Technical Disclosure Commons

Defensive Publications Series

November 01, 2018

DETERMINING NOMINAL QUALITY OF SERVICE NEEDS OF A DEVICE

Eliot Lear

Jerome Henry

Robert Barton

DETERMINING NOMINAL QUALITY OF SERVICE NEEDS OF A DEVICE

AUTHORS:
Eliot Lear
Jerome Henry
Robert Barton

ABSTRACT

Techniques are described herein for influencing network learning behaviors such as Quality of Service (QoS) using Manufacturer Usage Description (MUD) files. This mechanism improves the MUD QoS scheme by augmenting the dimensions of the QoS MUD component, and creating a new traffic mix hierarchy that converts the manufacturer expression of traffic importance into a locally significant MUD QoS hierarchy. This addition increases the security of MUD mechanisms by removing the possibility of Internet of Things (IoT) devices exploiting faults in allowed IoT protocols or servers. This mechanism also improves network allocation and planning by allowing different allocations based on traffic type and criticality.

DETAILED DESCRIPTION

As the number of types of Things proliferates on a network, many will require different types of network services. There is a need for the network administrator to be made aware of what sort of bandwidth such Things will use and what other Quality of Service (QoS) related factors they can be expected to require. Having that information available can permit provisioning of necessary QoS for particular devices as well as identify when a device is attempting to make use of more resources than one would expect of it (e.g., anomalous behavior).

Manufacturer Usage Description (MUD) allows a manufacturer to provide information about the expected Internet of Things (IoT) Thing needs, and a QoS component can be added to the MUD file. However, contrary to the security component, the QoS component of MUD may provide misleading information that would not reflect the real needs of the Thing in the local network. Techniques described herein fill that gap by providing an adaptive mechanism to localize the MUD QoS data.

Published by Technical Disclosure Commons, 2018

The basis of this approach is that the manufacturer knows what sort of network performance is likely to be required for the device to correctly function, and as importantly, what sort of network capabilities the device will not demand. The MUD architecture provides a solution to detail what type of access an object should be entitled to. This MUD file can contain a QoS component. A vending machine is likely to require very little QoS, if any. A fire alarm will require what it needs when it needs it. These techniques recognize that the manufacturer is in a good position to determine the device's network needs. The network need not only cover the Thing data content exchanges with the infrastructure, but may also include the other types of traffic (e.g., keepalives, configuration or firmware update exchanges, etc.). Each traffic type can be evaluated and described in a MUD, either built locally or provided by the manufacturer.

However, the manufacturer is unable to appreciate the importance, or criticality, of this particular Thing in comparison to other IoT objects in the same network. The manufacturer is also unable to evaluate the available network resources in a particular enterprise network, or determine what choice should be made when conflicting network resource needs arise. For example, what should happen if the control traffic of a vending machine conflicts (in terms of available network resources) with the control of traffic lights? Should fire extinguisher pressure status take precedence over fire door status reports? In this context, providing a MUD file with simple traffic hierarchy and description is not sufficient. Similarly, building a file from mere isolated device traffic observation is not sufficient.

This solution solves this issue by extending the intent, content, and effect of the QoS MUD file with parameters, and processing the MUD QoS file through a processing engine.

In general, the MUD file can include a specific location for QoS rules. The QoS provisions returned as part of the MUD files contain parameters descriptive of the expected IoT object network QoS behavior.

The techniques described herein expands this concept, and suggest a broader classification and description of the IoT objects and their traffic (instead of a description of only the expected traffic). In particular, this construct includes a device transmission category, device emergency group, and traffic type.

For a device transmission category, for example, the device may be labelled as C (continuous sender), expressing that the device will send a continuous flow of data (e.g., a video camera). Alternatively, the device type may be O (occasional sender), expressing that the device will send at stop sending at intervals. Subtypes are also envisioned (e.g., O-P (occasional but periodic), where the device data may be sent at known and specifiable intervals such as pressure values sent every five seconds; or O-T (occasional and threshold-based), where the device sends data when thresholds are reached). The category can apply to the entire device traffic, or a subset of that traffic. This extension is key to network planning.

For a device emergency group, clearly a fire detector is more critical than a humidity sensor (in most contexts). Organizing devices by criticality, or emergency types, along with the labels below, allows for better multi-dimensionality of network resource allocations.

For a traffic type, an IoT object may send or receive different types of traffic, such as management traffic (e.g., firmware updates, control of device configuration, keepalives, etc.) and data traffic. Each category may include subcategories with labels. These labels indicate more than application criticality or priority. This concept (application priority) can be determined by an IoT manufacturer, but cannot be easily translated into enterprise needs. For example, a vending machine "please refill me" message might be of highest criticality from the vending machine manufacturer standpoint (i.e., the most critical message type that the machine is expected to send), but may be of relatively low importance for the enterprise network from a resource allocation standpoint (e.g., if fire alarms battery level reports are competing for the same network resources). As such, labelling the traffic type is required before applying an importance hierarchy value to that traffic, so that the network administrator can apply hierarchical rules (e.g., "device emergency level n AND message type A is more important than device emergency level n+1 AND message type B").

For each traffic type, the traffic characteristics can be described, with elements such as burst size (bytes), burst duration, packets per seconds during burst (with min / max), tolerance to jitter or losses, or a concatenated bandwidth requirements value representing the burst. The traffic relative importance (from the manufacturer standpoint) can also be labelled with a Differentiated Services Code Point (DSCP) value.

The traffic description may integrate with the general MUD infrastructure (i.e., the described traffic may be matched against a MUD access rule, specifying the 5-tuple description of the traffic transmission within the MUD Access Control List (ACL)). An example is provided below.

Smart light:
82 bytes CS1 keepalive every 4 seconds, to controller on UDP port 5081
on trigger (virtual switch): 131 bytes once AF11 to controller on UDP port:5081
Management/configuration exchanges:
to and from controller 5080, triggered by controller one 237 byte packet BE, followed by burst of up to 40 pps / 54 kbps for up to 1 MB.

Fire alarm:
132 bytes status update every 12 seconds to controller on UDP port 1111 BE
155 bytes neighbor discovery once every 60 seconds to "local" on UDP port 1111, BE
155 byes neighbor discovery response, on trigger, BE
Upon trigger, 132 bytes update upon fire detection, EF, to controller on UDP 1111, repeated once per second.
Upon trigger, 165 bytes from controller on UDP port 1111 (clear alarm), repeated twice for 3 seconds total, EF

To effect these changes, an augmentation to the YANG MUD model is provided below. A new ACL in each direction is used to describe QoS behavior.

```
module mud-qos {
 yang-version 1.1;
 namespace "urn:ietf:params:xml:ns:yang:mud-qos";
 prefix mud-qos;
 import ietf-access-control-list {
  prefix acl;
 }
 import ietf-inet-types {
```

```
  prefix inet;
}
import ietf-mud {
  prefix mud;
}
organization
  "IETF OPSAWG (Ops Area) Working Group";
contact
  "WG Web: http://tools.ietf.org/wg/opsawg/
   WG List: opsawg@ietf.org
   Author: Eliot Lear
   Author: Jerome Henry
  ";
description
  "This YANG module augments the ietf-mud model to provide the
   network with some understanding as to the QoS requirements and
   anticipated behavior of a device.
   The to-device-policy and from-device-policy containers are
   augmented with one additional container, which expresses how many
   packets per second a device is expected to transmit, how much
   bandwidth it is expected to use, and what QoS is required, and
   how much bandwidth is to be expected to be prioritized.  An
   access-list is further specified to indicate how QoS should be
   marked on ingress and egress.
   Copyright (c) 2016,2017,2018 IETF Trust and the persons
   identified as the document authors.  All rights reserved.
   Redistribution and use in source and binary forms, with or
   without modification, is permitted pursuant to, and subject
   to the license terms contained in, the Simplified BSD
   License set forth in Section 4.c of the IETF Trust's Legal
   Provisions Relating to IETF Documents
```

```
   (http://trustee.ietf.org/license-info).
   This version of this YANG module is part of RFC XXXX; see
   the RFC itself for full legal notices.";
 revision 2018-03-01 {
  description
    "Initial proposed standard.";
  reference "RFC XXXX: QoS for MUD Specification";
 }
 grouping mud-qos-params {
  description
    "QoS and Bandwidth additions for MUD";
  container bw-params {
    description
      "Expected Bandwidth to/from device";
    leaf device-class {
      type enumeration {
        enum o;
        enum p;
        enum op;
        enum c;
      }
      description "Class of device - continuous or occasional speaker";
    }
    list service {
      key "name";
      description
        "a list of services that are being described.";
      leaf name {
        type string;
        description
          "Service Name";
```

```
}
leaf timeframe {
  type uint32;
  mandatory true;
  description
    "the period of time in seconds one
     expects a service to burst at described rates";
}
leaf pps {
  type uint32;
  description
    "number of packets per second to be expected.";
}
leaf bps {
  type uint64;
  description
    "number of bits per second to be expected.";
}
leaf dscp {
  type inet:dscp;
  description
    "The DSCP that packets for this service should
     treated with.  N.B., just because the manufacturer
     wants this, doesn't mean it will get it.  However,
     manufacturers who do set the DSCP value in their
     packets SHOULD indicate that in this description.
     This field differs from the dscp field in the matches
     portion of the access-list in that here the field is
     populated when the manufacturer states what the nominal
     value of the DSCP field MAY be, and how much bandwidth
     can be used when it is set.  Note that it is possible
```

```
            that the same service may use multiple DSCP values,
            depending on the circumstances.  In this case, service
            entry MUST be made.";
        }
        leaf aclname {
          type leafref {
            path "/acl:access-lists/acl:acl/acl:name";
          }
          description
            "The name of the ACL that will match packets
             for a given service.";
        }
      }
    }
  }
  augment "/mud:mud/mud:to-device-policy" {
    description
      "add inbound QoS parameters";
    uses mud-qos-params;
  }
  augment "/mud:mud/mud:from-device-policy" {
    description
      "add outbound QoS parameters";
    uses mud-qos-params;
  }
}
```

The second part of the techniques described herein is MUD intent localization and transformation. The QoS descriptions described herein augment and improve the MUD ACL by characterizing the allowed traffic in larger dimensions. However, merely receiving and applying the QoS MUD file is not sufficient. Contrary to security content of the MUD files (ACL), QoS entries do not express static rulesets. For example, “permit any to 1.1.1.1

port 5243, deny anything else" provides a simple security gating to block or allow traffic. By contrast, "please refill the vending machine" is the message with highest importance, and as such a DSCP CS5 tag only provides a relative ruleset that needs to be weighted and compared with all other messages from all other types of Things in the network. In other words, this message is of the highest importance in the view of the machine vendor, but may not have the same QoS characterization in view of other traffic mixes present in the network.

Therefore, the second step is to convert the relative intention in the MUD file provided by the manufacturer into the enterprise intent ruleset that balances all traffic and intents with its own importance hierarchy.

With this solution, the augmented QoS MUD files are fed into a Hierarchy Arbitration Tool (HAT). This engine may be a learning machine running standard regression in a multi-dimensional vector space represented by the various labels described above (each label and its range representing a dimension), a random forest structure, or a structured set of rules evaluating the labels in a hierarchical sequence.

Regardless of implementation details, the HAT relabels each Thing with various traffic types based on the relative presence of other Things with different emergency levels, so as to output a traffic hierarchy that is meaningful for the enterprise Thing mix.

For example, fire alarm battery status keepalives, in view of the other traffic present in the IoT network, may be relabeled from "importance level 2 on scale to 10, DSCP CS1" to "importance 5 on scale to 10, DSCP CS4" because such status is computed as more important than multiple other types of IoT traffic competing for the same resources. Similarly, the "Please fill the vending machine" alarm, initially labelled in the manufacturer MUD file as "importance level 5 on scale to 5, DSCP CS5," may be re-labelled "importance level 2 on scale to 10, DSCP AF11." At the end of HAT processing, new QoS MUD files emerge that do not only represent the traffic described by the various vendors, but also represent the relative importance of this traffic for the local network.

Figure 1 below illustrates how HAT dynamically recomputes the new set of MUD QoS rules as new object types are added or removed.

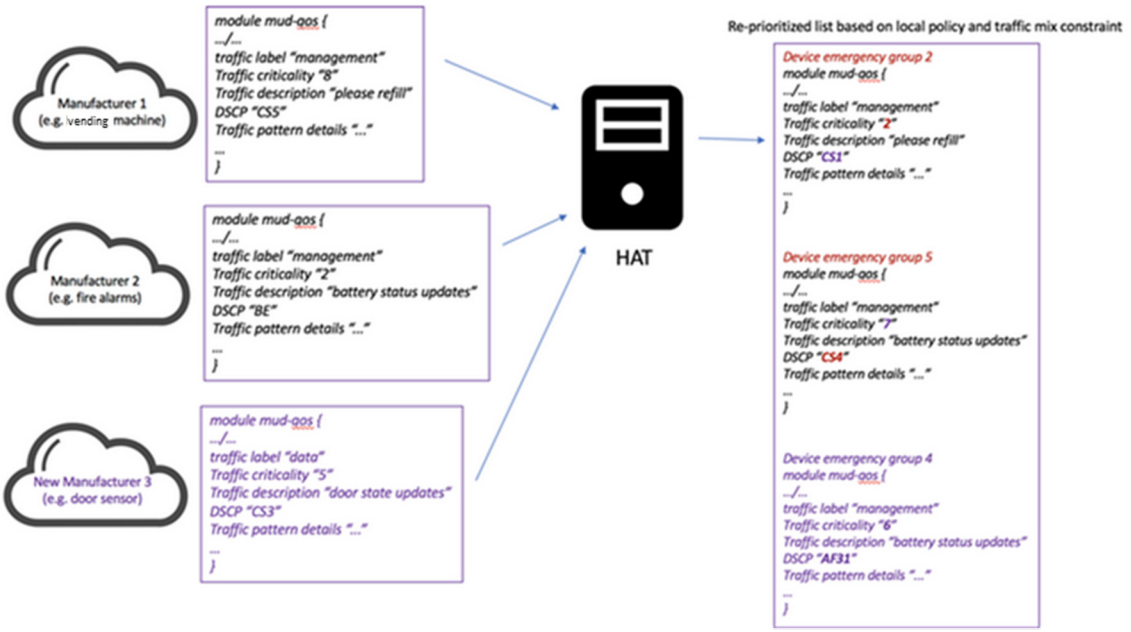

Figure 1

As more devices are added or removed, each with its associated MUD file, the system can dynamically rationalize the various conflicting intents to derive a hierarchical and organized intent in the form of a set of MUD QoS files.

It should be noted that the simplified re-classification example provided above focuses on two dimensions, but that all traffic description dimensions are evaluated in the process. For example, traffic volume or periodicity may also be taken into account to resolve prioritization conflicts between flows.

With this mechanism, a newly generated MUD QoS file set resolving all flows individual intents into a general intent structure can be inserted into a network allocation resource engine to dynamically assign network resources on the access and distribution connection points of the network. This allocation can be multi-dimensional.

For example, in 892.11ax networks, resources (802.11ax slots, or Resource Units (RUs)) can be allocated based on the expected traffic mix for the next time interval. In case of conflict (too much traffic for available RUs), device emergency group along with relative traffic importance and volume can be used to arbitrate the RU allocation.

For example, in an Ethernet network mix, the anticipation of the transmission of high criticality traffic may conduct a switch to drop incoming traffic of low importance, or

shape (delay) lower importance traffic to leave buffer space available for traffic of higher criticality.

This way, a fire alarm may be allocated a possible alarm transmission slot every 500ms, even when the network is congested, while a temperature sensor may be allocated only an occasional slot (during congestion) if its criticality is described as low. As illustrated in Figure 2 below, this mechanism ensures that congestion does not result in network resource starvation for high priority (high criticality) traffic and device types.

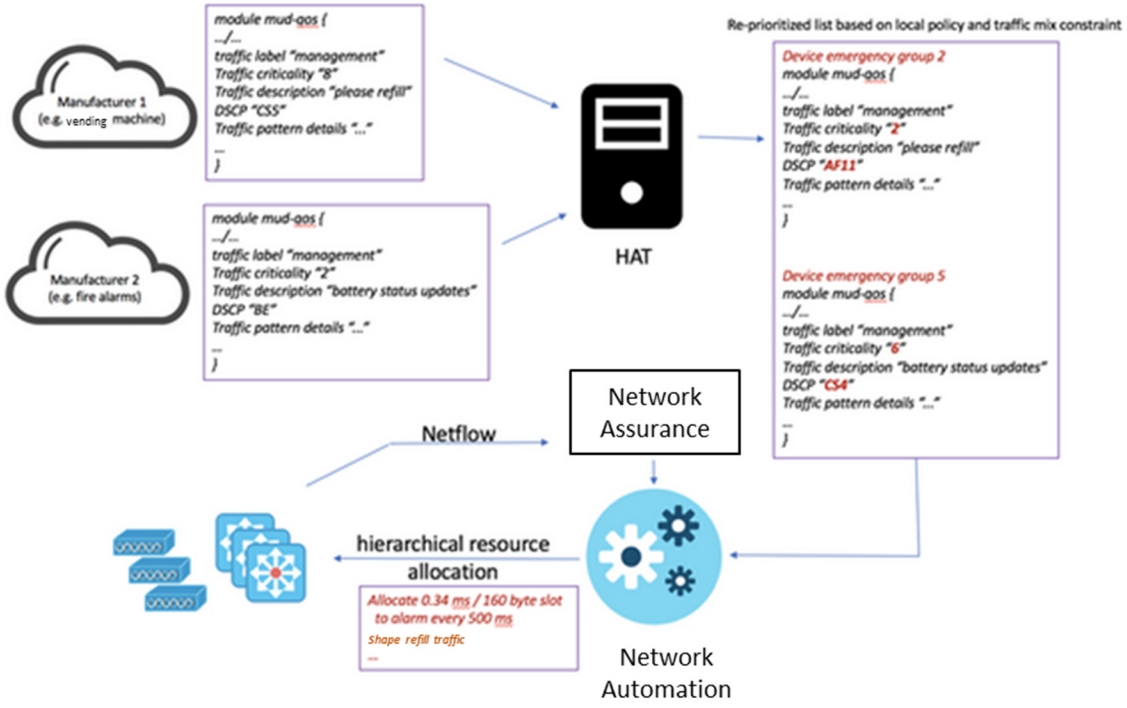

Figure 2

This mechanism allows for enhanced learning behaviors. Some Things can fail when expected traffic is not sent or received on time. With this submission, the effect of the new hierarchy is tested against the expected behavior of the Things. When the application of constraints to the Thing traffic results in unexpected behavior (e.g., device traffic is delayed because of the HAT ruleset, but then some of the device traffic stops then after the delay is applied), this fallibility or variability under stress is reported and integrated back into HAT, potentially to result in a different hierarchy mix. One goal here is not to classify the object in comparison to a database of known Things, but to evaluate the object traffic sensitivity to adverse network conditions.

This mechanism can be also be used to better protect the network against attacks. With the traditional MUD mechanism, the network is protected by limiting the device access to a set of rules defined by the MUD ACLs. As such, a compromised device is a limited attack vector, because the device cannot access Internet Protocol (IP) addresses and ports outside of those allowed by the MUD ruleset.

However, it is still possible for a compromised device to attempt attacks by using the IP addresses and ports allowed by the MUD file, for example by encapsulating attack packets within these allowed packets. The compromised device may this way exploit security weaknesses discovered within the allowed protocols or IoT application servers.

By adding QoS and traffic description parameters, and by learning the device behavior when traffic mix is set and QoS constraints are applied, a new device behavior baseline emerges, and thus the attack vector is limited further. Any traffic deviating from the expected traffic pattern is detected and flagged, and can result in the device being immediately quarantined (behavior deviation detected through strict ruleset matching).

In summary, techniques are described herein for influencing network learning behaviors such as QoS using MUD files. This mechanism improves the MUD QoS scheme by augmenting the dimensions of the QoS MUD component, and creating a new traffic mix hierarchy that converts the manufacturer expression of traffic importance into a locally significant MUD QoS hierarchy. This addition increases the security of MUD mechanisms by removing the possibility of IoT devices exploiting faults in allowed IoT protocols or servers. This mechanism also improves network allocation and planning by allowing different allocations based on traffic type and criticality.