/AlDanial/cloc v 1.70  T=20.77 s (45.0 files/s, 3073.8 lines/s)
-------------------------------------------------------------------------------
Language                     files          blank        comment           code
-------------------------------------------------------------------------------
Python                         309           6030           3389          31310
HTML                           611            565             47          19573
JavaScript                       5            197             45           1878
CSS                              7             90             34            623
JSON                             1              0              0             41
XML                              2              0              0             10
-------------------------------------------------------------------------------
SUM:                           935           6882           3515          53435
-------------------------------------------------------------------------------
user@bornfiber-acs-dev:~/devel/mrx/src/mrx$
```

Lessons Learned

- Django is an excellent choice for this kind of system
- We deloy to production multiple times a day
- Since the system is used a lot we needed a way to deploy with little or no downtime. Currently we use uwsgi with –touch-chain-reload
- We should have started out with fatter models and slimmer views, we are getting there.
- Also working towards py3 and Django 2.0
- And Channels.
- History is hard.
- Building an asynchronous system on top of a synchronous system is not without issues (race conditions?), MrTX?

The Django ACS App

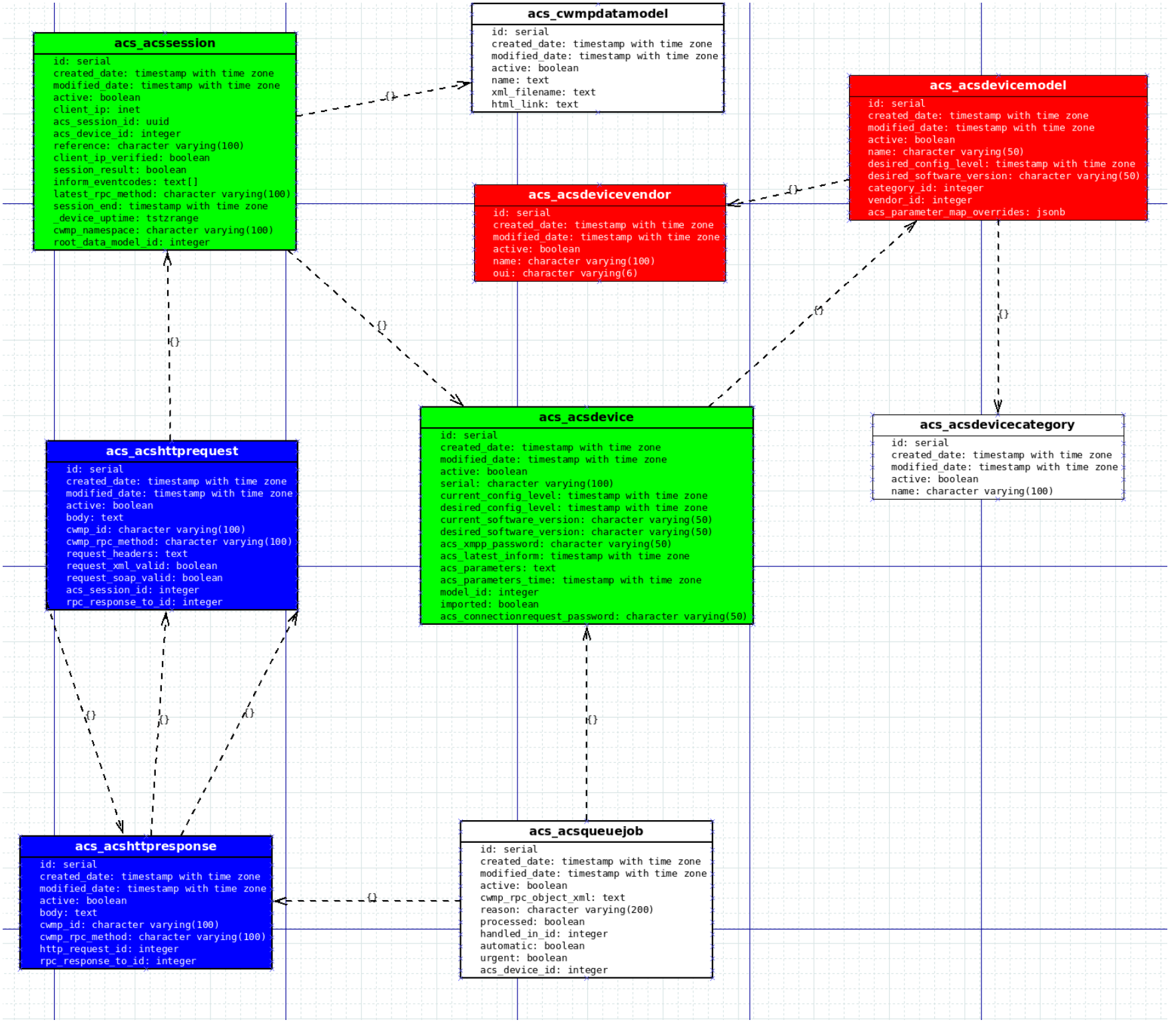
- The ACS is mostly one view, AcsView. 425 lines of code, 220 without comments and empty lines.
- 9 models plus a couple of abstract models, take up around 1400 lines, or just over 1000 without comments.
- Other apps are expected to subclass AcsDeviceBaseModel.
- We will look at the models before we dive into the server view.

ACS App Models

- acs_device.py – Dynamically created ACS clients
- acs_device_basemodel.py – Abstract model
- acs_device_category.py – Grouping devices
- acs_device_model.py – HW model (from Inform)
- acs_device_vendor.py – HW vendor (from Inform)
- acs_http_basemodel.py – Abstract for the following 2
- acs_http_request.py – All HTTP requests received
- acs_http_response.py – All HTTP responses sent
- acs_queue_job.py – ACS job queue
- acs_session.py – ACS sessions

ACS App Models

- Each new ACS client creates a new AcsDevice object.
- Each ACS session creates a new AcsSession object, with a fk to AcsDevice.
- Each HTTP request creates a new AcsHttpRequest object.
- Each HTTP response is saved as a new AcsHttpResponse object.
- AcsDeviceModel and AcsDeviceVendor objects are created as needed.


acs_acssession
id: serial
created_date: timestamp with time zone
modified_date: timestamp with time zone
active: boolean
client_ip: inet
acs_session_id: uuid
acs_device_id: integer
reference: character varying(100)
client_ip_verified: boolean
session_result: boolean
inform_eventcodes: text[]
latest_rpc_method: character varying(100)
session_end: timestamp with time zone
_device_uptime: tstzrange
cwmp_namespace: character varying(100)
root_data_model_id: integer
acs_cwmpdatamodel
id: serial
created_date: timestamp with time zone
modified_date: timestamp with time zone
active: boolean
name: text
xml_filename: text
html_link: text
acs_acsdevicemodel
id: serial
created_date: timestamp with time zone
modified_date: timestamp with time zone
active: boolean
name: character varying(50)
desired_config_level: timestamp with time zone
desired_software_version: character varying(50)
category_id: integer
vendor_id: integer
acs_parameter_map_overrides: jsonb
acs_acsdevicevendor
id: serial
created_date: timestamp with time zone
modified_date: timestamp with time zone
active: boolean
name: character varying(100)
oui: character varying(6)
acs_acsdevice
id: serial
created_date: timestamp with time zone
modified_date: timestamp with time zone
active: boolean
serial: character varying(100)
current_config_level: timestamp with time zone
desired_config_level: timestamp with time zone
current_software_version: character varying(50)
desired_software_version: character varying(50)
acs_xmpp_password: character varying(50)
acs_latest_inform: timestamp with time zone
acs_parameters: text
acs_parameters_time: timestamp with time zone
model_id: integer
imported: boolean
acs_connectionrequest_password: character varying(50)
acs_acsdevicecategory
id: serial
created_date: timestamp with time zone
modified_date: timestamp with time zone
active: boolean
name: character varying(100)
acs_acshttprequest
id: serial
created_date: timestamp with time zone
modified_date: timestamp with time zone
active: boolean
body: text
cwmp_id: character varying(100)
cwmp_rpc_method: character varying(100)
request_headers: text
request_xml_valid: boolean
request_soap_valid: boolean
acs_session_id: integer
rpc_response_to_id: integer
acs_acshttpresponse
id: serial
created_date: timestamp with time zone
modified_date: timestamp with time zone
active: boolean
body: text
cwmp_id: character varying(100)
cwmp_rpc_method: character varying(100)
http_request_id: integer
rpc_response_to_id: integer
acs_acsqueuejob
id: serial
created_date: timestamp with time zone
modified_date: timestamp with time zone
active: boolean
cwmp_rpc_object_xml: text
reason: character varying(200)
processed: boolean
handled_in_id: integer
automatic: boolean
urgent: boolean
acs_device_id: integer

An ACS Device

AcsDevice details for AD#550

ACS Device	AD#550
Vendor	DV#1 - AirTies
Model	DM#1 - Air4920DK-WA
Serial	AT1931603005299
Related Device	WD#565 (AT1931603005299)
Latest Inform	12/03-2018 15:39
Current Config Level	12/03-2018 15:22
Desired Config Level	15/03-2018 15:32 (LOCAL OVERRIDE)
Current Software Version	1.23.4.11.3795
Desired Software Version	1.23.4.11.3795 (LOCAL OVERRIDE)

Actions

+ Schedule ACS RPC Job

AD#550 created: 27/01-2017 14:24 (system) - AD#550 modified: 15/03-2018 15:32 (tsr)

Wifi Device Acs Parameters Last Updated: 2018-03-12 14:39:38.165249+00:00

Filter:

parameter	type	writable	notification	accesslist	value
Device.DeviceInfo.AdditionalHardwareVersion	xsd:string	0	0	Subscriber	None
Device.DeviceInfo.AdditionalSoftwareVersion	xsd:string	0	0	Subscriber	None
Device.DeviceInfo.Description	xsd:string	0	0	Subscriber	AirTies DualConcurrent AP
Device.DeviceInfo.DeviceCategory	xsd:string	0	0	Subscriber	COM_X_AccessPoint:AirTies
Device.DeviceInfo.FirstUseDate	xsd:dateTime	0	0	Subscriber	2017-01-26T14:50:57Z
Device.DeviceInfo.HardwareVersion	xsd:string	0	0	Subscriber	TW_0.7
Device.DeviceInfo.Manufacturer	xsd:string	0	0	Subscriber	AirTies Wireless Networks
Device.DeviceInfo.ManufacturerOUI	xsd:string	0	0	Subscriber	001CA8
Device.DeviceInfo.ModelName	xsd:string	0	0	Subscriber	Air4920
Device.DeviceInfo.ModelNumber	xsd:string	0	0	Subscriber	4920

AcsDevice Model Fields

```
class AcsDevice(BaseModel):
    model = models.ForeignKey('acs.AcsDeviceModel', related_name="acsdevices")
    serial = models.CharField(max_length=100)
    current_config_level = models.DateTimeField(null=True, blank=True)
    desired_config_level = models.DateTimeField(null=True, blank=True)
    current_software_version = models.CharField(max_length=50, blank=True)
    desired_software_version = models.CharField(max_length=50, blank=True)
    acs_xmpp_password = models.CharField(max_length=50, blank=True)
    acs_latest_inform = models.DateTimeField(null=True, blank=True)
    acs_parameters = models.TextField(blank=True)
    acs_parameters_time = models.DateTimeField(null=True, blank=True)
    acs_connectionrequest_password = models.CharField(max_length=50, blank=True)
```

AcsDevice Model Methods 1/2

def **latest_acs_session**(self): Return the latest AcsSession object

def **handle_user_config_changes**(self): Check self.acs_parameters for changes

def **get_desired_config_level**(self): Return the config level we want for this device

def **get_desired_software_version**(self): Return the software version we want

def **get_software_url**(self, version): Returns the download URL for the firmware for this device

def **update_acs_parameters**(self, attributes_rpc_response): Updates local XML param. cache

def **acs_parameter_dict**(self): Return a dict of the local XML parameter cache

def **acs_get_parameter_value**(self, parameterpath): Returns a specific value

AcsDevice Model Methods 2/2

def **acs_xmpp_username**(self): Returns the XMPP conn. req. username for this device

def **create_xmpp_user**(self): Creates an account on the XMPP server for this device

def **create_connreq_password**(self): Create a new HTTP conn. req. password

def **acs_connection_request_url**(self): Returns the connectionrequest URL for this device

def **acs_http_connection_request**(self): Do a connectionrequest (trigger Inform)

def **latest_client_ip**(self): Return the client IP this device had the last time we saw it

def **associate_with_related_device**(self): Associate this ACS device with a related real device

def **get_related_device**(self): Returns the real device with a relation to this AcsDevice

The ACS Session List

Acs Session list

Show all sessions | Show only failed sessions | Show only failed and ip verified sessions

Page 1 of 139611 next

acs session	acs device	client ip	client ip verified	Inform Event Codes	http req	http resp	start	duration	device uptime	last rpc method	bytes received	bytes sent	actions
AS#3502593	AD#2222	10.17.48.39	✔	2 PERIODIC	5	4	15/03-2018 18:03	0:00:00	11 days, 6:02:19	InformResponse	229516 bytes	2346 bytes	details
AS#3502592	AD#1198	10.19.104.92	✔	2 PERIODIC	5	5	15/03-2018 18:03	0:00:09	31 days, 18:58:15	(empty response body)	220395 bytes	2346 bytes	details
AS#3502591	AD#1135	10.19.96.26	✔	2 PERIODIC	5	5	15/03-2018 18:03	0:00:09	31 days, 18:59:21	(empty response body)	229456 bytes	2346 bytes	details
AS#3502590	AD#2331	10.19.50.52	✔	2 PERIODIC	5	5	15/03-2018 18:03	0:00:08	4 days, 6:02:27	(empty response body)	193061 bytes	2346 bytes	details
AS#3502589	AD#2322	10.19.50.82	✔	2 PERIODIC	5	5	15/03-2018 18:03	0:00:07	10 days, 21:02:19	(empty response body)	202174 bytes	2346 bytes	details
AS#3502588	AD#1029	10.17.35.140	✔	2 PERIODIC	5	5	15/03-2018 18:02	0:00:08	22:02:29	(empty response body)	211315 bytes	2346 bytes	details
AS#3502587	AD#2356	10.17.200.167	✔	2 PERIODIC	5	5	15/03-2018 18:02	0:00:08	2 days, 12:02:32	(empty response body)	193072 bytes	2346 bytes	details
AS#3502586	AD#180	10.19.101.247	✔	2 PERIODIC	5	5	15/03-2018 18:02	0:00:08	1 day, 10:02:30	(empty response body)	211288 bytes	2346 bytes	details
AS#3502585	AD#1990	10.19.84.53	✔	2 PERIODIC	5	5	15/03-2018 18:02	0:00:08	2 days, 1:02:27	(empty response body)	211304 bytes	2346 bytes	details
AS#3502584	AD#2178	10.17.36.100	✔	2 PERIODIC	5	5	15/03-2018 18:02	0:00:08	1 day, 8:02:29	(empty response body)	220419 bytes	2346 bytes	details
AS#3502583	AD#2350	10.19.106.249	✖	1 BOOT	5	5	15/03-2018 18:02	0:00:06	0:01:06	(empty response body)	179289 bytes	2346 bytes	details

An ACS Session

Acs Session details for AS#2929039

Acs Session	AS#2929039
Acs Device	AD#550
Client IP	127.0.0.1 (verified)
Acs Session ID	eec1b516c3ca40459c0c1a79b66f6d59
Inform EventCodes	1 BOOT
Acs Session Result	✔
Acs Device Uptime	0:02:59

AS#2929039 created: 12/03-2018 15:39 (system) - AS#2929039 modified: 12/03-2018 15:39 (system)

Acs Session Http Conversation

reqresp	direction	when	rpc method	headers	body	rpc response to	rpc response	acs queue job	actions
AP#11647941	Http response	12/03-2018 15:39	(empty response body)	N/A	0 bytes	N/A	N/A	N/A	details
AR#11662764	Http Request	12/03-2018 15:39	GetParameterAttributesResponse	266 bytes	83768 bytes	AP#11647940	N/A	N/A	details
AP#11647940	Http response	12/03-2018 15:39	GetParameterAttributes	N/A	630 bytes	N/A	AR#11662764	AQ#5890427	details
AR#11662763	Http Request	12/03-2018 15:39	GetParameterValuesResponse	266 bytes	54131 bytes	AP#11647939	N/A	N/A	details
AP#11647939	Http response	12/03-2018 15:39	GetParameterValues	N/A	622 bytes	N/A	AR#11662763	AQ#5890426	details
AR#11662762	Http Request	12/03-2018 15:39	GetParameterNamesResponse	266 bytes	52723 bytes	AP#11647938	N/A	N/A	details
AP#11647938	Http response	12/03-2018 15:39	GetParameterNames	N/A	568 bytes	N/A	AR#11662762	AQ#5890425	details
AR#11662761	Http Request	12/03-2018 15:39	(empty request body)	266 bytes	0 bytes	N/A	N/A	N/A	details
AP#11647937	Http response	12/03-2018 15:39	InformResponse	N/A	526 bytes	AR#11662760	N/A	N/A	details
AR#11662760	Http Request	12/03-2018 15:39	Inform	166 bytes	2372 bytes	N/A	AP#11647937	N/A	details

AcsSession Model Fields

```
class AcsSession(BaseModel):
    acs_device = models.ForeignKey('acs.AcsDevice', null=True,
blank=True, related_name='acs_sessions')
    acs_session_id = models.UUIDField(default=uuid.uuid4)
    client_ip = models.GenericIPAddressField()
    client_ip_verified = models.BooleanField(default=False)
    reference = models.CharField(max_length=100, default='', blank=True)
    session_result = models.BooleanField(default=False)
    latest_rpc_method = models.CharField(max_length=100, default='',
blank=True)
    session_end = models.DateTimeField(null=True, blank=True)
    _device_uptime = DateTimeRangeField(null=True, blank=True) # use the
property device_uptime instead
    inform_eventcodes = ArrayField(models.TextField(), default=list,
blank=True)
    cwmp_namespace = models.CharField(max_length=100, default='',
blank=True)
    root_data_model = models.ForeignKey('acs.CwmpDataModel', null=True,
blank=True, related_name='acs_sessions')
```

AcsSession Model Methods

def **configuration_done**(self): Property is true if already configured

def **get_device_parameterdict**(self, configdict): Returns dict with everything

def **configure_device_parameter_attributes**(self, reason, parameterlist, update_parameterkey=False): Change attributes on device

def **configure_device_parameter_values**(self, reason, update_parameterkey=False): Change values on device

def **soap_namespaces**(self): Returns a dict of namespaces to use

def **get_inform_eventcodes**(self, inform): Return list of Inform eventcodes

def **determine_data_model**(self, inform): Figure out which DM we're using

def **add_acs_queue_job**(self, cwmp_rpc_object_xml, reason, automatic=False, urgent=False): Add a new ACS job

def **configure_device**(self): Configures the device (if possible)

def **get_latest_http_tx**(self): Return latest HTTP request/response

def **update_session_result**(self): Session result is True if clean session

def **get_latest_rpc_method**(self): Return name of the latest RPC method

AcsSession Model Methods

def **acs_rpc_get_rpc_methods**(self, reason, automatic=False, urgent=False):

def **acs_rpc_set_parameter_values**(self, reason, parameterdict, automatic=False, urgent=False, update_parameterkey=False):

def **acs_rpc_get_parameter_values**(self, reason, parameterlist, automatic=False, urgent=False):

def **acs_rpc_get_parameter_names**(self, reason, parampath='', nextlevel='0', automatic=False, urgent=False):

def **acs_rpc_get_parameter_attributes**(self, reason, parameterlist, automatic=False, urgent=False):

def **acs_rpc_set_parameter_attributes**(self, reason, parameterdict, automatic=False, urgent=False, update_parameterkey=False):

def **acs_rpc_add_object**(self, reason, objectname, automatic=False, urgent=False, update_parameterkey=False):

def **acs_rpc_delete_object**(self, reason, objectname, automatic=False, urgent=False, update_parameterkey=False):

def **acs_rpc_reboot**(self, reason, automatic=False, urgent=False):

def **acs_rpc_download**(self, reason, parameterdict, automatic=False, urgent=False):

def **acs_rpc_upload**(self, reason, parameterdict, automatic=False, urgent=False):

def **acs_rpc_factory_reset**(self, reason, automatic=False, urgent=False):

def **acs_rpc_schedule_inform**(self, reason, parameterdictdict, automatic=False, urgent=False):

Using AcsDeviceBaseModel

- The idea is that other apps (in this example WifiService.WifiDevice) will subclass AcsDeviceBaseModel, overriding methods as needed
- An ACS device can live happily in the AcsDevice model without ever being associated with a real device – an AcsDevice it is just an HTTP client after all
- A real device can exist happily in a model subclassing AcsDeviceBaseModel without ever seeing an ACS session.
- But ideally most objects in the model subclassing AcsDeviceBaseModel will end up having a FK relation to AcsDevice
- IP verification on each session through radius ensures everything is secure

AcsDeviceBasemodel methods 1/2

```
class AcsDeviceBaseModel(BaseModel):
    class Meta:
        abstract = True

    def verify_acs_client_ip(self, ip):
        """ Method to verify the ACS client IP, override in your own models. """
        raise NotImplementedError

    def is_configurable(self):
        """ Method to determine if an acsdevice is configurable, override in your own models. """
        raise NotImplementedError

    def acs_session_pre_verify_hook(self):
        """ This method is called every time an ACS device runs an ACS session, before
verify_acs_client_ip() is called. Override in your own models as needed. """
        return False
```

AcsDeviceBasemodel methods 2/2

```
    def acs_session_post_verify_hook(self):
        """ This method is called every time an ACS device runs an ACS session, after verify_acs_client_ip() is called. Override in your own models as needed. """
        return False


def get_acs_config(self):
        """ This method is called while configuring an ACS device. Override in your own models to add device specific config."""
        raise NotImplementedError


    def get_user_config_changelist(self):
        """This method should acs_device.acs_parameters versus the local records and returns a list of changed elements, if any. Should return an empty list if everything in acs_parameters matches the local records."""
        raise NotImplementedError


    def handle_user_config_change(self):
        """Called whenever the configuration on an ACS device is different from what we configured on it."""
        raise NotImplementedError
```

Device Quirks

- TR-069 is not perfect
- And it is pretty complex
- And CPEs are developed on a budget
- So expect bugs.
- Bad ones.

Device Quirks - HTTP

- The first device I worked with has pretty poor HTTP support
- It doesn’t support HTTPS.
- It doesn’t even support HTTP 1.1
- It resolves the hostname, and inserts the IP address as Host: header, and does the request. Basically HTTP/0.9.
- We run our ACS server on a seperate port for this reason
- Actually IANA assigned TCP/7547 for CWMP

Device Quirks - TLS

- TLS support is poor to say the least. TR-069 mandates TLS, latest version says minimum TLS 1.0 but recommends 1.2.
- Real world looks a lot different.
- TLS is used for CWMP of course, but also for downloads and for the XMPP client, to name a few.
- I have yet to encounter a device with a nice, well-functioning TLS client.
- Problems range from no TLS at all, to bad ciphers, and bad CA support.

Device Quirks – Cookie Handling

- Something as plain as cookie handling can present issues.
- We use the normal Django cookie-based session handling which worked well until we came across a device where it didn’t.
  - Set-Cookie: acs_session_id=6244efcf78c741c38defa63105fb457c; expires=Wed, 21-Feb-2018 14:03:53 GMT; Max-Age=60; Path=/
- Became:
  - Cookie: acs_session_id=6244efcf78c741c38defa63105fb457c; $Path="/acs/";$Domain="acs.example.invalid"
  - Cookie: 21-Feb-2018;$Domain="acs.example.invalid"

Device Quirks – Data Model

- In something as complex as TR-069 it is pretty important to keep to the standards.
- Mostly devices pick a CWMP version (actually the server picks it really) and they use some datamodel, which can be read or inferred from the Inform. But not always:

Hej Thomas,

Lige nu er der implementeret en blanding af TR98, TR181 og noget AirTies specifikt.

De arbejder hen imod kun at have TR181 - men det er ikke noget der er lige på trapperne, dvs. inden for 3 md. Der vil nok gå mellem 6 og 12 md. før vi er der.

Hindsight and Future Plans

- The ACS app has changed multiple times. The first iteration was considerably less DRY.
- I consider it more or less complete for our needs. Future plans include:
  - Keep XML in files instead of the DB
  - We are adding more devices
  - I want to opensource the ACS app (although I have no idea how)

Numbers

- AcsSession: 3.502.819
- AcsHttpRequest: 14.529.910
- AcsHttpResponse: 14.514.803
- Average session time: 8 sec (ish)

Questions

?