A Software Friendly DOCSIS Control Plane

Alon Bernstein
Cisco Systems

Abstract

It has been 15 years since the initial set of DOCSIS specs have been authored. In those 15 years software engineering has seen an explosion in productivity at the same time that the DOCSIS control plane has remained fairly unchanged. Can we apply these productivity tools to the DOCSIS control plane to facilitate greater simplicity and feature velocity?
This paper will outline both software trends and protocol design trends that are relevant to the above discussion and how they can be applied to DOCSIS.

OVERVIEW

DOCSIS is primarily an interface protocol between a CM and a CMTS. There is a wide palette of options for a protocol designer and all are relevant to DOCSIS design. Each option has its tradeoffs and the role of the protocol designer is to choose the option that fits the system requirements and constraints the best. The list of options include:

- Generalized interface vs. mission specific interface
- Legacy protocol vs. mission specific protocol
- Stateless vs. stateful
- Client-Server vs. Peer-to-peer

And more…In many cases there are no simple rights and wrongs and a choice that might have made sense at the time of the protocol design turns out to be sub-optimal as systems often end up getting deployed in a manner that is different then what they were designed for. All of these choices have an impact on software. Its not always the case the choice that is optimal for software is the ideal for meeting the system requirements, still its unfortunate that in many cases the software implementation ease is considered as a relatively low priority item. This observation is patricianly in place for DOCSIS since the amount of software resources needed to support the DOCSIS set of protocols is significant.
Before proceeding, a word of caution: in cases where there are requirements and constraints that supersede software requirements then clearly the guidelines explained in this document will not apply. The challenge is to identify these requirements and constrains correctly and not to pre-optimize at the expense of "software friendliness". To quote the author of the "Art Of Computer Programming":

> *"We should forget about small efficiencies, say about 97% of the time: premature optimization is the root of all evil"* (Donald Knuth)

SOFTWARE CONSIDERATIONS

Software engineering and protocol design share the same approach to simplifying complex system requirements:

- Modularization: sub-divide a large and complex system into simple and easy to test modules with well-defined inputs and outputs.
- Layering: Define the hierarchy of modules, what services each component provides to another.
- Abstraction: identifying common services that can be shared across modules

Software design methodologies have evolved around the same timelines as the Internet revolution and the creation of networking protocols. But while the timelines are similar the amount of change software went through is much larger the amount of change in the suite of networking protocols that drive the Internet.

Software has evolved from the "C" programming language to object oriented languages such as C++ and Java which allowed for further modularization/layering and abstraction of code to web technologies that brought amazing scale, speed and flexibility. At the same time network protocols stuck with the OSI 7 layer model [6] as possibly the only attempt to apply modularization/layering and abstraction to networking. Case in point: many of the routing protocol RFCs have pages upon pages of interface specifications and message formants. If written from with a "software friendliness" point of view they could have had a well-defined separation of the methods to distribute data across a group of routers (which is similar in many of the routing protocols) and the actual routing algorithms.

Obviously the issue outline above has a wider scope then DOCSIS, so to keep the discussion focused here are a couple of examples of why the current DOCSIS specifications does not follow basic software implementation guidelines along with a high-level proposal on how to fix it (going into fine details is outside the scope of this paper):

Example 1: DOCSIS registration.

The DOCSIS registration process starts with bringing up the physical layer, jumps to authentication and IP bring up, then to service provisioning then back to physical layer bring up.

Initially services are created in the registration process. Additional services are added with a different mechanism then the one used in registration (DSx).

Why is it not software friendly? The fact that the cable modem bring-up has a dependency on the IP layer bring-up makes it difficult to independently develop (aka "feature velocity") and independently test (aka "product quality"), the registration process.

This is a good example where an idea that seemed to offer:

1. Simplification: because the same mechanism used to provision services is also used for the modem bootstrap
2. Optimization: fewer messages since all the various layers are squeezed into the same

Turns out to be not-such-a-good-idea when it comes to software implementation. This becomes painfully obvious when the system is physically distributed. Imagine an implementation where DOCSIS functionality is segregated into processor A and Layer 3 functionality into processor B. Because of the way registration is handled the DOCSIS processor needs to know a little about the IP layer and the IP processor needs to know a bit about DOCSIS. Clearly these are solvable problems and "anything can be done in software" but as mentioned above there is a price to pay in speed of implementation, testability and debug of system issues.

How would a software friendly registration protocol look? A software friendly specification would have clear and independent stages as depicted in the figure below:

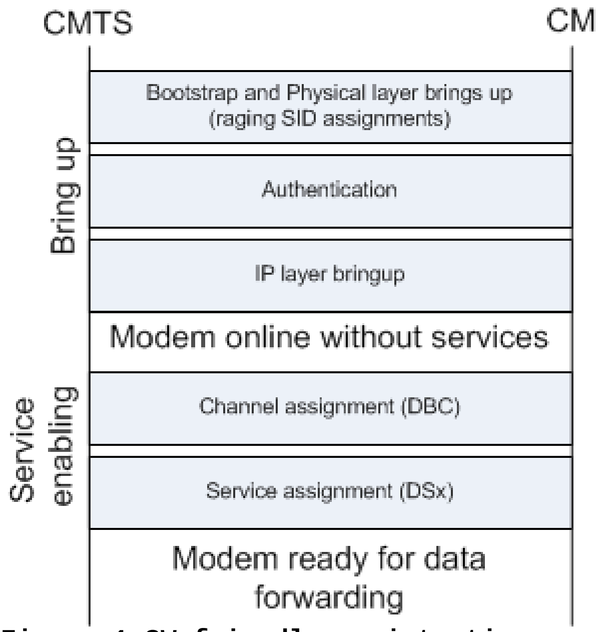
CMTS
CM
Bring up
Bootstrap and Physical layer brings up (raging SID assignments)
Authentication
IP layer bringup
Modem online without services
Service enabling
Channel assignment (DBC)
Service assignment (DSx)
Modem ready for data forwarding

Figure 1 SW friendly registration

1. bootstrap: initial physical layer bring up

2. Authentication: validation of the cable modem for network access
3. L3 bring up: DHCP processing and IP address assignment

Each one of the above steps would be treated as an independent transaction and the three of them would be the workflow needed to bring a cable modem online.

For these 3 steps the major deviation from the current registration is that we don't rely on TFTP for service provisioning. There are two reasons to skip TFTP; the first being that the IP layer is not even up so we can't access anything beyond the CMTS and the second being that we want to postpone the service providing part to a later stage anyway. Having said all that, the CM still needs to communicate with the CMTS and it still needs some form of a service flow to do it. If we don't have any services provisioned how do the CMTS and CM communicate? The "temporary flow" that DOCSIS creates anyway for registration would just leave on until after the IP bring-up phase and only after that would be replaced by the "real" in the service enablement phase

As far as the next steps go we fortunately have clear transactions to handle:

- Service provisioning using DSX
- Changing physical layer parameters with the DBC

These can be used to change services and channel assignments after the modem is online. Note that the CMTS can still use TFTP to retrieve service parameters and those will be parsed into a DSA message.

Naturally there are tradeoffs to this proposal; the number of messages has increased and a new form of service enablement has been added, however the payoff is significant in terms of software modularity.

Example 2: The Mac Domain Descriptor MDD message is a dumping ground for information about plant topology, IPv6, error message report throttling, security, physical layer parameters and more. A software friendly specification would create independent messages for each of the functional areas. Though one can argue that MDD is "just a transport" for data that can be managed by independent modules in the software, however the inclusion of all of them in the same message creates dependencies that are easier to avoid were the MDD to be broken into separate messages.

END-TO-END PRINCIPAL

Some link-level protocols (such as DOCSIS) assume reliability is required and come up with their own set of timers to assure delivery at the link layer. This might be justified if the link layer is highly unreliable, and even in that case the timeouts set for retransmission must be an order of magnitude shorter then the timeouts of the end-to-end application. If retransmission timers are too long then all sorts of odd corner cases might occur. For example: the DSx-RSP timeout is about 1 sec and there can be 3 of them. If an end-to-end signaling protocol, such as SIP [4] has a message re-transmission time of the same order of magnitude then a DOCSIS implementation might release a SIP message when the application level has already timed out and re-transmitted its own copy. This will not cause the system to break since a robust implementation knows how to deal with messages that are duplicated or out of order, but it will clutter the error counters and fault logs with a "duplicate message" event which would have been avoided if the DOCSIS link layer counted (as it should have) on an end-to-end session establishment protocol. The reader might ask, "what if the end-to-end protocol is designed to be unreliable"? Even in that case it's not the role of the DOCSIS link-layer to assure delivery if the higher lever application does not require assurance in order to operate correctly.
The DOCSIS software may trigger a timeout for a DSx-RSP, however the expiration of this timeout would be only used for recording a failure and releasing system resources allocated for the DSx, not for triggering a re-

transmission.

If the media is highly unreliable and failures are a common occurrence then they might be room for link-level error repair but in that case the timeouts need to be an order of magnitude shorter then the application timeouts - a suggested range would be 100ms or so.

A further simplification based on the end-to-end principal is to remove the DSx-ACK phase. DOCSIS uses a 3-way handshake for DSx. The rough outline of the conversation at the service activation phase goes like this:

1. CMTS -> CM: please start a service (DSA-REQ)
2. CM -> CMTS: ok, I started the service (DSA-RSP)
3. CMTS -> CM: cool, my CMTS resources are ready you can start sending data (DSA-ACK)

But this third step is not really needed for the same end-to-end argument. For example consider this zoon-in of a PCMM message sequence (figure 9 in ref [2])

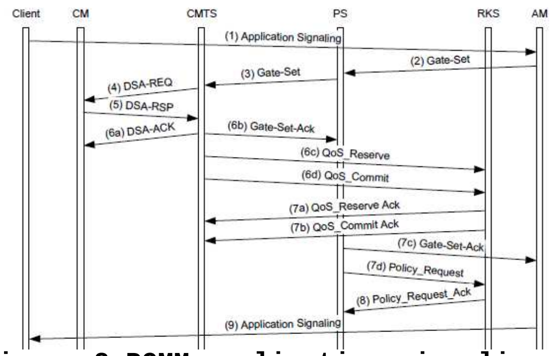

Figure 2 PCMM application signaling

It's obvious from this diagram that the DSA-ACK is not fulfilling any useful function. For one it's a "dead-end" not resulting in any further action, and since it is sent at the same time as the Gate-Set-Ack it is useless in guaranteeing any sequencing of events.

As a side note, some have suggested that the DSA-ACK is needed for extra reliability but this would be an even worst violation of protocol rules since the DSA-RSP is already an acknowledgement and a protocol should not acknowledge and acknowledgment.

ENCODING

DOCSIS uses TLVs to serialize information however TLVs are not common in modern networking stacks and not supported in many of the productivity tools and code generation tools used today. Non-TLV types of encodings include JSON/HTTP/XML/google "protocol buffers" and others. The advantages of the above mentioned tools are:

1. They come with code generation tools that relive the software developer from the burden of parsing messages into native data structures.

2. Most of them encode information in human readable strings that makes debugging easier.

TLVs are a more compact form of serializing data but as bandwidth available on the cable media increases this is becoming a non-issue.

TLVs might also be easier to parse, but CPU power is much less of an issue then it was at the time the DOCSIS specification were written. In fact, the modern cable modems have more powerful CPUs then early CMTS products!

OPEN SOURCE

Another software trend that has been going strong is the movement to open source. As a development methodology it has proven to deliver on wide scale and highly complex software projects. How can open source apply to cable? The CMTS/CM interaction is not likely to be of interest to the open source committee since its so domain specific and for product differentiation reasons it's highly unlikely that CMTS/CM vendors will open their source code.

This document proposes to use source a companion to the CableLabs standard documentation process. For example, if a new registration process was to be pursued then high-level function calls and JSON encodings could be published as open source. This would hopefully promote better interoperability and shorter ATP cycles as it removes a lot of ambiguity in the interface design.

DATABASE TECHNOLGIES

A CMTS implementation needs to manage a database of cable modems, plant topology and more. In many cases this information needs to be shared with a CM and so one view of a DOCSIS system could be that it's a distributed database of CM state and resources. With that observation it's clear that the only type of data sharing that DOCSIS allows for is the transactional type. That used to be the only model for data sharing in the database industry in general but the scale that companies such as google and facebook had to grow to gave rise to a new model, one that priorities performance over accuracy. Clearly for some types of data this model will not work well (financial transactions for example) while for others it makes sense (searching through web pages).

An interesting observation made by the Internet community is captured in what's called the "CAP theorem" [5]. In a nutshell what the CAP theorem states is that when a database designer is requested to support a distributed database that provides [1]:

1. Performance
2. Consistency of data across components of the distributed database
3. Resiliency in cases for system malfunctions, for example, packet drops.

Only two out of these three requirements can be met. The designer still has to choice of which two are fulfilled, but it is not possible to meet all three.

As mentioned above DOCSIS supports a transactional sharing of data that represents a choice of consistency and resiliency over performance, and as long as performance requirements are met (for example, number of voice call created per-second) it is a win-win situation. But as new applications become available and the load on the control plane increases it may make sense to consider other choices. The proposal in the previous section to avoid re-transmissions represents the option of demoting resiliency. Another option is to assume an "optimistic model" where the CMTS can allocate and activate resources on a DSA-REQ, assuming that a positive DSA-RSP will follow and intentionally allowing for short period of times of inconsistency if cases where the DSA-RSP was not successful.

Another useful tool from the database world is the concept of "data normalization". Its outside the scope of this paper to go into the detail of data normalization (see ref [3]), but in a nutshell it's a set of guidelines on how to break complex data into a list of tables with rows and columns where each row is fairly atomic. When inspecting some of the DOCSIS MIBs and MAC messages its obvious that some break at least one of the normal forms. For example, the inclusion of a "service flow reference" in the same table as the "service flow id" violates the normal form that prohibits dependencies between columns of a table. Without getting into too many details this paper only makes the observation that management constructs that are "normal" are easier to implement in software.

SECURITY

An obvious security hole in DOCSIS is letting the cable modem parse the configuration in order to reflect it back to the CMTS. It's worth mentioning it in this document because (ironically) this might have been the single attempt in DOCSIS to help software by offloading the task of parsing the cable modem configuration to the cable modem. However, in order to plug this security hole the CMTS needs to parse the configuration anyway and overall it's a great example of how premature optimization can create more harm then good.

[1] [ab] CAP stands for "consistency, availability, and fault tolerance". I took the liberty of translating the above to terms familiar to the cable community since the original terminology might be confused with existing cable terms.

CONCLUSION

DOCSIS has obviously been a very successful protocol. The DOCSIS provisioning and back-office system is part of this success, especially when comparing it to its DSL counterparts where a strong standard for provisioning and service enablement does not exist. Nevertheless a 15-year critical review, and possible updates would certainly help DOCSIS to become even better in facing the challenges ahead.

This paper suggests that software implementation ease and modern software tools need to play a bigger role in the design of future DOCSIS protocols and while some of the proposals made here are of academic and demonstrative value only, others can be relevant to future versions and enhancements of DOCSIS.

REFERANCES

1. DOCSIS MULPI : http://www.cablelabs.com/specifications/CM-SP-MULPIv3.0-I18-120329.pdf
2. PCMM: http://www.cablelabs.com/specifications/PKT-SP-MM-I06-110629.pdf
3. Codd, E.F. (June 1970). "A Relational Model of Data for Large Shared Data Banks". Communications of the ACM 13 (6):377–387.doi:10.1145/362384.362685.
4. Session Imitation Protocol, RFC 3261
5. CAP Theorm : http://www.cs.berkeley.edu/~brewer/cs262b-2004/PODC-keynote.pdf
6. ITU-T, X.200 series recommendations: http://www.itu.int/rec/T-REC-X/en