Implementação de um Gerente de Rede Utilizando SNMPv2C e Python3

Gabriel C. Chiele[1], **Maiki Buffet**[1]

[1]Escola Politécnica – Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS)
Caixa Postal 15.064 – 91.619-900 – Porto Alegre – RS – Brazil

gabriel.chiele@acad.pucrs.br, maiki.buffet@acad.pucrs.br

Abstract. *This article describes the implementation and use process of a web manager based on SNMPv2 protocol. For the development of this assignment was used the language Python, version 3, and it's library PYSNMP.*

Resumo. *Este artigo descreve o processo de implementação e utilização de um gerente de redes baseado no protocolo SNMPv2. Para o desenvolvimento deste trabalho foi utilizada a linguagem de programação Python, versão 3, e a biblioteca PYSNMP.*

1. Introdução

O objetivo deste trabalho é desenvolver um gerente de redes, baseado no protocolo *SNMPv2*, que seja capaz de realizar os comandos clássicos do *SNMP* (*get*, *getnext*, *set*, *getbulk* e *walk*), além de comandos customizados (*gettable* e *getdelta*). A implementação do gerente foi realizada em *Python*, versão 3, com auxílio das bibliotecas *PYSNMP* e *OPTPARSER*. A linguagem foi definida para o desenvolvimento deste trabalho devido a sua praticidade e simplicidade.

2. Gerenciador de Redes

O administrador de rede em informática é o responsável por projetar e manter uma rede de computadores em funcionamento, de acordo com o desejado pelo próprio ou por quem o designou para a função. Tem como atribuição principal o gerenciamento da rede local, bem como dos recursos computacionais relacionados direta ou indiretamente.

Para cumprir suas funções, o administrador de rede geralmente utiliza um programa gerente da rede, e este é responsável pelo monitoramento e controle dos sistemas de *hardware* e *software* que a compõem. Suas operações consistem em detectar e corrigir problemas que causem ineficiência na comunicação e eliminar as condições que poderão fazer o problema surgir novamente.

3. *Simple Network Management Protocol* (*SNMP*)

O *SNMP* é um protocolo padrão da *Internet* para gerenciamento de dispositivos em redes *IP*. Os dispositivos que normalmente suportam *SNMP* são roteadores, computadores, servidores, estações de trabalho, impressoras, racks modernos e etc. Este protocolo é usado na maioria das vezes em sistemas de gerenciamento de rede para monitorar dispositivos conectados a ela.

O *SNMP* é um conjunto de padrões de gerenciamento de rede, que inclui um protocolo da camada de aplicação, um esquema de banco de dados e um conjunto de objetos de dados. [CASE 1990]

Uma rede gerida pelo protocolo *SNMP* é formada por três componentes chaves:

- **Dispositivos Geridos:**
  Dispositivo gerido é um nó de rede que possui um agente *SNMP* instalado e se encontra numa rede gerida. Estes dispositivos coletam e armazenam informações de gestão e mantém estas informações disponíveis para sistemas *NMS* através do protocolo *SNMP*. Dispositivos geridos, também às vezes denominados de dispositivos de rede, podem ser roteadores, servidores de acesso, impressoras, computadores, servidores de rede, *switches*, dispositivos de armazenamento, dentre outros.
- **Agentes:**
  Um agente é um módulo de *software* de gestão de rede que fica armazenado num dispositivo gerido. Um agente tem o conhecimento das informações de gestão locais e traduz estas informações para um formato compatível com o protocolo *SNMP*.
- **Sistema de Gerenciamento de Redes:**
  Um sistema *NMS* (*Network-Management Systems*) é responsável pelas aplicações que monitoram e controlam os dispositivos geridos. Normalmente é instalado em um (ou mais que um) servidor de rede, dedicado a estas operações de gestão, que recebe informações (pacotes *SNMP*) de todos os dispositivos geridos daquela rede.

3.1. Comandos *SNMP*

A primeira versão do protocolo *SNMP* não define um grande número de comandos, apenas alguns pacotes de unidades de dados (*PDU*), que são estes:

- ***GET*:** usado para retirar um pedaço de informação de gerenciamento.
- ***GETNEXT*:** usado interativamente para retirar sequências de informação de gerenciamento.
- ***GETBULK*:** usado para retirar informações de um grupo de objetos.
- ***SET*:** usado para fazer uma mudança no subsistema gerido.
- ***TRAP*:** usado para reportar uma notificação ou para outros eventos assíncronos sobre o subsistema gerido.

A versão 2 do *SNMP* é uma evolução do protocolo inicial. O *SNMPv2* oferece uma boa quantidade de melhoramentos em relação ao *SNMPv1*, incluindo operações adicionais do protocolo, melhoria na performance, segurança, confidencialidade e comunicações gerente-para-gerente.

4. Implementação

A implementação do gerente de rede foi realizada utilizando a linguagem de programação *Python*, com auxílio das bibliotecas *PYSNMP* e *OPTPARSER*. A biblioteca *PYSNMP* fornece métodos para criação dos pacotes *SNMP*, necessários para o desenvolvimento do gerente de redes. Já a *OPTPARSER* é utilizada para criar a interface de linha de comando, a partir de parâmetros de entrada. Neste capítulo vamos apresentar os detalhes do desenvolvimento da ferramenta de gerência, que é o objeto de estudo deste documento.

4.1. Interface

A implementação da interface foi realizada utilizando a biblioteca *OPTPARSER*, pois ela proporciona um meio rápido de adicionar e analisar argumentos de entrada. Primeiro, se cria uma instância de um objeto da classe *OptionParser*, esta então utiliza o método *add_option* que irá receber os parâmetros *flag*, e um destino para o valor ser armazenado. Após todos os possíveis argumentos serem adicionados ao objeto, utiliza-se a função *parse_options* que resulta em um dicionário, onde o nome do destino dado ao adicionar o parâmetro, é a chave para busca do valor passado na linha de comando.

```
parser = OptionParser()

parser.add_option("-p", "--public"   , dest="public"   , help='publico'    , action='store_true', default='false')
parser.add_option("-c", "--community", dest="community", help='comunidade')
parser.add_option("-t", "--target"   , dest="target"   , help='alvo'       )
```

Figura 1. Exemplo de uso da biblioteca *optparser*

4.2. Comandos

A implementação dos comandos foi realizada utilizando a biblioteca *pysnmp*. Ela fornece os métodos clássicos do protocolo *SNMP*, descritos no capítulo 3.

O primeiro passo para utilizar a biblioteca é criar uma *SnmpEngine*, que é o objeto responsável por criar a comunicação entre os agentes e o gerente. O próximo passo, é definir a comunidade e o ip alvo, que é realizado a partir da criação dos objetos *CommunityData* e *UdpTransportTarget*. Depois, é criado o objeto *ObjectIdentity* com o *OID* que deverá ser acessado. Estes objetos são parâmetros para o comando que se deseja utilizar. [ETINGOF 2005]

No exemplo da Figura2, foi utilizado o comando *GET*. Dessa forma o programa se encontra pronto para enviar a mensagem *SNMP*, via o método *next*, que armazena em *varBinds* o retorno do comando enviado.

```
engine = SnmpEngine()
comm   = CommunityData('public')
target = UdpTransportTarget(('localhost', 161))
id     = ObjectIdentity('1.3.6.1.2.1.1.9')

command = getCmd(engine, comm, target, ContextData(), ObjectType(id))
errorIndication, errorStatus, errorIndex, varBinds = next(command)
```

Figura 2. Exemplo de uso da biblioteca *pysnmp*

Este é o processo padrão utilizado para desenvolver os comandos suportados pelo protocolo, com exceção do método *GETBULK*. Os métodos customizados foram implementados utilizando uma combinação das funções clássicas.

4.2.1. GETBULK

Para implementar o *GETBULK*, utilizamos o parâmetro *NR* para controlar o número de chamadas do método *GET*, e após isso, o parâmetro *NC* para controlar o número de chamadas do método *NEXT* para cada *OID* subsequente.

```
nr = int(options.NR)
for oid in lstOIDs:
    if (nr > 0):
        command = get_.Get(options, oid)
        nr = nr - 1
    else:
        oid2 = oid
        for j in range(int(options.NC)):
            command = nextCmd(SnmpEngine(), CommunityData(comm),\
                 UdpTransportTarget((options.target, 161)),\
                 ContextData(), ObjectType(ObjectIdentity(oid2)))

            errorIndication, errorStatus, errorIndex, varBinds = next(command)

            if errorIndication:
                print(errorIndication)
            elif errorStatus:
                print('%s at %s' % (errorStatus.prettyPrint(),
                                    errorIndex and varBinds[int(errorIndex) - 1][0] or '?'))
            else:
                for varBind in varBinds:
                    id, value = varBind
                    oid2 = id
                    print(' = '.join([x.prettyPrint() for x in varBind]))
```

Figura 3. Implementação do *GETBULK*

4.2.2. GETTABLE

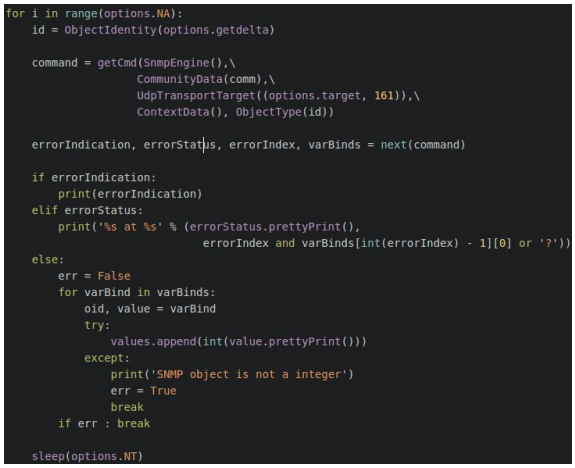
Para implementar o *GETTABLE*, utilizamos uma sequência de chamadas do método *NEXT*, até o próximo *OID* não pertencer a mesma Tabela. Os valores retornados são armazenados em uma lista e, ao final da execução, esses valores são formatados para uma tabela.

```
for (errorIndication, errorStatus, errorIndex, varBinds) in nextCmd(SnmpEngine(),
                           CommunityData(comm),
                           UdpTransportTarget((options.target, 161)),
                           ContextData(),
                           ObjectType(ObjectIdentity(options.gettable)),
                           lexicographicMode=False):

    if errorIndication:
        print(errorIndication)
        break
    elif errorStatus:
        print('%s at %s' % (errorStatus.prettyPrint(),
                           errorIndex and varBinds[int(errorIndex)-1][0] or '?'))
        break
    else:
        for varBind in varBinds:
            oid, value = varBind
            rest, cl = oid.prettyPrint().split('::', 1)
            c, l = cl.split('.', 1)

            if not (c in collumns):
                collumns.append(c)

            if not (l in lines):
                lines.append(l)

            values.append(value.prettyPrint())
```

Figura 4. Implementação do *GETTABLE*

4.2.3. GETDELTA

Para implementar o *GETDELTA*, utilizamos o parâmetro *NA* para controlar o número de chamadas do método *GET*, e após isso, o parâmetro *NT* para controlar o tempo entre as requisições. Os valores retornados são armazenados em uma lista e, ao final da execução, os deltas são calculados e apresentados no terminal.



```
for i in range(options.NA):
    id = ObjectIdentity(options.getdelta)

    command = getCmd(SnmpEngine(),\
                     CommunityData(comm),\
                     UdpTransportTarget((options.target, 161)),\
                     ContextData(), ObjectType(id))

    errorIndication, errorStatus, errorIndex, varBinds = next(command)

    if errorIndication:
        print(errorIndication)
    elif errorStatus:
        print('%s at %s' % (errorStatus.prettyPrint(),
                            errorIndex and varBinds[int(errorIndex) - 1][0] or '?'))
    else:
        err = False
        for varBind in varBinds:
            oid, value = varBind
            try:
                values.append(int(value.prettyPrint()))
            except:
                print('SNMP object is not a integer')
                err = True
                break
        if err : break

    sleep(options.NT)
```

Figura 5. Implementação do *GETDELTA*

5. Problemas Encontrados

As bibliotecas utilizadas durante o desenvolvimento do trabalho aumentaram a velocidade de implementação, diminuindo o tempo necessário para criar um código que não fosse parte do objetivo do trabalho. Entretanto, foram encontrados problemas na tentativa de fazê-las operar como desejado. Mesmo com estes imprevistos sendo contornados, acabaram por ocasionar atrasos na implementação.

Uma das dificuldades encontradas foi que a biblioteca utilizada para realizar a análise de argumentos (*optparser*), não permite a leitura de mais de um valor por *flag*. Isto foi contornado utilizando um padrão que permite múltiplos valores de entrada.

O maior problema encontrado foi relacionado com o comando *GETBULK*. Nele, não foi possível utilizar o método existente na biblioteca *pysnmp*, pois ela não aceitava múltiplos *OID* em uma única chamada da função. Isto fez com que tivéssemos que implementar manualmente o funcionamento deste comando.

6. Conclusão

Após a realização deste trabalho, pudemos concluir que a tarefa de gerenciar redes, sem o uso de ferramentas de gerência, é uma tarefa complicada. Além disso, podemos nos familiarizar melhor com o protocolo *SNMP* e aprender a criar funções mais complexas, juntando os métodos clássicos provido pelo mesmo.

Referências

CASE, J. D. e. a. (1990). Simple network management protocol (snmp).

ETINGOF, I. (2005). Pysnmp - api reference.