June 2020

INTENT-BASED SERVICE AUTOMATION FOR 5G CORE NETWORK FUNCTION SLICING

Viktor Leijon

John Mullooly

Scott Wainner

Fredrik Jansson

INTENT-BASED SERVICE AUTOMATION FOR 5G CORE NETWORK FUNCTION SLICING

AUTHORS:

Viktor Leijon
John Mullooly
Scott Wainner
Fredrik Jansson

ABSTRACT

The techniques presented herein maintain network function (NF) resource adjacencies regardless of the shared or dedicated nature of the NF and independent of the segmentation of the NF's cluster resources across geographic management domains. To achieve this, the techniques use intent-based networking models in order to create NF resources that facilitate the network slice creation and resource allocation. More specifically, the techniques utilize a Yet Another Next Generation (YANG) model representation of a network slice to facilitate creation and state management of all NFs associated with a network slice instance. The YANG model enables inter-site and inter-cluster adjacencies where cloud-native container network interface (CNI) communications is invalid. This also provides reference points for inter-domain orchestration between the mobility NFs, the underlying data center switching resources, and wide area network (WAN) transport slices.

DETAILED DESCRIPTION

The Fifth Generation (5G) 3GPP standards have "disaggregated" the Mobile Packet Core Domain into many individual "Network Function" (NF) elements that are typically deployed individually in a cloud-native micro-service-based architecture, typically using Kubernetes. These NF elements include control plane functions such as the Session Management Function (SMF), the Policy Control Function (PCF), and the Access Management Function (AMF). A user plane gateway (called a User Plane Function (UPF)) is also a necessary element of the 5G Core Domain, but is potentially provided in a cloud native form-factor. That said, when these NFs are deployed, they can be deployed in the

same Data Center (DC) or in distributed DCs, and they need to be configured in a coherent manner in order to enable a working mobile service.

Moreover, if a "slicing" concept is utilized for the 5G Domain Core, the NFs can be deployed as either shared and/or dedicated NFs to satisfy a specific business purpose (e.g., meeting requirements from a service-level agreement (SLA)). For a service to operate, a slice will still need to have a full complement of NFs (even if some are shared across slices) and the slice NFs need to be configured to work together to provide that service.

Often, when migrating to a cloud-native micro-services architecture, a presumption is made that NFs will inherently discover their adjacencies through a container network interface (CNI) and a network resource function that serves to resolve end-points necessary for the adjacencies. However, a full set of micro-services is rarely contained within a Kubernetes cluster and, instead, the full set is often distributed across multiple clusters. Thus, partitioned adjacencies between the micro-services of distant clusters complicates NF configurations, especially when NF components are configured in a coherent manner that meets a slice's business purpose. Overall, the vast number of configuration permutations, the number of partitioned adjacencies, and cross-parameter assignments make creating a functional network service complex and challenging.

In view of the foregoing issues, some solutions focus on taking telemetry feedback from NF elements while the management system modifies the behavior of a network slice to try to satisfy an SLA. However, these solutions do not facilitate network slice creation and resource allocation and, instead, typically require a target SLA to be created by network slice instantiation and the NF resource configuration. Alternatively, some solutions try to use machine learning to select slice criteria (e.g., slice type, slice descriptor), but depend on an orchestration system to create/initialize one or more slices.

Still further, some solutions facilitate slice selection with dynamic assertion of different profiles on an embedded Subscriber Identity Module (eSIM). However, these solutions typically presume that the various slice types have been instantiated *a priori* and typically require the unified data management (UDM) entity and eSIM to be dynamically updated to facilitate network slice selection on a temporary basis. Yet other solutions try to create one network slice instance per user equipment (UE), thereby eliminating

orchestration and coordination between shared NFs, by collapsing all of the necessary 5G Core NF elements into a single package (e.g., a Kubernetes Pod) that is dedicated to a UE. However, this solution may be resource (e.g., memory, compute, network, and storage resources) intensive and require extensive management (e.g., requiring a dedicating user-agent and all of the necessary control NF for each UE).

In view of the foregoing issues, the techniques presented herein use declarative intent-based networking software principles and a service modeling approach to provide a 5G Core Domain slice "service" that meets a specific business purpose. The service model abstracts the complexity of the underlying NF provisioning requirements into operator facing business slice "primitives," such as:

- the need for a shared NF or dedicated NF per slice;
- the placement (locality) decision for a NF associated with a slice; and
- an "adjacency" based paradigm where each NF is associated with its supporting NF.

Consequently, the service model allows an operator to configure slices based on the business requirements present in the service model and underlying changes to the slice can be made by changing primitives exposed in the intent. The hierarchy of components (NFs) that comprise a slice service can easily be obtained, monitored, measured, and repaired as a single service construct, including any service assurance components that need to be deployed or configured as part of that slice.

The 5G Core Domain slice service model could then be used by a higher layer cross-domain orchestrator to provision a true end-to-end slice across other technology domains to provide the requested business service. Advantageously, since the declarative intent-based models allow delegation of intent to individual sub-systems, the same mapping principle can be reused at every level, simplifying the design and implementation. That is, the declarative intent-based models allow this orchestration in a simple manner that resolves or removes challenging complexities commonly generated in end-to-end slice provisioning.

Generally, network slice creation requires a complete set of 5G Core NFs that facilitate the identification, assignment, and state management of all the NFs that support the slice. Although network slice creation is often considered to be handled by a specific

NF dedicated to creating network slices, in actuality, the very first 5G network slice requires all of the NF elements. Thus, a 5G Core network cannot be built without the instantiation of the first network slice, i.e., a "bootstrap network slice." The bootstrap network slice initializes all the NFs that *might* be shared across all subsequent network slices. A workflow system can then be used to obtain attributes and requirements of subsequent business-specific slices that may leverage the bootstrap network slice resources or instantiate new NFs to further differentiate the business-specific slice from the bootstrap slice. The allocation of NF resources generally follows a hierarchical model where an AMF selects a PCF, a PCF scopes the potential SMF, and the SMF selects a UPF, while a UPF binds the UE Packet Data Unit (PDU) to a particular Data Network Name (DNN) and

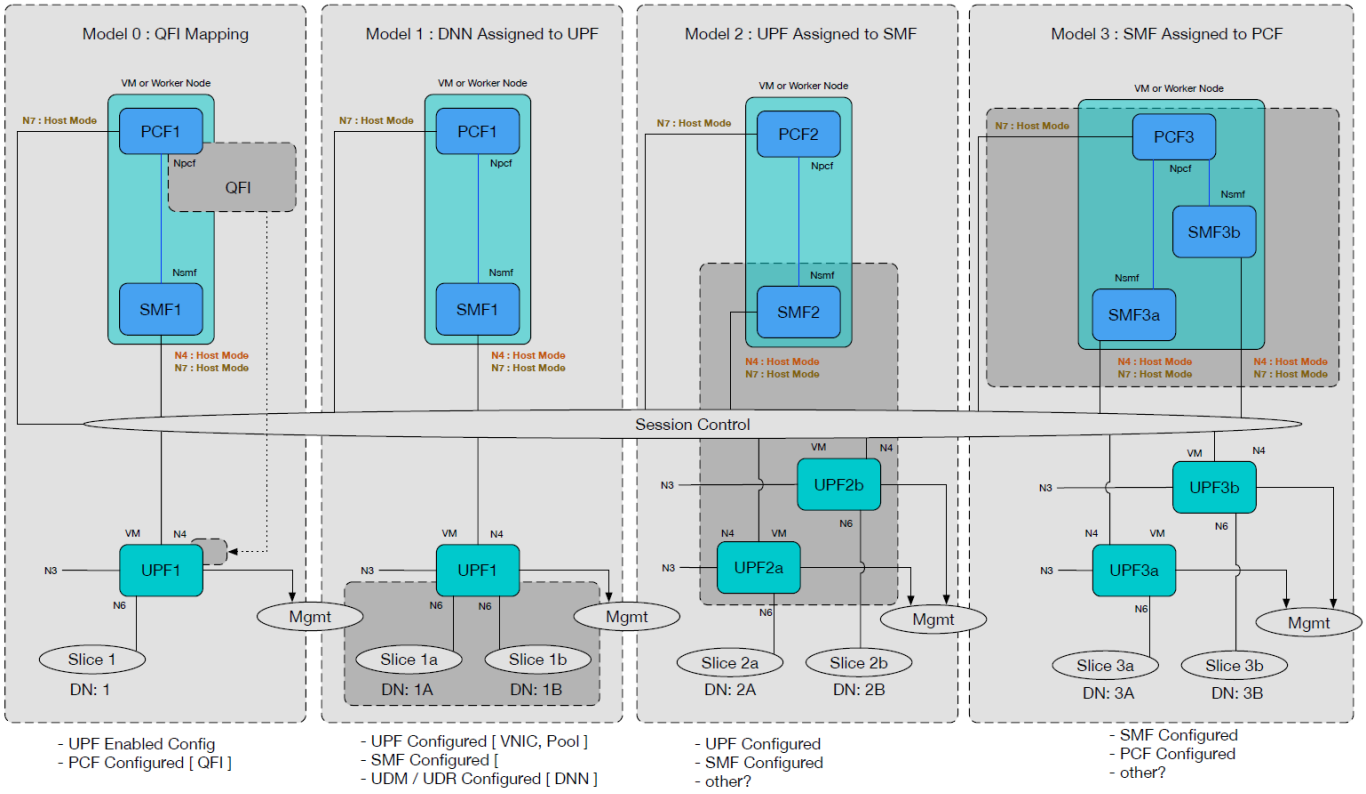
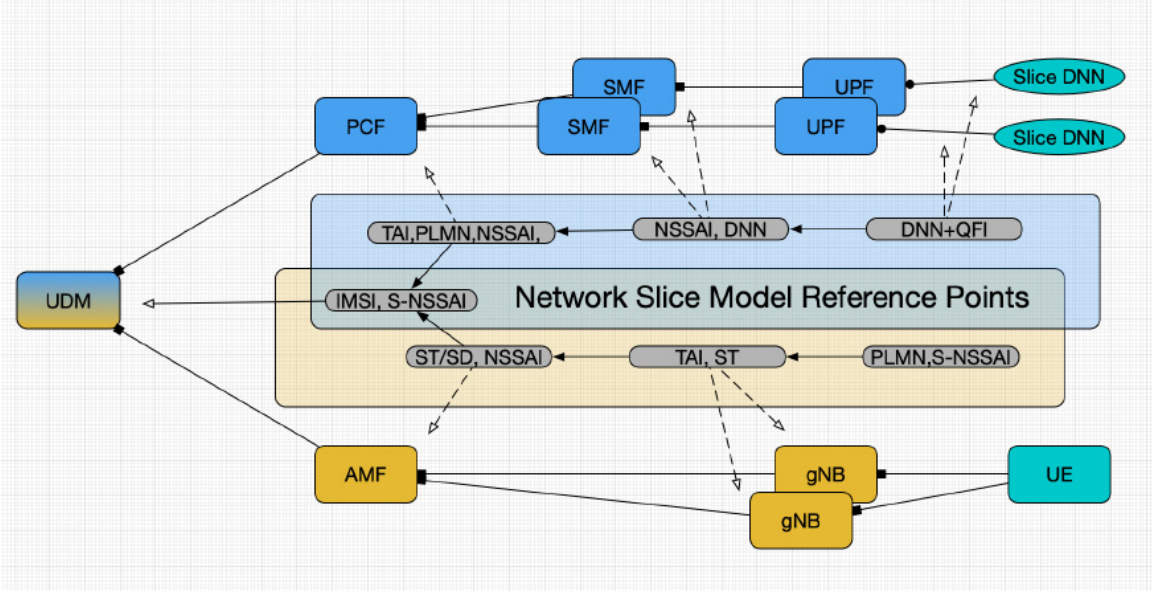


Figure 1

Quality of Service (QoS) Flow Identifier (QFI) context, as is shown in Figure 1 above.

The techniques presented herein model this hierarchical selection paradigm with a YANG model that identifies the adjacencies necessary to bind the NF into a cohesive network slice regardless of the location. The YANG model represents a potential state of a network slice instance (since a model may represent something that does not exist). Specifically, in the YANG model, network adjacencies are represented as YANG relationships between NFs. As instances of a network slice are created, the reference points configured in each NF are intrinsically linked to other NF predicated by the YANG model.



In modeling a network slice, there are two reference points - the model itself and instances of the model. In Figure 2 above, the model is represented as dotted-line representations of potential adjacencies while the instances (deployed network slices) represent specific adjacencies that have already been established and configured in the network:

To create this model, the addresses of specific NFs may be discovered dynamically through traditional Internet Protocol (IP) named network techniques such as DNS, Kubernetes tools (e.g., Etcd or Consul), and the 5G Core Network Repository Function (NRF) function. Moreover, although many of the NFs must be updated with the context of a network slice instance in order for the discovery procedures to function properly, the YANG model may represents all the necessary touch-points for the creation of a network slice instance across all the NFs. As an example, in an architecture including UE, Next Generation NodeB (gNB), AMF, Network Slice Selection Function (NSSF), PCF, SMF,

and UPF, the YANG model may represent updates to existing NFs to allow a new network slice or new NF to be created.

With the techniques presented herein, the optimal distribution of the NF are predicated on the given business model. Thus, NFs may be distributed across multiple sites and multiple clusters and, in fact, may be distributed across multiple clusters within a site in order to accommodate vendor-specific NF orchestration requirements. Thus, NFs will involve inter-site (and inter-cluster) communications. The intent-based YANG representation of the network slice adjacencies presented herein may enable such communications by referencing NF anchor points regardless of their location to provide reference points across multiple clusters and multiple sites.

As one example, NFs may communicate using an inherent CNI communication paradigm (the cloud-native model of 5G Core NF implies that CNI may be inherent). The CNI offers several modes of instantiating NF interfaces that are generally appropriate for application functions; however, two of the cloud-native 5G Core NFs present interfaces that may not congruent with the cloud-native networking paradigm: the AMF and the SMF. The AMF presents an N2 interface (e.g., a Stream Control Transmission Protocol (SCTP)/IP interface) to the network resources of the radio access network (RAN). Meanwhile, the SMF presents an N4 interface (e.g., a User Datagram Protocol (UDP)/Internet Protocol (IP) interface) to UPF network resources. While these protocol adjacencies might be made cloud-native (e.g., Hypertext Transfer Protocol (HTTP)), they often remain discrete interfaces in the creation of the NFs. The remaining interfaces of the NFs presumably use the cloud-native interfaces of the CNI cluster and, thus, the NF anchor points referenced regardless of their location may allow inter-site (and inter-cluster) communications.

As a more specific example, if a PCF is instantiated and provides services for a set of network slices (e.g., software defined (SD)-WAN business services), the PCF instance becomes a reference point for that set of network slices. The set of network slices associated with the business intent may use a distributed set of SMF resources across multiple sites, thereby providing inherent inter-site/inter-cluster communication relation defined as the N7 interface. Presumably, the SMF-PCF N7 interface uses a cloud-native CNI communication paradigm, but often NFs are disjoint in the network. Thus, with the

techniques presented herein, the PCF centralized N7 interface becomes a reference point that is populated in the YANG model and subsequently referenced by the distributed set of SMF across the network. To enable the successful discovery of the PCF N7 reference point, the set of SMFs and their associated NRF must be updated to facilitate the N7 adjacency. The network slice YANG model defines these touch points (SMF config, NRF config, PCF config) across all the NFs such that the discovery mechanisms will succeed.

Moreover, the techniques presented herein may facilitate network slice management with at least two approaches. The first approach uses a consumption model where the network slice resources are created prior to a UE attachment. Specific service locations are identified and the UE is assigned to the 'best' NF resources during the Non-Access Stratum (NAS) procedures. The second approach uses a dynamic assertion of network resources at the time of NAS. This requires the network slicing YANG model to describe the candidate locations of the network function resources and their current state of availability.

At the time UE NAS attachment, the authentication procedures between AMF and the authentication server function (AUSF) / UDM trigger the orchestration system to instantiate the appropriate network resources in the optimal location. Currently, the time to instantiate the NF resources may preclude real-time assertion and subsequent attachment of the UE with a PDU to a UPF. However, in the future, creating a UPF container may allow the on-demand creation of a network slice instance. In either case, the YANG model for the network slice can describe the reference points to which the creation of a UPF must attach. Subsequent UE attachments to the same slice type will reference the existing network slice instance. Moreover, logic can be created to deprecate the network slice resources as UEs vacate the network slice. In doing so, the YANG model knows all the resources that can be destroyed using a 'last reference' counter.

As a specific example, Appendix A illustrates an example YANG model for the potential structure of "amf-slice/pcf-slice/smf-slice/upf-slice" and describes resource assignment that may be built in the network. Meanwhile, Appendix B illustrates an example YANG tree structure of a network slice. As can be seen in these appendices, the UE can only establish a PDU session to an existing resource allocated in the network when it identifies the resource by the S-NSSAI (Specific - Network Slice Selection Assistance

Information). However, the S-NSSAI may be associated with a model representation for a set of resources that do not yet exist in the network.

If a network slice instance does not already exist, TAI (Tracking Area Identifier) may be used during the NAS (Network Attachment Service) to identify where the network slice instance should be placed. The TAI and S-NSSAI can then trigger the instantiation of the network slice instance such that a PDU session can be established. Alternatively, when there is a UE attachment using the NAS procedures, either a network slice YANG model or an instance derived from the YANG model can be used. Often, the S-NSSAI is preconfigured in the network (i.e. a network slice instance derived from the YANG model); however, if it is not, the YANG model may be utilized because it represents a 'future state' of an S-NSSAI request.

If a complete network slice instance is established, the UE can attach to that instance using the S-NSSAI and simply consume the resources already allocated. This method is readily implemented today by statically assigning resources, configuring the necessary adjacencies, publishing the resources, and associating the UE's request to those resources. The network slice instance represents the complete state of the resources allocated and potentially consumed by a UE's request in the future. With the techniques presented herein, the instance of the YANG model represents the candidate network slice instances to which a UE may consume and S-NSSAI is mapped to a network slice instance derived from the YANG model.

If a network slice instance is intended to be created as a result of a UE's NAS procedure, the UE can attach to that future state network slice instance using an S-NSSAI that is associated with the network slice YANG model (as opposed to a preconfigured network slice instance). The NAS triggers the instantiation of a network slice instance based on the YANG model, which represents all the dependencies and adjacencies required to create a network slice instance. Thus, the NAS procedures actually trigger the instantiation of a network slice instance in accordance to the model and bind the UE to the resources created.

Overall, at a high-level, the YANG model can be used for at least two tasks. First, the YANG model can create a network slice instance (created from a YANG model) that represents a future state of a UE attachment (Specific Network Slice Selection Assistance

Information (S-NSSAI)). As is discussed above, the NF anchor points in the YANG model representation represent network attachment points that may require segmentation across the network and intra-site and inter-site communications may maintain the segmentation requirements of a network slice instance. Thus, the YANG model can provide a mechanism for establishing the intent of the future state within the context of a 'transport slice.' Among other advantages, this may facilitate slice protection between NF regardless of their location in the network.

For example, the model may allow Internet NF resources to be segmented from business class NF resources. The economics associated with the consumption of the Internet network slice will be vastly different than the economics associated with consumption of resources for enterprise business services. The YANG model representing the Internet slice and the business class slice will delineate between mobility NFs while also requiring segmented network resources in the WAN (e.g. Segment Routed Traffic Engineering) or local area network (LAN) (e.g. ACI policy-based models). The inter-domain orchestration model can link the YANG model of the mobility domain to the YANG model of the data center switching domain and subsequently to the WAN transport domain. Notably, in the absence of a defining a mobility network slice YANG model, manual configuration would required for every inter-domain and inter-site adjacency, which would drastically complicate the instantiation of a network slice and likely precludes the on-demand creation of a network slice.

Second, in addition or as an alternative to creating network slice instances, a network slice YANG model can represent NSSAI where a set of S-NSSAI may attach to a future state of a network slice instance. Notably, this may lead to a self-configuring network based on intent modeled in YANG. Moreover, the first and second aspects described herein are congruent because, when used together, these two aspects may represent two states of network - future state and current state.

In sum, the techniques presented herein maintain network function (NF) resource adjacencies regardless of the shared or dedicated nature of the NF and independent of the segmentation of the NF's cluster resources across geographic management domains. To achieve this, the techniques use intent-based networking models in order to create NF resources that facilitate the network slice creation and resource allocation. More

specifically, the techniques utilize a YANG model representation of a network slice to facilitate creation and state management of all NFs associated with a network slice instance. The YANG model enables inter-site and inter-cluster adjacencies where cloud-native CNI communications is invalid. It also provides reference points for inter-domain orchestration between the mobility NFs, the underlying data center switching resources, and WAN transport slices.

APPENDIX A

EXAMPLE YANG MODEL OF A NETWORK SLICE

```
module mobility-app {
  namespace "http://cisco.com/mobility-app";
  prefix mobility-app;
  yang-version 1.1;

  import ietf-inet-types {
    prefix inet;
  }
  import tailf-common {
    prefix tailf;
  }
  import tailf-ncs {
    prefix ncs;
  }

  description
    "Bla bla...";

  revision 2018-11-20 {
    description
      "Initial revision.";
  }

  typedef octet-string8 {
    type string {
      pattern
        '[0-9a-fA-F]{2}';
    }
  }

  typedef octet-string24 {
    type string {
      pattern
        '[0-9a-fA-F]{6}';
    }
  }

  identity vm {
    description
      "Plan component used by the Dep service";
    base ncs:plan-component-type;
  }

  identity kubernetes {
    description
      "Kuberneters deployed";
    base ncs:plan-component-type;
  }

  identity application {
    description
      "Mobility application";
    base ncs:plan-component-type;
  }

  identity k8s-cluster-available {
    description "Application cluster is available";
    base ncs:plan-state;
  }

  identity deployed {
    description "Application has been deployed in the k8s cluster";
    base ncs:plan-state;
  }

  identity onboarded {
    description "Application has been deployed in the k8s cluster";
    base ncs:plan-state;
  }

  grouping engine-list {
    list engine {
      key name;
      min-elements 1;
      leaf name {
        type string;
      }

      leaf replicas {
        type uint32;
```

```
      default 1;
    }

    choice engine-repository {
      mandatory true;
      leaf repository {
        type inet:uri;
      }
      leaf default-repository {
        type empty;
      }
    }
  }
}

grouping base-app-dep-settings {
  leaf repository {
    mandatory true;
    tailf:info "Application repository";
    type inet:uri;
  }

  leaf k8s-cluster {
    mandatory true;
    type leafref {
      path "/mobility/k8s-cluster/name";
    }
  }
}

grouping application-dep-settings {
  container deployment {
    uses base-app-dep-settings;

    leaf namespace {
      type string;
    }

    leaf netconf-port {
      mandatory true;
      type inet:port-number;
    }

    leaf deployed {
      type empty;
      config false;
      tailf:cdb-oper {
        tailf:persistent true;
      }
    }

    leaf onboarded {
      type empty;
      config false;
      tailf:cdb-oper {
        tailf:persistent true;
      }
    }

    leaf active {
      type empty;
      config false;
      tailf:cdb-oper {
        tailf:persistent true;
      }
    }

    action deploy {
      tailf:actionpoint "mobility-app-deploy-app";
      input {
        leaf force {
          type empty;
        }
      }
    }

    action onboard {
      tailf:actionpoint "mobility-app-onboard-app";
      input {
        leaf force {
          type empty;
```

```
        }
      }
    }
  }
}

grouping rest-endpoint {
  container rest-endpoint {
    leaf address {
      mandatory true;
      type inet:ip-address;
    }
    leaf port {
      mandatory true;
      type inet:port-number;
    }
  }
}

grouping upf-network {
  leaf address {
    mandatory true;
    type inet:ip-address;
  }

  leaf netmask {
    mandatory true;
    type inet:ip-address;
  }

  leaf gateway {
    mandatory true;
    type inet:ip-address;
  }
}

grouping tai-group-list {
  list tai-group {
    min-elements 1;
    key "name";
    leaf name {
      type string;
    }

    list mcc-mnc {
      min-elements 1;
      key "mcc mnc";
      leaf mcc {
        type uint32;
      }
      leaf mnc {
        type uint32;
      }
      list tac {
        min-elements 1;
        key code;
        leaf code {
          type uint32;
        }
        leaf name {
          type string;
        }
      }
    }
  }
}

grouping sst-sdt-list {
  list sst-sdt {
    key name;
    leaf name {
      type string;
    }

    leaf sst {
      mandatory true;
      type octet-string8;
    }

    leaf sdt {
      mandatory true;
```

```
      type octet-string24;
    }
  }
}

grouping upf-network-settings {
  leaf hostname {
    tailf:info "End-user Desired UPF Host Name";
    type string {
      length "1..24";
    }
  }

  container n3 {
    uses upf-network;
  }

  container n4 {
    uses upf-network;
    leaf port {
      // mandatory true;
      tailf:info "Port the SMF will connect to";
      type inet:port-number;
      default 8805;
    }
  }

  leaf as-number {
    mandatory true;
    type uint32;
  }

  leaf pe-bgp-as-number {
    mandatory true;
    type uint32;
  }


  container n6 {
    uses upf-network;
  }

  leaf gi-server-loopback {
    tailf:info "IPv4 address for Server emulation Loopback interface, /32 NetMask ";
    type inet:ip-address;
    default "34.34.34.34";
  }
}

container mobility {

  container applications {
    list cnee {
      key "name";

      uses ncs:service-data;
      ncs:servicepoint "mobility-app-cnee";
      uses ncs:plan-data;

      leaf name {
        type string;
      }

      uses application-dep-settings;
    }
    list amf {
      uses ncs:service-data;
      ncs:servicepoint "mobility-app-amf";
      uses ncs:plan-data;
      key "name";
      leaf name {
        type string;
      }

      uses application-dep-settings {
        augment deployment {
          leaf cnee {
            mandatory true;
            type leafref {
              path "../../../cnee/name";
            }
```

```
    }
  }
}

uses rest-endpoint;

leaf nrf {
  mandatory true;
  type leafref {
    path "../../nrf/name";
  }
}

leaf nssf {
  mandatory true;
  type leafref {
    path "../../nssf/name";
  }
}

container pcf-discover-method {
  leaf plmn {
    type empty;
  }
  leaf dnn {
    type empty;
  }
  leaf slice {
    type empty;
  }
  must "plmn | dnn | slice";
}

container smf-discover-method {
  leaf plmn {
    type empty;
  }
  leaf dnn {
    type empty;
  }
  leaf slice {
    type empty;
  }
  must "plmn | dnn | slice";
}

uses tai-group-list;

container ausf {
  choice discover {
    container rest-endpoint {
      leaf address {
        mandatory true;
        type inet:ip-address;
      }
      leaf port {
        mandatory true;
        type inet:port-number;
      }
    }
    container discover-method {
      leaf plmn {
        mandatory true;
        type empty;
      }
    }
  }
}

container udm {
  choice discover {
    container rest-endpoint {
      leaf address {
        mandatory true;
        type inet:ip-address;
      }
      leaf port {
        mandatory true;
        type inet:port-number;
      }
    }
```

```
      container discover-method {
        leaf plmn {
          mandatory true;
          type empty;
        }
      }
    }

    container sctp {
      leaf ip-address {
        mandatory true;
        type inet:ip-address;
      }
      leaf port {
        type inet:port-number;
        default 1000;
      }
      leaf k8-node-hostname {
        mandatory true;
        type string;
      }
    }

    uses sst-sdt-list;

  }

  list nrf {
    uses ncs:service-data;
    ncs:servicepoint "mobility-app-nrf";
    uses ncs:plan-data;
    key "name";
    leaf name {
      type string;
    }

    uses application-dep-settings {
      augment deployment {
        leaf cnee {
          mandatory true;
          type leafref {
            path "../../../cnee/name";
          }
        }
      }
    }

    uses rest-endpoint;
    uses engine-list;
  }

  list nssf {
    uses ncs:service-data;
    ncs:servicepoint "mobility-app-nssf";
    uses ncs:plan-data;
    key "name";
    leaf name {
      type string;
    }

    leaf amf {
      mandatory true;
      type leafref {
        path "../../amf/name";
      }
    }

    uses application-dep-settings {
      augment deployment {
        leaf cnee {
          mandatory true;
          type leafref {
            path "../../../cnee/name";
          }
        }
      }
    }

    uses rest-endpoint;
    uses engine-list;
```

```
}

list pcf {
  uses ncs:service-data;
  ncs:servicepoint "mobility-app-pcf";
  uses ncs:plan-data;
  key "name";
  leaf name {
    type string;
  }

  uses rest-endpoint;
  uses engine-list;

  leaf amf {
    mandatory true;
    type leafref {
      path "../../amf/name";
    }
  }

  list tai-group {
    min-elements 1;
    key name;
    leaf name {
      type leafref {
        path "deref(../../amf)/../tai-group/name";
      }
    }

    list mcc-mnc {
      min-elements 1;
      key "mcc mnc";
      leaf mcc {
        type leafref {
          path "deref(../../name)/../mcc-mnc/mcc";
        }
      }

      leaf mnc {
        type leafref {
          path "deref(../mcc)/../mnc";
        }
      }
    }
  }

  list sst-sdt {
    key "name";
    leaf name {
      type leafref {
        path "deref(../../amf)/../sst-sdt/name";
      }
    }
  }

  uses application-dep-settings {
    augment deployment {
      leaf cnee {
        mandatory true;
        type leafref {
          path "../../../cnee/name";
        }
      }
    }
  }
}

list smf {
  uses ncs:service-data;
  ncs:servicepoint "mobility-app-smf";
  uses ncs:plan-data;
  key "name";
  leaf name {
    type string;
  }

  uses application-dep-settings {
    augment deployment {
      leaf cnee {
        mandatory true;
```

```
      type leafref {
        path "../../../cnee/name";
      }
    }
  }
}
uses rest-endpoint;

leaf pcf {
  mandatory true;
  type leafref {
    path "../../pcf/name";
  }
}

list sst-sdt {
  key "name";
  leaf name {
    type leafref {
      path "deref(../../pcf)/../sst-sdt/name";
    }
  }
}

container udm {
  leaf address {
    mandatory true;
    type inet:ip-address;
  }
  leaf port {
    mandatory true;
    type inet:port-number;
  }
}

list dnn {
  key name;
  leaf name {
    type string;
  }

  list ipv4-pool {
    key name;
    leaf name {
      type string;
    }

    leaf prefix {
      mandatory true;
      type inet:ipv4-prefix;
    }

    container ip-range {
      leaf start {
        mandatory true;
        type inet:ipv4-address;
      }
      leaf end {
        mandatory true;
        type inet:ipv4-address;
      }
    }
    leaf vrf {
      mandatory true;
      type string;
    }
  }
  list ipv6-pool {
    key name;
    leaf name {
      type string;
    }

    leaf prefix {
      mandatory true;
      type inet:ipv6-prefix;
    }

    leaf vrf {
      mandatory true;
      type string;
```

```
      }

      leaf prefix-lifetime {
        type uint32;
      }
    }
  }

  list smf {
    key name;
    leaf name {
      type string;
    }

    list service {
      key name;
      leaf name {
        type string;
      }

      leaf bind-address {
        mandatory true;
        type inet:ip-address;
      }

    }
  }

  container profile {
    container protocol {
      leaf external-address {
        type inet:ip-address;
      }
      leaf node-label {
        mandatory true;
        type string;
      }
    }
  }

  container tracing-endpoint {
    leaf host {
      type string;
    }
    leaf port {
      type inet:port-number;
    }
  }
}

list upf {
  uses ncs:service-data;
  ncs:servicepoint "mobility-app-upf";
  uses ncs:plan-data;
  key "name";
  leaf name {
    type string;
  }

  leaf smf {
    mandatory true;
    type leafref {
      path "../../smf/name";
    }
  }

  leaf smf-profile {
    mandatory true;
    type leafref {
      path "deref(../smf)/../smf/name";
    }
  }

  leaf smf-service {
    mandatory true;
    type leafref {
      path "deref(../smf-profile)/../service/name";
    }
  }

  list dnn {
```

```
      key name;
      leaf name {
        type leafref {
          path "deref(../../smf)/../dnn/name";
        }
      }
    }

    leaf device {
      // mandatory true;
      type string;
    }

    uses upf-network-settings;
  }

  list waiting-for-deletion {
    key device;
    leaf device {
      type leafref {
        path "/ncs:devices/ncs:device/ncs:name";
      }
    }

    leaf url {
      mandatory true;
      type inet:uri;
    }

    action delete {
      tailf:actionpoint "mobility-app-delete-app";
    }

    leaf try-count {
      type uint8;
      default 0;
      config false;
      tailf:cdb-oper {
        tailf:persistent true;
      }
    }
    leaf error-message {
      config false;
      type string;
      tailf:cdb-oper {
        tailf:persistent true;
      }
    }
  }

  action restart {
    tailf:actionpoint 'mobility-app-restart';
  }
}

tailf:unique-selector "amf-slice/pcf-slice" {
  tailf:unique-leaf "name";
}

tailf:unique-selector "amf-slice/pcf-slice/smf-slice" {
  tailf:unique-leaf "name";
}

tailf:unique-selector "amf-slice/pcf-slice/smf-slice/upf-slice" {
  tailf:unique-leaf "name";
}

list amf-slice {
  uses ncs:service-data;
  ncs:servicepoint "mobility-app-slice";
  uses ncs:plan-data;

  key name;
  leaf name {
    type string;
  }

  container cnee {
    uses base-app-dep-settings;
  }
```

```
container amf {
  uses base-app-dep-settings;
  leaf rest-address {
    mandatory true;
    type leafref {
      path "deref(../k8s-cluster)/../worker/address";

    }
  }
}

container nrf {
  uses base-app-dep-settings;
  leaf rest-address {
    mandatory true;
    type leafref {
      path "deref(../k8s-cluster)/../worker/address";

    }
  }
}

container nssf {
  uses base-app-dep-settings;
  leaf rest-address {
    mandatory true;
    type leafref {
      path "deref(../k8s-cluster)/../worker/address";
    }
  }
}

uses tai-group-list;
uses sst-sdt-list;

list pcf-slice {
  key name;
  leaf name {
    type string;
  }

  uses base-app-dep-settings;
  leaf rest-address {
    mandatory true;
    type leafref {
      path "deref(../k8s-cluster)/../worker/address";

    }
  }

  list tai-group {
    min-elements 1;
    key name;

    leaf name {
      type leafref {
        path "../../../tai-group/name";
      }
    }
    list mcc-mnc {
      min-elements 1;
      key "mcc mnc";
      leaf mcc {
        type leafref {
          path "deref(../../name)/../mcc-mnc/mcc";
        }
      }

      leaf mnc {
        type leafref {
          path "deref(../mcc)/../mnc";
        }
      }
    }
  }

  list sst-sdt {
    // min-elements 1;
    key name;

    leaf name {
```

```
        type leafref {
          path "../../../sst-sdt/name";
        }
      }
    }

    list smf-slice {
      key name;
      leaf name {
        type string;
      }

      uses base-app-dep-settings;
      leaf rest-address {
        mandatory true;
        type leafref {
          path "deref(../k8s-cluster)/../worker/address";

        }
      }

      list sst-sdt {
        min-elements 1;
        key name;

        leaf name {
          type leafref {
            path "../../../sst-sdt/name";
          }
        }
      }

      list dnn {
        key name;
        leaf name {
          type string;
        }

        leaf network {
          mandatory true;
          type inet:ip-prefix;
          // type uint8 {
          //   range "8..30";
          // }
        }

        leaf vrf {
          mandatory true;
          type string;
        }
      }

      list upf-slice {
        key name;
        leaf name {
          type string;
        }

        list dnn {
          key name;

          leaf name {
            type leafref {
              path "../../../dnn/name";
            }
          }
        }

        leaf management-address {
          mandatory true;
          type inet:ip-address;
        }

        uses upf-network-settings;
      }
    }
  }
}

list k8s-cluster {
  key "name";
```

```
      leaf name {
        type string;
      }

      leaf k8s-master {
        tailf:info "domain name/address of the k8s master";
        mandatory true;
        type inet:uri;
      }

      leaf base-uri {
        tailf:info "The base URI for the mobilty API";
        mandatory true;
        type inet:uri;
      }

      container api-credentials {
        presence "Set API access credentials";
        leaf username {
          mandatory true;
          type string;
        }
        leaf password {
          mandatory true;
          type string;
        }
      }

      leaf application-yaml-directory {
        mandatory true;
        type inet:uri;
      }

      leaf netconf-port-pool {
        type string;
      }

      list worker {
        key address;

        leaf address {
          type inet:ip-address;
        }

        leaf hostname {
          type string;
        }

        leaf label {
          type string;
        }
      }

      leaf netconf-address {
        tailf:info "The applications will listen on this address";
        mandatory true;
        type inet:ip-address;
      }

      leaf cluster-available {
        type empty;
        config false;
        tailf:cdb-oper {
          tailf:persistent true;
        }
      }

      action check-cluster-availability {
        tailf:actionpoint "mobility-app-check-k8s-cluster";
        output {
          leaf api-available {
            type boolean;
          }
        }
      }
    }
  }
}
```

APPENDIX B

EXAMPLE YANG TREE STRUCTURE OF A NETWORK SLICE

```
module: mobility-app
  +--rw mobility-application
  |  +--rw k8s-deployment* [name]
  |     +--rw name                          string
  |     +--rw esc                           -> /ncs:devices/device/name
  |     +--rw k8s-deployer                  -> /ncs:devices/device/name
  |     +--rw tenant?                       string
  |     +--rw http-proxy?                   inet:uri
  |     +--rw network-1-name?               string
  |     +--rw network-2-name?               string
  |     +--rw network-3-name?               string
  |     +--rw network-4-name?               string
  |     +--rw vm* [name]
  |        +--rw name             string
  |        +--rw network-1-ip     inet:ip-address
  |        +--rw network-2-ip?    inet:ip-address
  |        +--rw network-3-ip?    inet:ip-address
  |        +--rw network-4-ip?    inet:ip-address
  |        +--rw type*            enumeration
  |        +--rw image-name       string
  |        +--rw flavor?          string
  +--rw mobility
     +--rw applications
     |  +--rw cnee* [name]
     |  |  +--rw name                          string
     |  |  +--rw deployment
     |  |     +--rw repository      inet:uri
     |  |     +--rw k8s-cluster     -> /mobility/k8s-cluster/name
     |  |     +--rw namespace?      string
     |  |     +--rw netconf-port    inet:port-number
     |  |     +--ro deployed?       empty
     |  |     +--ro onboarded?      empty
     |  |     +--ro active?         empty
     |  +--rw amf* [name]
     |  |  +--rw name                          string
     |  |  +--rw deployment
     |  |  |  +--rw repository      inet:uri
     |  |  |  +--rw k8s-cluster     -> /mobility/k8s-cluster/name
     |  |  |  +--rw namespace?      string
     |  |  |  +--rw netconf-port    inet:port-number
     |  |  |  +--ro deployed?       empty
     |  |  |  +--ro onboarded?      empty
     |  |  |  +--ro active?         empty
     |  |  |  +--rw cnee            -> ../../../cnee/name
     |  |  +--rw rest-endpoint
     |  |  |  +--rw address    inet:ip-address
     |  |  |  +--rw port       inet:port-number
     |  |  +--rw nrf                           -> ../../nrf/name
     |  |  +--rw nssf                          -> ../../nssf/name
     |  |  +--rw pcf-discover-method
     |  |  |  +--rw plmn?    empty
     |  |  |  +--rw dnn?     empty
     |  |  |  +--rw slice?   empty
     |  |  +--rw smf-discover-method
     |  |  |  +--rw plmn?    empty
     |  |  |  +--rw dnn?     empty
     |  |  |  +--rw slice?   empty
     |  |  +--rw tai-group* [name]
     |  |  |  +--rw name       string
     |  |  |  +--rw mcc-mnc* [mcc mnc]
     |  |  |     +--rw mcc    uint32
     |  |  |     +--rw mnc    uint32
     |  |  |     +--rw tac* [code]
     |  |  |        +--rw code    uint32
     |  |  |        +--rw name?   string
     |  |  +--rw ausf
     |  |  |  +--rw (discover)?
     |  |  |     +--:(rest-endpoint)
     |  |  |     |  +--rw rest-endpoint
     |  |  |     |     +--rw address    inet:ip-address
     |  |  |     |     +--rw port       inet:port-number
     |  |  |     +--:(discover-method)
     |  |  |        +--rw discover-method
     |  |  |           +--rw plmn    empty
     |  |  +--rw udm
     |  |  |  +--rw (discover)?
     |  |  |     +--:(rest-endpoint)
     |  |  |     |  +--rw rest-endpoint
     |  |  |     |     +--rw address    inet:ip-address
     |  |  |     |     +--rw port       inet:port-number
```

```
|  |  |     +--:(discover-method)
|  |  |        +--rw discover-method
|  |  |           +--rw plmn    empty
|  |  +--rw sctp
|  |  |  +--rw ip-address          inet:ip-address
|  |  |  +--rw port?               inet:port-number
|  |  |  +--rw k8-node-hostname    string
|  |  +--rw sst-sdt* [name]
|  |     +--rw name    string
|  |     +--rw sst     octet-string8
|  |     +--rw sdt     octet-string24
|  +--rw nrf* [name]
|  |  +--rw name                          string
|  |  +--rw deployment
|  |  |  +--rw repository      inet:uri
|  |  |  +--rw k8s-cluster     -> /mobility/k8s-cluster/name
|  |  |  +--rw namespace?      string
|  |  |  +--rw netconf-port    inet:port-number
|  |  |  +--ro deployed?       empty
|  |  |  +--ro onboarded?      empty
|  |  |  +--ro active?         empty
|  |  |  +--rw cnee            -> ../../../cnee/name
|  |  +--rw rest-endpoint
|  |  |  +--rw address    inet:ip-address
|  |  |  +--rw port       inet:port-number
|  |  +--rw engine* [name]
|  |     +--rw name                       string
|  |     +--rw replicas?                  uint32
|  |     +--rw (engine-repository)
|  |        +--:(repository)
|  |        |  +--rw repository?          inet:uri
|  |        +--:(default-repository)
|  |           +--rw default-repository?   empty
|  +--rw nssf* [name]
|  |  +--rw name                          string
|  |  +--rw amf                           -> ../../amf/name
|  |  +--rw deployment
|  |  |  +--rw repository      inet:uri
|  |  |  +--rw k8s-cluster     -> /mobility/k8s-cluster/name
|  |  |  +--rw namespace?      string
|  |  |  +--rw netconf-port    inet:port-number
|  |  |  +--ro deployed?       empty
|  |  |  +--ro onboarded?      empty
|  |  |  +--ro active?         empty
|  |  |  +--rw cnee            -> ../../../cnee/name
|  |  +--rw rest-endpoint
|  |  |  +--rw address    inet:ip-address
|  |  |  +--rw port       inet:port-number
|  |  +--rw engine* [name]
|  |     +--rw name                       string
|  |     +--rw replicas?                  uint32
|  |     +--rw (engine-repository)
|  |        +--:(repository)
|  |        |  +--rw repository?          inet:uri
|  |        +--:(default-repository)
|  |           +--rw default-repository?   empty
|  +--rw pcf* [name]
|  |  +--rw name                          string
|  |  +--rw rest-endpoint
|  |  |  +--rw address    inet:ip-address
|  |  |  +--rw port       inet:port-number
|  |  +--rw engine* [name]
|  |  |  +--rw name                       string
|  |  |  +--rw replicas?                  uint32
|  |  |  +--rw (engine-repository)
|  |  |     +--:(repository)
|  |  |     |  +--rw repository?          inet:uri
|  |  |     +--:(default-repository)
|  |  |        +--rw default-repository?   empty
|  |  +--rw amf                           -> ../../amf/name
|  |  +--rw tai-group* [name]
|  |  |  +--rw name       -> deref(../../amf)/../tai-group/name
|  |  |  +--rw mcc-mnc* [mcc mnc]
|  |  |     +--rw mcc    -> deref(../../name)/../mcc-mnc/mcc
|  |  |     +--rw mnc    -> deref(../mcc)/../mnc
|  |  +--rw sst-sdt* [name]
|  |  |  +--rw name    -> deref(../../amf)/../sst-sdt/name
|  |  +--rw deployment
|  |     +--rw repository      inet:uri
|  |     +--rw k8s-cluster     -> /mobility/k8s-cluster/name
|  |     +--rw namespace?      string
```

```
|  |     +--rw netconf-port    inet:port-number
|  |     +--ro deployed?       empty
|  |     +--ro onboarded?      empty
|  |     +--ro active?         empty
|  |     +--rw cnee            -> ../../../cnee/name
|  +--rw smf* [name]
|  |  +--rw name                          string
|  |  +--rw deployment
|  |  |  +--rw repository      inet:uri
|  |  |  +--rw k8s-cluster     -> /mobility/k8s-cluster/name
|  |  |  +--rw namespace?      string
|  |  |  +--rw netconf-port    inet:port-number
|  |  |  +--ro deployed?       empty
|  |  |  +--ro onboarded?      empty
|  |  |  +--ro active?         empty
|  |  |  +--rw cnee            -> ../../../cnee/name
|  |  +--rw rest-endpoint
|  |  |  +--rw address    inet:ip-address
|  |  |  +--rw port       inet:port-number
|  |  +--rw pcf                           -> ../../pcf/name
|  |  +--rw sst-sdt* [name]
|  |  |  +--rw name    -> deref(../../pcf)/../sst-sdt/name
|  |  +--rw udm
|  |  |  +--rw address    inet:ip-address
|  |  |  +--rw port       inet:port-number
|  |  +--rw dnn* [name]
|  |  |  +--rw name         string
|  |  |  +--rw ipv4-pool* [name]
|  |  |  |  +--rw name        string
|  |  |  |  +--rw prefix      inet:ipv4-prefix
|  |  |  |  +--rw ip-range
|  |  |  |  |  +--rw start    inet:ipv4-address
|  |  |  |  |  +--rw end      inet:ipv4-address
|  |  |  |  +--rw vrf         string
|  |  |  +--rw ipv6-pool* [name]
|  |  |     +--rw name               string
|  |  |     +--rw prefix             inet:ipv6-prefix
|  |  |     +--rw vrf                string
|  |  |     +--rw prefix-lifetime?   uint32
|  |  +--rw smf* [name]
|  |  |  +--rw name       string
|  |  |  +--rw service* [name]
|  |  |     +--rw name           string
|  |  |     +--rw bind-address   inet:ip-address
|  |  +--rw profile
|  |  |  +--rw protocol
|  |  |     +--rw external-address?   inet:ip-address
|  |  |     +--rw node-label          string
|  |  +--rw tracing-endpoint
|  |     +--rw host?   string
|  |     +--rw port?   inet:port-number
|  +--rw upf* [name]
|  +--rw name                          string
|  +--rw cnee
|  |  +--rw repository     inet:uri
|  |  +--rw k8s-cluster    -> /mobility/k8s-cluster/name
|  +--rw amf
|  |  +--rw repository      inet:uri
|  |  +--rw k8s-cluster     -> /mobility/k8s-cluster/name
|  |  +--rw rest-address    -> deref(../k8s-cluster)/../worker/address
|  +--rw nrf
|  |  +--rw repository      inet:uri
|  |  +--rw k8s-cluster     -> /mobility/k8s-cluster/name
|  |  +--rw rest-address    -> deref(../k8s-cluster)/../worker/address
|  +--rw nssf
|  |  +--rw repository      inet:uri
|  |  +--rw k8s-cluster     -> /mobility/k8s-cluster/name
|  |  +--rw rest-address    -> deref(../k8s-cluster)/../worker/address
|  +--rw tai-group* [name]
|  |  +--rw name       string
|  |  +--rw mcc-mnc* [mcc mnc]
|  |     +--rw mcc    uint32
|  |     +--rw mnc    uint32
|  |     +--rw tac* [code]
|  |        +--rw code    uint32
|  |        +--rw name?   string
|  +--rw sst-sdt* [name]
|  |  +--rw name    string
|  |  +--rw sst     octet-string8
|  |  +--rw sdt     octet-string24
|  +--rw pcf-slice* [name]
```

```
|     +--rw name             string
|     +--rw repository       inet:uri
|     +--rw k8s-cluster      -> /mobility/k8s-cluster/name
|     +--rw rest-address     -> deref(../k8s-cluster)/../worker/address
|     +--rw tai-group* [name]
|     |  +--rw name        -> ../../../tai-group/name
|     |  +--rw mcc-mnc* [mcc mnc]
|     |     +--rw mcc     -> deref(../../name)/../mcc-mnc/mcc
|     |     +--rw mnc     -> deref(../mcc)/../mnc
|     +--rw sst-sdt* [name]
|     |  +--rw name     -> ../../../sst-sdt/name
|     +--rw smf-slice* [name]
|        +--rw name             string
|        +--rw repository       inet:uri
|        +--rw k8s-cluster      -> /mobility/k8s-cluster/name
|        +--rw rest-address     -> deref(../k8s-cluster)/../worker/address
|        +--rw sst-sdt* [name]
|        |  +--rw name     -> ../../../sst-sdt/name
|        +--rw dnn* [name]
|        |  +--rw name        string
|        |  +--rw network     inet:ip-prefix
|        |  +--rw vrf         string
|        +--rw upf-slice* [name]
|           +--rw name                    string
|           +--rw dnn* [name]
|           |  +--rw name     -> ../../../dnn/name
|           +--rw management-address     inet:ip-address
|           +--rw hostname?               string
|           +--rw n3
|           |  +--rw address     inet:ip-address
|           |  +--rw netmask     inet:ip-address
|           |  +--rw gateway     inet:ip-address
|           +--rw n4
|           |  +--rw address     inet:ip-address
|           |  +--rw netmask     inet:ip-address
|           |  +--rw gateway     inet:ip-address
|           |  +--rw port?       inet:port-number
|           +--rw as-number              uint32
|           +--rw pe-bgp-as-number       uint32
|           +--rw n6
|           |  +--rw address     inet:ip-address
|           |  +--rw netmask     inet:ip-address
|           |  +--rw gateway     inet:ip-address
|           +--rw gi-server-loopback?   inet:ip-address
+--rw k8s-cluster* [name]
   +--rw name                           string
   +--rw k8s-master                     inet:uri
   +--rw base-uri                       inet:uri
   +--rw api-credentials!
   |  +--rw username    string
   |  +--rw password    string
   +--rw application-yaml-directory     inet:uri
   +--rw netconf-port-pool?             string
   +--rw worker* [address]
   |  +--rw address     inet:ip-address
   |  +--rw hostname?   string
   |  +--rw label?      string
   +--rw netconf-address                inet:ip-address
   +--ro cluster-available?             empty
```