Česká zemědělská univerzita v Praze

Provozně ekonomická fakulta

Katedra informačního inženýrství

Diplomová práce

Monitorování telekomunikačního systému pomocí Nagiosu

Bc. Ivan Štětka

ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE

Provozně ekonomická fakulta

ZADÁNÍ DIPLOMOVÉ PRÁCE

Ivan Štětka

Informatika

Název práce

Monitorování telekomunikačního systému pomocí Nagiosu

Název anglicky

Monitoring of telecommunication system using Nagios

Cíle práce

Hlavním cílem diplomové práce je zajištění monitorování telekomunikačního systému pomocí open source systému Nagios pro společnost zabývající se poskytováním IT a telekomunikačních služeb. Mezi dílčí cíle patří pilotní implementace monitorování nahrávacího systému ReDat do Nagiosu pomocí SNMP a tvorba vizualizace výstupních dat z Nagiosu pomocí NagVisu.

Metodika

Metodika řešené problematiky diplomové práce je založena na studiu a analýze odborných informačních zdrojů, analýze telekomunikačního systému, analýze požadavků na monitorování telekomunikačního systému. Syntézou studia těchto analýz je konfigurace a realizace monitorování telekomunikačního systému pomocí Nagiosu.

Doporučený rozsah práce

60 – 80 stran

Klíčová slova

Nagios, SNMP, MIB, NRPE, NagVis, telekomunikační systém, ReDat, monitorování počítačové sítě

Doporučené zdroje informací

FOROUZAN, B. A. TCP/IP protocol suite. 4th ed. Boston: McGraw-HillHigherEducation, 2010. ISBN 978-0-07-337604-2.

KOCJAN, W. Learning Nagios 3.0. Birmingham: Packt Publishing, 2008. ISBN: 978-1-84719-518-0.

KRETCHMAR, J.M. – DOSTÁLEK, L. Administrace a diagnostika sítí: pomocí OpenSource utilit a nástrojů. Brno: Computer Press, 2004. ISBN 80-251-0345-5.

MATOUŠEK, P. Síťové aplikace a jejich architektura. Brno: Nakladatelství Vysokého učení technického v Brně VUTIUM, 2014. ISBN 978-80-214-3766-1.

MAURO, D. – SCHMIDT, K. Essentials SNMP, Second Edition. Sebastopol: O'Reilly Media, 2005. ISBN: 0-596-00840-6.

Předběžný termín obhajoby

2017/18 LS – PEF

Vedoucí práce

Ing. Jiří Brožek, Ph.D.

Garantující pracoviště

Katedra informačního inženýrství

Elektronicky schváleno dne 13. 3. 2016

Ing. Martin Pelikán, Ph.D.

Vedoucí katedry

Elektronicky schváleno dne 13. 3. 2016

Ing. Martin Pelikán, Ph.D.

Děkan

V Praze dne 26. 03. 2018

Čestné prohlášení

Prohlašuji, že svou diplomovou práci „Monitorování telekomunikačního systému pomocí Nagiosu“ jsem vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou citovány v práci a uvedeny v seznamu literatury na konci práce. Jako autor uvedené diplomové práce dále prohlašuji, že jsem v souvislosti s jejím vytvořením neporušil autorská práva třetích osob.

V Praze dne 26.3.2018 _____

Ivan Štětka

Poděkování

Rád bych touto cestou poděkoval Ing. Jiřímu Brožkovi, Ph.D. za odborné vedení a podnětné připomínky při tvorbě této diplomové práce. Dále bych poděkoval svým kolegům a obchodním partnerům, se kterými spolupracuji v rámci zaměstnání, za poskytnutí rad a podkladových materiálů pro tuto práci.

Monitorování telekomunikačního systému pomocí Nagiosu

Abstrakt

Diplomová práce řeší problematiku monitorování telekomunikačního systému pomocí open source systému Nagios pro společnost, která se mimo jiné zabývá poskytováním IT a telekomunikačních služeb. V rámci této práce je provedena pilotní implementace monitorování záznamového systému ReDat do Nagiosu pomocí protokolu SNMP a tvorba vizualizace výstupních dat z Nagiosu pomocí NagVisu.

Nejprve jsou popsána teoretická východiska v několika kapitolách. První kapitola nás uvádí do světa počítačových sítí, druhá popisuje monitorování síťových zařízení obecně, ve třetí jsou pak podrobně popsány vybrané protokoly pro monitorování. Čtvrtá kapitola popisuje systém Nagios a Nagvis, pátá Telekomunikační systém a šestá záznamový systém ReDat.

Ve vlastní práci jsou specifikovány požadavky na monitorování telekomunikačního systému společnosti, jsou analyzovány možnosti monitorování Nagiosu, je provedena konfigurace Nagiosu a NagVisu a jsou navrženy vlastní pluginy (skripty) pro monitorování.

Klíčová slova: Nagios, SNMP, MIB, NRPE, NagVis, telekomunikační systém, ReDat, monitorování počítačové sítě.

Monitoring of telecommunication system using Nagios

Abstract

This diploma thesis solves the problems of monitoring of the telecommunication system using Nagios open source system for the company dealing among other things, with the provision of IT and telecommunication services. In this work, a pilot implementation of the monitoring of the ReDat recording system in Nagios via SNMP and the creation of visualization of output data from Nagios using NagVis was performed.

Firstly, the theoretical points are described in several chapters. The first chapter introduces us to the world of computer networks, the second describes the monitoring of network devices in general, the third one describes in detail the selected monitoring protocols. The fourth chapter describes the Nagios and Nagvis system, the fifth Telecommunication system and the sixth ReDat recording system.

In the main part of this diploma thesis are specified the requirements for monitoring the company's telecommunication system and further are analyzed the possibilities of Nagios monitoring, Nagios and NagVis are configured, and their own monitoring scripts are designed.

Keywords: Nagios, SNMP, MIB, NRPE, NagVis, the telecommunication system, ReDat, network monitoring.

Obsah

Seznam obrázků

Seznam tabulek

Seznam použitých zkratek

AACC – Avaya Aura Contact Center
ACD – Automatic Call Distribution
AES – Advanced Encryption Standard

AgentX – Agent Extensibility Protocol
API – Application Programming Interface
ASN.1 – Abstract Syntax Notation One
ATM – Asynchronous Transfer Mode
BSD – Berkeley Software Distribution
CAN – Campus Area Network
CGI – Common Gateway Interface
CPAN – Comprehensive Perl Archive Network
CPU – Central Processing Unit
CTI – Computer Telephony Integration
ČSN ISO – Česká technická norma dle ISO
DB – Database
DIY – Do It Yourself
DMZ – Demilitarized Zone
DN – Domain Name
DNS – Domain Name System
DP – Diplomová práce
DVD – Digital Versatile Disc
DVI-D – Digital Visual Interface – Digital
EGP – Exterior Gateway Protocol
FTP – File Transfer Protocol
GIF – Graphics Interchange Format
GNU GPL – GNU General Public License
HDD – Hard Disk Drive
HID – Human Interface Devices
HP – Hewlett-Packard
HTTP – Hypertext Transfer Protocol
HTTPS – Hypertext Transfer Protocol Secure
HW – Hardware
iCC – Intelligent Contact Centre
ICMP – Internet Control Message Protocol
ICT – Information and Communications Technology
IDE – Integrated Development Environment
IGMP – Internet Group Management Protocol
IETF – Internet Engineering Task Force
IIS – Internet Information Server
IM – Instant Messaging
IP – Internet Protocol
IPX/SPX – Internetwork Packet Exchange/Sequenced Packet Exchange
ISDN – Integrated Services Digital Network
ISO – International Standards Organization
IT – Information Technology

ITU – International Telecommunication Union
IVR – Interactive Voice Response
JPEG – Joint Photographic Experts Group
KC – Kontaktní centrum
KP – Konfigurační položka
LAN – Local Area network
LDAP – Lightweight Directory Access Protocol
LCD – Liquid Crystal Display
LLC – Logical Link Control,
MAC – Media Access Control
MAN – Metropolitan Area Network
MD5 – Message-Digest algorithm
MIB – Management Information Base
MNTOS – Multi-Nagios Tactical Overview Systém
MS – Microsoft
MS-DOS – Microsoft Disk Operating Systém
MS SCOM – Microsoft System Center Operations Manager
MSP – Managed Service Providers
MySQL – My Structured Query Language
Nagios – Nagios Ain't Gonna Insist On Sainthood - rekurzivní akronym:
NEB – Nagios Event Broker
NIC – Network Interface Controller
NOC – Network Operations Center
NRPE – Nagios Remote Plugin Executor
NTP – Network Time Protocol
OCHP – Obsessive Compulsive Host Processor
OCSP – Obsessive Compulsive Service Processor
OID – Object IDentifier
OSI – Open Systems Interconnection
OS – Operating Systém
OVSD – OpenView Service Desk
PAN – Personal Area Network
PBX – Private Branch Exchange
PBX systém – Telekomunikační systém Zadavatele
PC – Personal Computer
PDU – Protocol Data Unit
PERL – Practical Extraction and Reporting language
PNG – Portable Network Graphics
PNP – PNP is Not PerfParse - rekurzivní akronym
PSTN – Public Switched Telephone Network
R3 – ReDat 3
RAID – Redundant Array of Independent Disks

RAM – Random Access Memory
RIP – Routing Information Protocol
PRI – Primary Rate Interface
REX – ReDat eXperience
RFC – Request for Commnets
RS-232 – Recommended Standard 232
RTP – Real-time Transport Protocol
RTPC – RTP Control Protocol
RVR – ReDat VoIP Recorder
RVRG – ReDat Voice Recording Gateway
SFTP – Secure File Transfer Protocol
SGMP – Simple Gateway Monitoring Protocol
SHA – Secure Hash Algorithm
SIP – Session Initiation Protocol
SQL – Structured Query Language
SMI – Structure of Management Information
SMS – Short Message Service
SMTP – Simple Mail Transfer Protocol
SNMP – Simple Network Management Protocol
SNA – Systems Network Architecture
SPA – Správa Provozních Aplikací
SSH – Secure Shell
SSI – Správa Síťové Infrastruktury
STDOUT – Standard Output
STS – Správa Telekomunikačních Systémů
SSL – Secure Sockets Layer
SUS – Správa Unix/Linux Systémů
SW – Software
SWA – Správa Windows Aplikací
TAPI – Telephony Application Program Interface
TCP – Transmission Control Protocol
TDM – Time Division Multiplex
TTL – Time To Live
USB – Universal Serial Bus
UDP – User Datagram Protocol
UOM – Unit Of Measurement
VAC – Volt Alternating Current
VDC – Volt Direct Current
VLAN – Virtual LAN
VoIP – Voice over IP
WAN – Wide Area Network
XML – eXtensible Markup Language

1 Úvod

Informační technologie se v průběhu času vyvíjejí čím dál tím rychleji a přinášejí nové možnosti ve využití. Mezi tyto technologie patří i telekomunikační systémy. Ty prošly vývojem od manuálního přepojování hovorů, přes automatické analogové a digitální systémy až k IP telefonii. Díky tomuto rychlému vývoji a následné cenové dostupnosti jsou dnes soukromé společnosti a státní instituce schopny provozovat své vlastní telekomunikační systémy. Mezi takové společnosti patří i střední až velká společnost (dále jen Zadavatel, pozn.: společnost si přeje zůstat v anonymitě), která poskytuje mimo jiné i nepřetržité telekomunikační služby pro více než 3000 uživatelů. Součástí telekomunikačních systémů bývají různé aplikační nadstavby a doplňkové technologie, jako například tarifikační aplikace, kontaktní centrum, dispečerský systém, konferenční systém, databáze telefonního seznamu, hlasová pošta, fax server, přenosový systém, proudový zdroj. Specifickou částí těchto technologií jsou hovorové záznamové systémy, které jsou důležité pro vyhodnocování vzniklých krizových situací, mimořádných událostí, nestandardních provozních situací, požárních zásahů, bezpečnostních incidentů a v neposlední řadě řešení reklamací a stížností zákazníků. Záznamové systémy jsou využívány například v železniční a letecké dopravě, u složek záchranných sborů, v energetice, bankovnictví a kontaktních centrech. Mezi nejvýznamnější záznamové systémy v České republice patří systém ReDat od pardubické společnosti Retia.

Výše zmíněné technologie jsou v dnešní době již zcela závislé na počítačových sítích, a tím je kladen velký důraz na dostupnost síťových zařízení a služeb. Každé prodlení nebo výpadek služby mnohdy představuje velkou finanční ztrátu, poškození jména nebo samotného chodu společnosti, a proto je žádoucí monitorovat stav technologií připojených do sítě a tím předcházet problémům nebo minimálně včas reagovat na vzniklé problémy.

Téma diplomové práce (dále jen DP) bylo vybráno na základě potřeby Zadavatele zajistit monitorování telekomunikačního systému. Tato potřeba vznikla v době, kdy došlo ke snižování nákladů snížením počtu administrátorů telekomunikačního systému ze šesti na čtyři a zrušením nočních služeb. Administrátoři dříve prováděli kontroly a dohled systému manuálně v rámci služeb H24, což již dále nebylo možné. Dalším impulzem bylo ukončení podpory a rozvoje monitorovací aplikace ReDat Explorer výrobcem Retia. V rámci snižování nákladů Zadavatele byla snaha co nejvíce využít stávající zdroje, technologie a systémy nebo doplnit tyto systémy o svobodný software a software s otevřeným kódem.

2 Cíl práce a metodika

2.1 Cíl práce

Hlavním cílem DP je zajištění monitorování telekomunikačního systému pomocí open source systému Nagios pro společnost zabývající se poskytováním IT a telekomunikačních služeb. Mezi dílčí cíle patří pilotní implementace monitorování záznamového systému ReDat do Nagiosu pomocí SNMP a tvorba vizualizace výstupních dat z Nagiosu pomocí NagVisu.

2.2 Metodika

Metodika řešené problematiky DP je založena na studiu a analýze odborných informačních zdrojů, analýze telekomunikačního systému a analýze požadavků na monitorování telekomunikačního systému. Syntézou studia těchto analýz je konfigurace a realizace monitorování telekomunikačního systému pomocí Nagiosu.

Teoretická východiska pro tuto práci jsou rozdělena do několika oblastí. V první oblasti jsou vysvětleny základní pojmy ve světě počítačových TCP/IP sítí. Druhá oblast popisuje monitorování síťových zařízení obecně, včetně základního dělení monitorování. Ve třetí oblasti jsou pak nastíněny nejpoužívanější nástroje a podrobně popsány protokoly ICMP a SNMP a jejich využití v monitorování. Čtvrtá oblast se zabývá monitorovacím systémem Nagios a vizualizačním nástrojem NagVis. V páté oblasti jsou vysvětleny pojmy a historický vývoj telekomunikačních systémů dle generací telefonních ústředen. Poslední oblast se zabývá záznamovým systémem ReDat a jeho možnostmi monitorování pomocí protokolu SNMP.

V praktické části DP jsou na základě analýzy u Zadavatele specifikovány požadavky na monitorování telekomunikačního systému. Jsou analyzovány možnosti monitorování Nagiosu. Výstupem těchto analýz je provedení konfigurace Nagiosu, vizualizace výstupních dat z Nagiosu pomocí aplikace NagVis a jsou navrženy vlastní pluginy (skripty) pro Nagios monitorování. Součástí konfigurace Nagiosu je také provedení pilotní implementace monitorování záznamového systému ReDat pomocí protokolu SNMP.

3 Teoretická východiska

3.1 Počítačové sítě

Počítačovou sítí označujeme technické prostředky, které zajišťují komunikaci a sdílení zdrojů mezi dvěma a více zařízeními (například počítači) vzájemně propojenými. Tyto sítě lze dělit podle různých parametrů. (Sosinsky, 2010, s. 27-42), (Bouška, 2010, s. 8)

Dle velikosti rozlohy a účelu:

- PAN (Personal Area Network, někdy označované jako pLAN) – osobní sítě s dosahem jednotek metrů, které například tvoří propojení osobního počítače/notebooku, mobilního telefonu s headsetem přes Bluetooth,
- LAN (Local Area Network) – nejčastějším příkladem počítačové sítě, jedná se o lokální sítě v rámci kanceláře, podlaží nebo jedné budovy,
- CAN (Campus Area Network) – v rámci více budov určitého zaměření, například v rámci podniku, univerzity,
- MAN (Metropolitan Area Network) – síť přibližně velikosti na úrovni města,
- WAN (Wide Area Network) – geograficky největší sítí, za WAN lze označit celosvětová síť Internet.

Dle síťové topologie: lineární (sběrnice), hvězdicové, kruhové, členité, hybridní.

Dle architektury sítě: Ethernet, Token ring, ARCnet.

Dle použitého komunikačního protokolu: TCP/IP (viz dále), IPX/SPX (Internetwork Packet Exchange/Sequenced Packet Exchange), SNA (Systems Network Architecture), NetBEUI, AppleTalk.

Dle přenosového media: koaxiální kabel, kroucená dvojlinka, optické vlákno, bezdrátové.

Dle síťového operačního systému: UNIX, NetWare, Windows, hybridní.

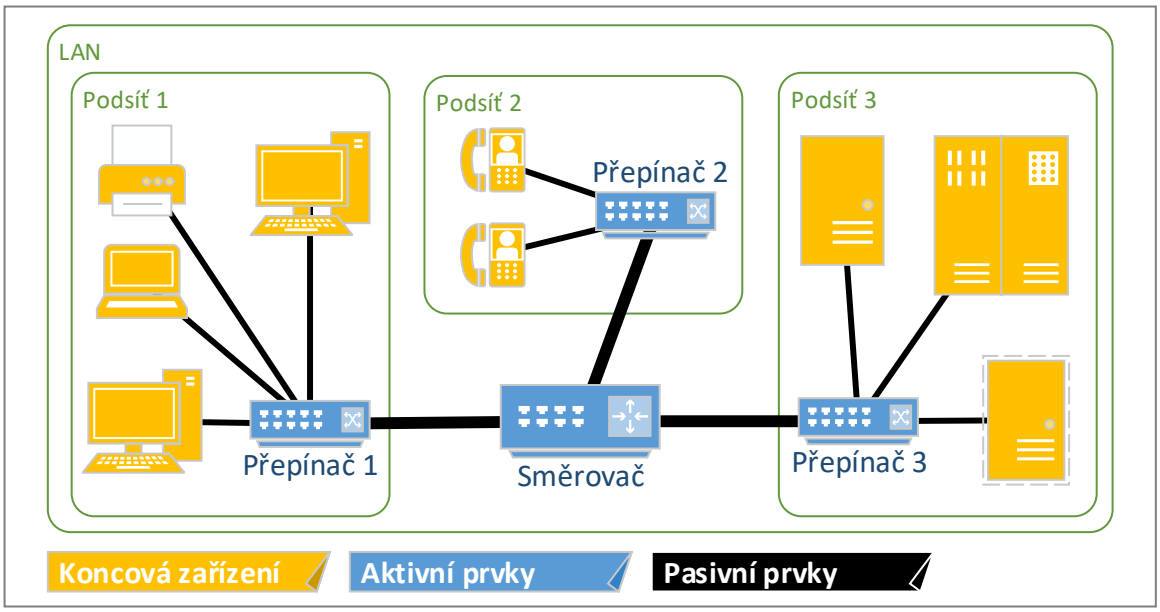
3.1.1 Síťová zařízení

Počítačová síť je složena z určitých síťových zařízení, která můžeme rozdělit do tří kategorií: koncových zařízení, pasivních prvků a aktivních prvků (Bouška, 2010, s. 8).

Koncová zařízení jsou zdrojem nebo cílovým příjemcem dat jako například počítače, servery, tiskárny, VoIP telefony, IP hodiny, čidla (teplotní, požární, pohybová apod.).

Pasivní prvky zajišťují propojení mezi koncovými zařízeními a aktivními prvky (data přes ně prochází bez aktivního zásahu). Jsou to především: kabeláž, patch panely, datové a serverové rozvaděče (racky), konektory a telekomunikační zásuvky.

Aktivní prvky přijímají, odesílají a přeposílají data ke koncovým zařízením. Jedná se o přepínače (switch), směrovače (router), firewally, rozbočovače (hub), opakovače (repeater), přístupové body bezdrátové sítě (WiFi Access point), brány (gateway), síťové adaptéry/karty NIC (Network Interface Controller). (Bouška, 2010, s. 8)



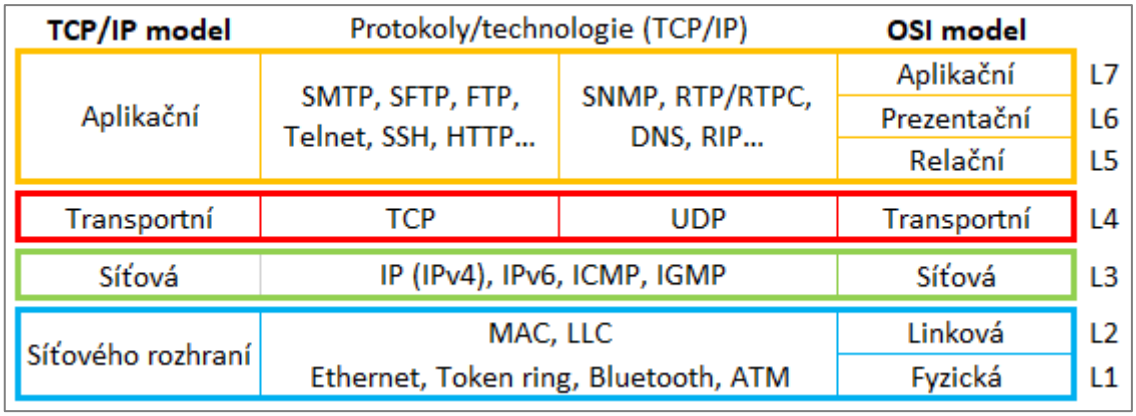
Obrázek 1 - Prvky počítačové sítě
Zdroj: Autor dle (Bouška, 2010, s. 8)

Aktivní prvky a komunikační protokol určují, jakým způsobem jsou data fyzicky posílána v síti. Dnešní sítě používají přepínaný Ethernet s využitím přepínačů, které propojují koncová zařízení v rámci jedné podsítě (subnetu). V případě posílání dat do jiné podsítě se využívá routování (určování cest přenášených dat) pomocí směrovače.

Mimo dělení sítí na fyzické podsítě lze síť rozdělovat logicky bez ohledu na fyzické zapojení. K tomuto účelu slouží tzv. virtuální sítě VLAN. Toto logické členění se realizuje už na úrovni přepínačů. (Bouška, 2010, s. 8)

3.1.2 Síťový komunikační model

Na popis síťové komunikace můžeme použít referenční abstraktní OSI model (Open Systems Interconnection) vydaný Mezinárodní standardizační organizací ISO (International Standards Organization) jako ISO 7498:1984 a revidovaný ISO 7498:1994 (ISO, ©1994). OSI model se skládá ze sedmi vrstev, kde každá vrstva zajišťuje určitou část komunikace či služeb. V tomto modelu jsou přesně definovány vrstvy, přičemž se každé vrstvě povoluje komunikace pouze se sousední vrstvou. Díky své komplikované struktuře nebyl OSI model nikdy plně implementován. V praxi je proto používanější model TCP/IP založený na protokolech TCP (Transmission Control Protocol) a IP (Internet Protocol). Model vychází z referenčního modelu Arpanet pouze jako doporučení RFC (Request for Commnets) vydávané Komisí pro technickou stránku internetu IETF (Internet Engineering Task Force), viz RFC 871 a RFC 1122 (*IETF Documents*, 2017). (Matoušek, 2010, s. 20)



Obrázek 2 - Vrstvy TCP/IP a OSI modelu
Zdroj: Autor dle (Matoušek, 2010, s. 17 a 22)

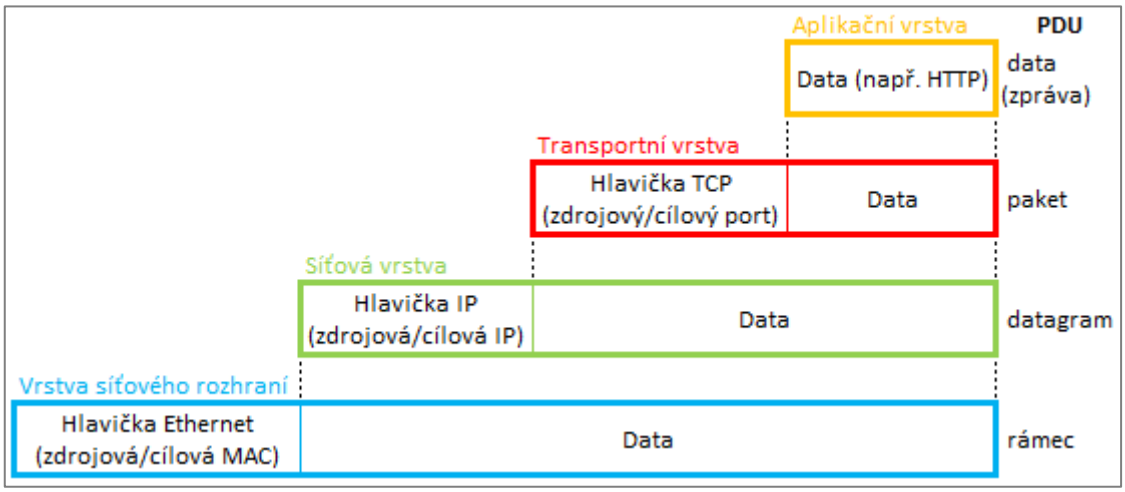
TCP/IP model používá vlastní upravený model, který se skládá pouze ze čtyř vrstev, viz obrázek 2. V každé vrstvě se rozlišují názvy základních datových jednotek PDU (Protocol Data Unit). Je potřeba vzít na vědomí, že různé zdroje uvádějí odlišné názvy. V této práci bude dále použito názvosloví PDU pro TCP/IP model dle (Matoušek, 2010, s. 18), viz obrázek 3.

Vrstva síťového rozhraní popisuje standardy (například Ethernet) a přístup k fyzickému přenosovému médiu (ovladače síťových karet). K identifikaci síťového rozhraní se používá fyzická adresa MAC (Media Access Control).

Síťová vrstva zajišťuje síťovou adresaci, směrování a předávání datagramů nejčastěji IP protokolem, jehož součástí je i protokol ICMP, který slouží ke kontrole a řízení toku dat. K identifikaci zařízení na této vrstvě slouží IP adresa (například 193.84.35.164).

Transportní vrstva zajišťuje transportní služby pro předávání dat mezi koncovými uzly pomocí protokolů TCP nebo UDP. Služba je identifikována pomocí portu – číselného označení. TCP (Transmission Control Protocol) umožňuje kontrolovaný spolehlivý přenos dat. UDP (User Datagram Protocol) umožňuje rychlý nekontrolovaný přenos dat. Transportní protokoly také zajišťují rozdělování aplikačních dat na menší pakety.

Aplikační vrstva zajišťuje zpracování dat aplikačními protokoly na nejvyšší úrovni včetně kódování a reprezentace dat. K identifikaci zařízení na aplikační úrovni se používají doménové adresy (DN – Domain Name), které jsou pro uživatele zapamatovatelnější než IP adresy (například IP: 193.84.35.164 a DN: pef.czu.cz). K překladu IP adres na doménové adresy slouží služba DNS (Domain Name System).



Obrázek 3 - Zapouzdření PDU v TCP/IP
Zdroj: Autor dle (Matoušek, 2010, s. 21)

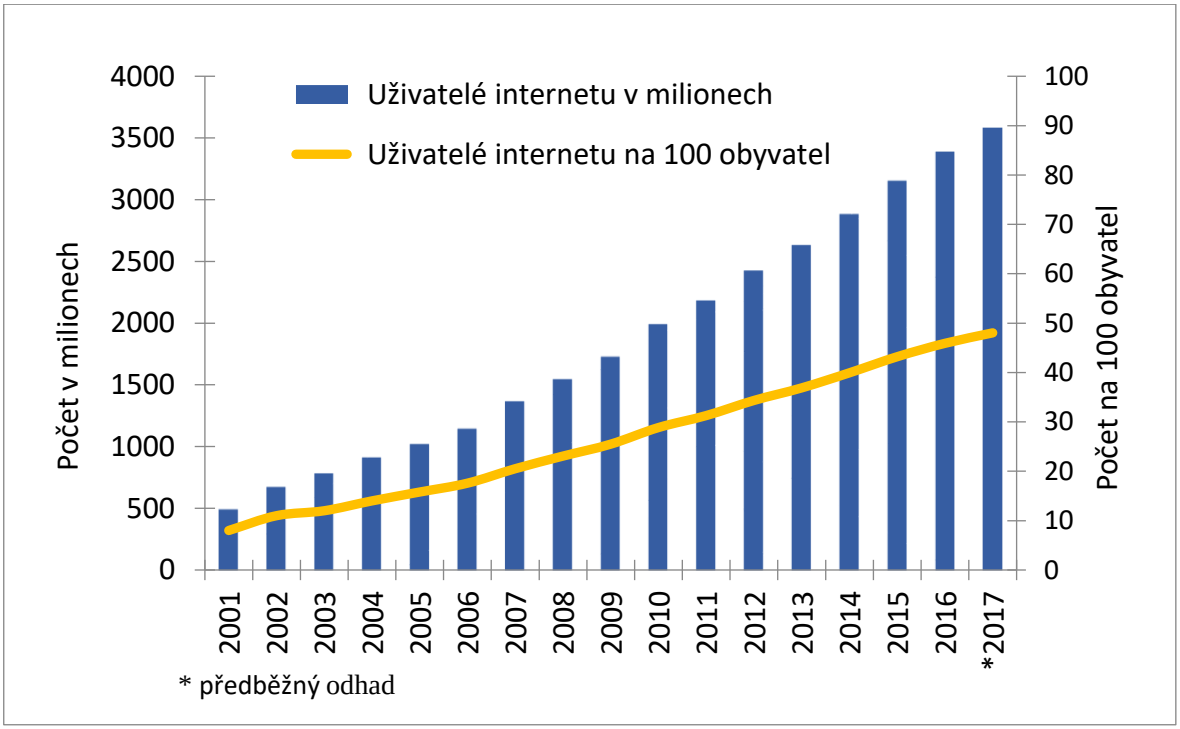
Při odesílání dat po síti dochází k zapouzdření dat horních vrstev do PDU nižších vrstev na straně odesílatele a opačně rozbalení dat na straně příjemce. Uživatelská data (například z webového klienta – požadavkem na zobrazení webových stránek příkazem

GET) jsou nejdříve na aplikační vrstvě vložena do PDU aplikačního protokolu HTTP. Přidáním hlavičky transportní vrstvy je vytvořen TCP paket, který mimo jiné obsahuje číslo portu aplikačního protokolu. V síťové vrstvě je k této PDU připojena IP hlavička se zdrojovou a cílovou IP adresou. Poté je tento IP datagram zabalen do ethernetového rámce a odeslán na přenosové medium a dále přes pasivní a aktivní síťové prvky k příjemci. Na straně příjemce (webový server) pak dochází k rozbalení dat a převzetí požadavku na zobrazení webové stránky. Webový server pak obdobným způsobem pošle požadovanou stránku zpět klientovi. (Matoušek, 2010, s. 21-22)

Podrobnější a komplexnější informace o počítačových sítích lze nalézt v knihách (Sosinsky, 2010), (Matoušek, 2010).

3.2 Monitorování síťových zařízení

V době počítačových sítí s neustále rostoucím počtem uživatelů Internetu, viz obrázek 4, kdy se málokteré odvětví obejde bez lokálních počítačových sítí nebo Internetu, je kladen velký důraz na dostupnost síťových zařízení a služeb.



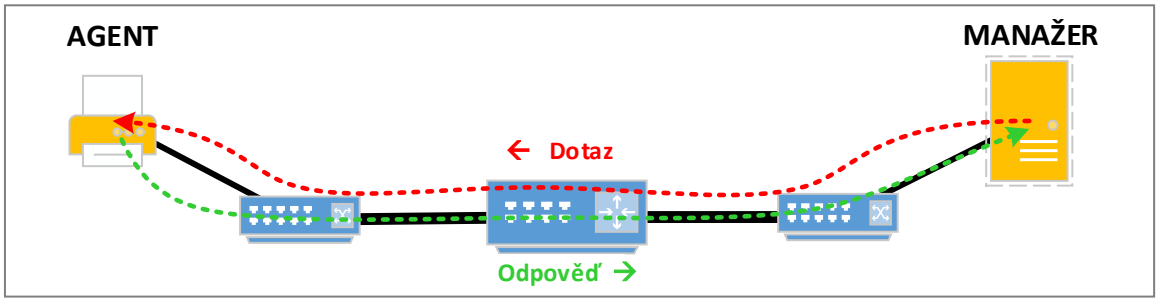
Obrázek 4 - Globální počet osob využívajících Internet celkem a na 100 obyvatel
Zdroj: Autor dle statistik (ITU, ©2018)

Hlavním mottem každého opravdového síťového administrátora nebo správce systému by především mělo být: „*odhalit problémy dříve, než je zaznamenají uživatelé*“ (Matoušek, 2014, s. 357), a to bez monitorování není možné. Monitorováním sítě rozumíme nepřetržité zjišťování stavu síťových zařízení a služeb. Monitorování tedy poskytuje informace o dostupnosti a funkci nabízených služeb, čímž také umožňuje plánovat budování síťové infrastruktury a umožňuje včas plánovat rozmístění a výkon zařízení (Matoušek, 2014, s. 358). Při monitorování se používá koncept Manažera (klient) a Agenta (server). To znamená, že Manažer kontroluje a monitoruje sadu Agentů, obvykle směrovačů, přepínačů, serverů, počítačů, tiskáren, čidel apod. (Matoušek, 2014, s. 368)

Monitorování síťových zařízení lze rozdělit do dvou základních kategorií: aktivní monitorování a pasivní monitorování. Specifickou částí monitorování je pak analýza síťových dat.

3.2.1 Aktivní monitorování

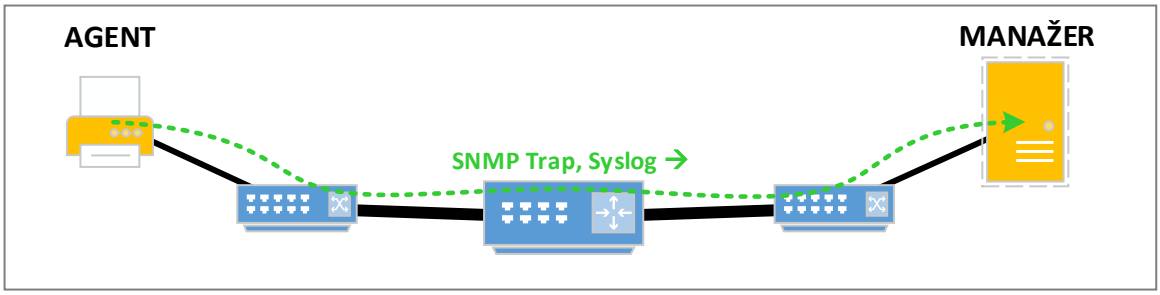
Aktivní monitorování představuje komunikaci mezi Manažerem a Agentem, kdy se Manažer dotazuje Agenta na jeho stav a Agent pošle odpověď, viz obrázek 5. Tento typ komunikace se také někdy nazývá polling (vyzývání) a je nejčastěji používán monitorovacími systémy jako například Nagios, Zabbix, Cacti a OpenNMS. Tyto systémy se mohou v pravidelných intervalech dotazovat zařízení například pomocí protokolu SNMP nebo ICMP, generovat průběžná hlášení o stavu a zobrazovat zjištěné informace jak v textové, tak v grafické podobě. (Matoušek, 2014, s. 359)



Obrázek 5 - Aktivní monitorování
Zdroj: Autor

3.2.2 Pasivní monitorování

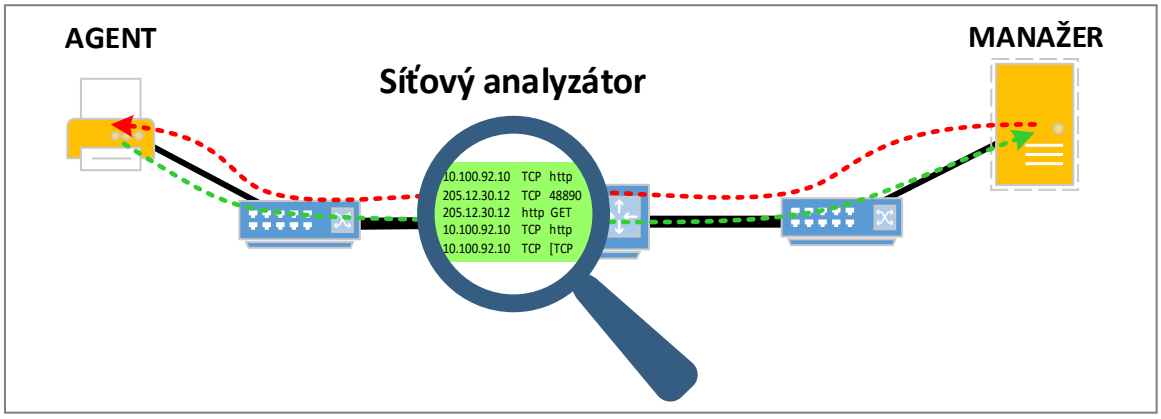
Při pasivním monitorování probíhá asynchronní komunikace, viz obrázek 6, kterou zahajuje Agent při dosažení určitého stavu (změna stavu, překročení limitní hodnoty, neočekávaný stav zařízení, chybový stav apod.). Tento typ komunikace se také někdy nazývá event reporting (hlášení událostí). Jedná se především o zasílání logovaných informací služeb a aplikací pomocí protokolu Syslog, zasílání dat ze sond NetFlow nebo zasílání zpráv typu SNMP Trap. (Matoušek, 2014, s. 359)



Obrázek 6 - Pasivní monitorování
Zdroj: Autor

3.2.3 Analýza síťových dat

Analýza přenášených síťových dat představuje podrobnější zkoumání a vyhodnocování nasbíraných dat rozborem jednotlivých protokolů, viz obrázek 7. Analýza dat umožňuje identifikovat možné útoky nebo škodlivé aktivity, zjistit využití sítě, vyhledávat neefektivní a nezabezpečené aplikace nebo chybně nakonfigurovaná síťová zařízení. K analýze se využívá tzv. sniffer (síťový analyzátor), který umožňuje sledování dat v reálném čase včetně zachycení dat pro následnou analýzu jako například Wireshark, Tcpdump a Free Network Analyzer. (Matoušek, 2014, s. 360)



Obrázek 7 - Analýza síťového provozu
Zdroj: Autor

3.3 Nástroje a protokoly pro monitorování

Monitorovací systémy využívají různé nástroje a protokoly pro zjištění informací o stavu sítě, síťových zařízení nebo služeb. Mezi nejpoužívanější nástroje a protokoly patří:

NetFlow – protokol pro sledování síťových toků umožňující odhalit úzká místa v síti, odhalovat vnitřní a vnější útoky, sledovat jakým protokolem a jak dlouho spolu strany komunikují a zjišťovat dominantní zdroje provozu (Kretchmar, 2003, s. 84-87),

Syslog – protokol pro sběr a přeposílání logů po síti umožňující koncentraci logů ze síťových zařízení a jejich aplikací na centrální Syslog server (Matoušek, 2014, s. 388),

Wireshark – samostatná aplikace pro zachycení, konverzi a následnou analýzu dat, která umožňuje interpretovat jak síťové protokoly (IP, ICMP), tak i protokoly transportní vrstvy (UDP, TCP) a aplikační vrstvy (DNS, http, SNMP, SIP). V zařízení, kde je tato aplikace nainstalována, probíhá zachycení dat pomocí síťové karty v tzv. promiskuitním režimu. V promiskuitním režimu dokáže síťová karta naslouchat nejen datům, která jsou jí adresována, ale i celému provozu v síťovém segmentu. (Sanders, 2012, s. 11-22)

Existují další nástroje pro monitorování sítí jako například MRTG, Neo, Oak, Tcpdump, Netcat a Traceroute, o kterých se lze dočíst v (Kretchmar, 2003).

V této DP je věnována pozornost nástrojům pro monitorování, založeným především na software s otevřeným kódem (open source) a svobodném software (free software) pod licencí GNU GPL a BSD, více o tomto typu software se lze dočíst v (SPI, ©1997-2017) a (Free Software Foundation, Inc., ©2014-2016).

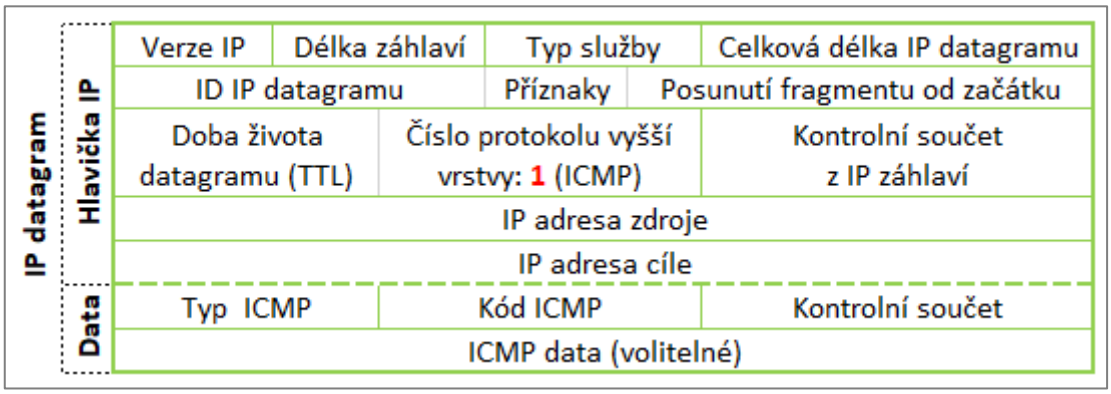
Pro potřeby DP se dále podrobněji zaměříme na protokoly ***ICMP*** a ***SNMP***.

3.3.1 ICMP

Protokol ICMP (Internet Control Message Protocol) je obslužný protokol na úrovni síťové vrstvy, který slouží pro signalizaci mimořádných událostí v TCP/IP sítích. Samotný IP protokol neobsahuje žádný mechanismus pro hlášení chyb. Protokol ICMP byl navržen tak, aby kompenzoval právě nedostatky IP protokolu. (Forouzan, 2010, s. 245)

Protokol ICMP je součástí IP protokolu dle RFC 792, ale chová se jako protokol vyšší vrstvy, který je identifikován číslem 1 (například TCP má číslo vyšší vrstvy 6 a UDP

číslo 17). Je to v podstatě IP datagram, který v datech obsahuje zprávu ICMP, viz obrázek 8. (Kabelová, 2012, s. 134-135)



Obrázek 8 - IP datagram ICMP protokol
Zdroj: Autor dle (Matoušek, 2010, s. 362)

ICMP zpráva obsahuje tato pole:

- Typ (Type) – klasifikace/typ zprávy dle RFC 792,
- Kód (Code) – dílčí specifikace zprávy dle RFC 792,
- Kontrolní součet (Checksum) – ověření, zda nedošlo k narušení obsahu ICMP zprávy,
- Nepovinná data (Variable) – doplňující volitelné informace.

Zprávy ICMP lze rozdělit do dvou základních kategorií: zprávy o chybách a dotazovací zprávy, viz tabulka 1. Zprávy o chybách ohlašují problémy, které mohou vzniknout při zpracování IP datagramu po síti. Dotazovací zprávy jsou párové zprávy typu dotaz/odpověď – polling. (Forouzan, 2010, s. 246)

Tabulka 1 - Kategorie a typy ICMP

Kategorie	Typ ICMP	Zpráva dle RFC	Poznámka
Zprávy o chybách	3	Destination Unreachable	Nedostupný cíl (podrobněji dle ICMP Kódu: 0=síť, 1=hostitel, 2=port atd.)
	4	Source Quench	Žádost o snížení toku dat (zahlcení sítě)
	5	Redirection	Přesměrování datagramu
	11	Time Exceeded	Vypršení času (hodnota TTL=0)
	12	Parameter Problem	Chybný parametr
Dotazovací zprávy	8 0	Echo Echo Reply	Žádost o echo Echo odpověď
	13 14	Timestamp Timestamp Reply	Žádost o časovou synchronizaci Odpověď s časovou synchronizací
	15 16	Information Request Information Reply	Žádost o informaci (IP adresa zdroj a cíl) Odpověď s informací (IP adresa zdroj a cíl)

Zdroj: Autor dle (Forouzan, 2010, s. 246)

Velmi oblíbený nástroj síťových administrátorů pro rychlou diagnostiku síťových problémů je program ***ping***. Tímto programem je možné zjistit dostupnost určitého síťového zařízení včetně statistik o parametrech sítě mezi zařízením odkud se ping spouští a cílovým zařízením. Ping ke své činnosti využívá protokol ICMP. Cílovému zařízení posílá ICMP zprávy Typ 8 (Echo) s určitou velikostí datagramu a čeká na odpovědi Typ 0 (Echo Reply). Na základě toho, zda odpověď dorazí a za jak dlouho, vypíše průběžně odpovědi, které došly a vyhodnotí celkovou statistiku odpovědí, viz obrázek 9. (Kretchmar, 2003, s. 154)

```
C:\Users>ping pef.czu.cz

Pinging pef.czu.cz [193.84.47.38] with 32 bytes of data:
Reply from 193.84.47.38: bytes=32 time=2ms TTL=55
Reply from 193.84.47.38: bytes=32 time=6ms TTL=55
Reply from 193.84.47.38: bytes=32 time=3ms TTL=55
Reply from 193.84.47.38: bytes=32 time=2ms TTL=55

Ping statistics for 193.84.47.38:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 2ms, Maximum = 6ms, Average = 3ms
```

Obrázek 9 - Ping
Zdroj: Výstup z příkazového řádku OS Windows

Ping je součástí většiny OS (Operačních Systémů), spouští se z příkazového řádku a lze ho modifikovat pomocí různých parametrů. V různých typech OS se mohou parametry a výchozí nastavení programu lišit, a proto je dobré si nejdříve spustit nápovědu parametrem – h (help). Například výchozím nastavením pingu ve většině OS Linux je opakované zasílání dotazů do doby, než se stiskne klávesová zkratka CTRL+C. Ping v OS Windows se takto chová až po přidání parametru – t (výchozí jsou pouze 4 dotazy). (Kretchmar, 2003, s. 155)

3.3.2 SNMP

Protokol SNMP (Simple Network Management Protocol) je protokol na úrovni aplikační vrstvy, určený pro správu a monitorování síťových zařízení v TCP/IP sítích. První verze SNMP z r. 1988 vychází z protokolu SGMP (Simple Gateway Monitoring Protocol), původně jako přechodný mechanismus, než dojde k přechodu na propracovaný ISO/OSI model se stejně propracovaným managementem pro správu. Jak již bylo výše zmíněno, model ISO/OSI nebyl v praxi plně nasazen, proto se s protokolem SNMP setkáváme převážně na všech aktivních prvcích dodnes (Matoušek, 2014, s. 369). Hlavní výhoda protokolu SNMP spočívá v jeho jednoduchosti a v tom, že je obecně považován za standard.

Tím může být implementován i do jednoduchých zařízení (teplotních čidel, IP hodin, tiskáren apod.) a dělá z něj atraktivní nástroj pro monitorování a administraci zařízení různých výrobců. (Kretchmar, 2003, s. 26)

V současnosti jsou dostupné tři následující verze protokolu SNMP:

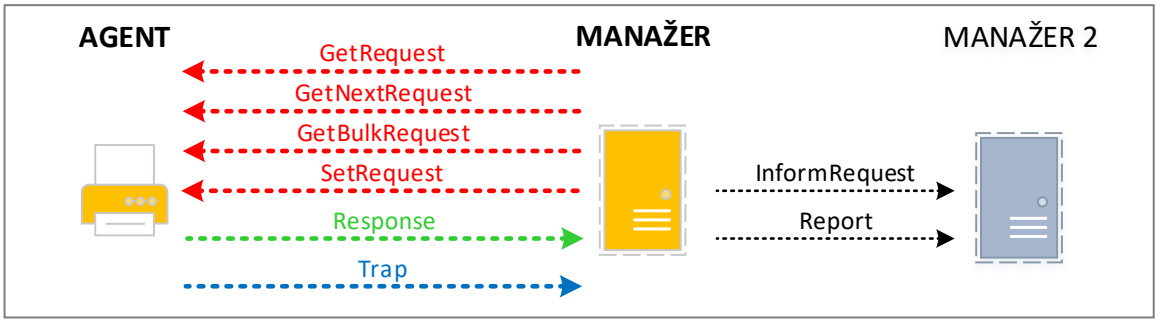
SNMPv1 – vychází u IETF jako doporučení RFC 1157 (r. 1990), který nahrazuje původní RFC 1098 (r. 1989) a RFC 1067 (r. 1988). Bezpečnost SNMPv1 je založena na komunitách (community name) čili heslech, které jsou pouhým textovým řetězcem.

SNMPv2 – od roku 1993 je vydáváno několik RFC s označením v2. Nejdříve je vydáno RFC 1452, které definuje rozdíly od verze 1. RFC 1901 (r. 1996) definuje verzi SNMPv2c (Community-based SNMPv2), která využívá stejné zabezpečení jako verze 1 jen s rozdílem, že místo názvu „community name" se začalo používat „community string". Následně pak RFC 1910 (r. 1996) přichází s verzí SNMPv2u (User-based Security Model SNMPv2), která obsahovala zvýšenou úroveň bezpečnosti (autentizace), ale není výrobci zařízení implementována. Proto, když v praxi mluvíme o verzi SNMPv2, máme na mysli ***SNMPv2c***. Obecně verze 2 přináší schopnost požadavku většího množství dat, možnost komunikace manager-manager a rozšířenou verzi SNMP Trapv2.

SNMPv3 – v roce 2002 pak vychází několik doporučení pro verzi 3 (RFC 3410-3417), především pak RFC 3414. Verze 3 navazuje na SNMPv2u a obsahuje bezpečnostní mechanismus pomocí autentizace odesílatele prostřednictvím algoritmu MD5 (Message-Digest algorithm) a SHA (Secure Hash Algorithm). V roce 2004 pak RFC 3826 definuje šifrování PDU pomocí AES (Advanced Encryption Standard). (Mauro, 2005, s. 2-3)

Architektura systému SNMP vychází z modelu komunikace Agent/Manažer. ***Manažer*** představuje aplikaci, která spouští SNMP metody (zprávy, požadavky) směrem k Agentovi. Odpovědi od Agenta pak Manažer sbírá, zpracovává, ukládá statistiky a prezentuje výsledná data. Vzhledem k tomu, že se protokol SNMP přenáší nespolehlivým transportním protokolem UDP (port 161), je vhodné, aby Manažer obsahoval mechanismy opakování požadavků a časových limitů, které eliminují nedostatky UDP. ***Agent*** představuje běžící proces na monitorovaném zařízení, který zajišťuje sběr informací o SNMP objektech zařízení a konfiguraci. Tyto objekty mohou být pro čtení nebo i pro zápis. Manažer tedy

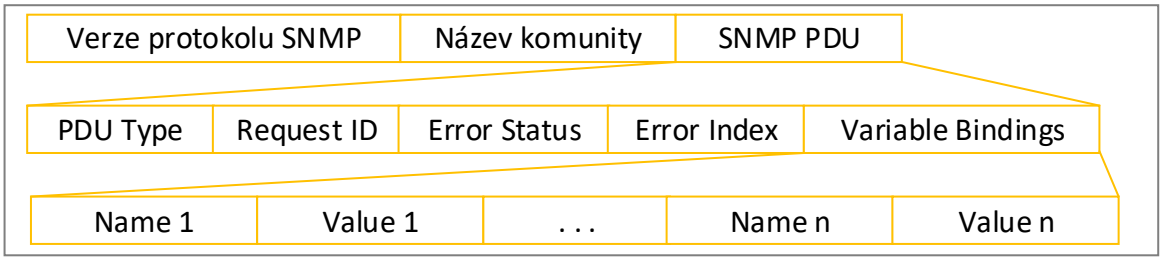
může pomocí SNMP metod nejen získávat informace o těchto objektech, ale může měnit i jejich proměnnou. Databázi všech SNMP objektů v zařízení nazýváme ***MIB*** (Management Information Base). Každý objekt má svůj unikátní číselný identifikátor ***OID*** (Object IDentifier). Přehled komunikace pomocí SNMP protokolu je znázorněn na obrázku 10, podrobněji pak v tabulce 2. (Matoušek, 2014, s. 369-370)



Obrázek 10 - SNMP komunikace
Zdroj: Autor dle (Forouzan, 2010, s. 720)

Kromě polling komunikace umožňuje protokol SNMP i asynchronní typ komunikace, která se nazývá ***SNMP Trap*** (UDP port 162). V tomto případě zasílá Agent zprávy při výskytu mimořádných událostí. Verze protokolu SNMPv2 přináší i novou verzi Trapv2, která v případě architektury monitorování s více Manažery umožňuje zasílání událostí mezi Manažery. (Kozierok, 2005, s. 1110)

Každá SNMP zpráva obsahuje tři základní údaje. Prvním údajem v hlavičce je číselné označení ***verze protokolu SNMP*** (SNMPv1=0, SNMPv2c=1, SNMPv2u=2, SNMPv3=3). Druhý údaj v hlavičce obsahuje ***název komunity***. Komunita slouží jako jednoduchá metoda autentizace, pomocí které je Manažerovi umožněno získat přístup k objektům zařízení. Obvykle existují tři komunity: pouze pro čtení, pro čtení a zápis, a pro SNMP Trapy. Třetí údaj ***SNMP PDU*** je pak tělem zprávy, viz obrázek 11. (Kozierok, 2005, s. 1119-1131)



Obrázek 11 - Zpráva SNMP
Zdroj: Autor dle (Kozierok, 2005, s. 1119-1121)

Tabulka 2 - Typy SNMP PDU

Typ	Název metody	Význam	Verze SNMP		
0	GetRequest	Požadavek na hodnotu proměnné	1	2	3
1	GetNextRequest	Požadavek na hodnotu proměnné dalšího objektu	1	2	3
2	Response	Odpověď na dotaz	1	2	3
3	SetRequest	Požadavek na nastavení hodnoty proměnné	1	2	3
4	Trapv1	Hlášení událostí (nahrazen Trapv2)	1		
5	GetBulkRequest	Požadavek na blok hodnot		2	3
6	InformRequest	Zasílání hodnot proměnných od Agentů pro kontrolu		2	3
7	Trapv2	Hlášení událostí verze 2		2	3
8	Report	Zasílání událostí mezi Manažery		2	3

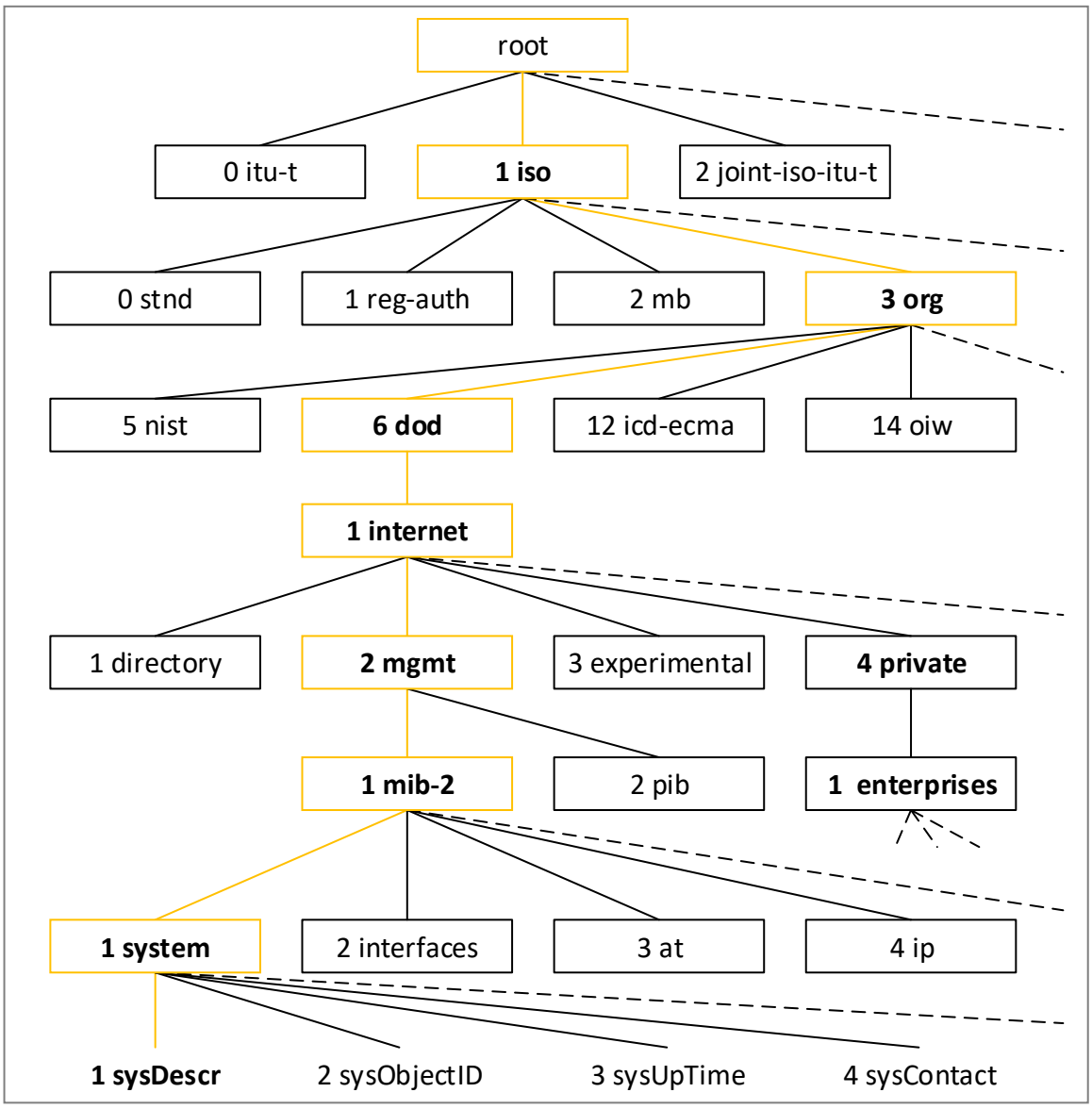
Zdroj: Autor dle (Clark, 2003, s. 384-386)

SNMP PDU se dále skládá z:

- PDU Type – číselné označení typu PDU, viz tabulka 2.
- Request ID – identifikátor požadavku je číslo, které se používá k párování požadavků s odpověďmi (stejné číslo u požadavku i odpovědi).
- Error status – číselné označení chybového stavu zaslané v odpovědi Agenta. Je definováno až 19 typů chyb. Verze SNMPv1 obsahuje: (0=bez chyby, 1=dlouhá odpověď nelze přenést, 2=požadovaný objekt nenalezen, 3=chybný datový typ v požadavku, 4=objekt je pouze ke čtení, 5=jiná chyba). Informace o dalších chybách lze nalézt v (Kozierok, 2005, s. 1127).
- Error index – pokud nastane chybový stav, obsahuje toto pole ukazatel na objekt, který generoval chybu.
- Variable Bindings – variabilní pole párových hodnot objektů MIB (OID, hodnota), které může obsahovat jednu nebo více párových hodnot (v případě GetRequest požadavků je hodnota rovna 0, v případě SetRequest obsahuje požadovanou hodnotu k nastavení proměnné objektu, v Response odpovědi je to hodnota proměnné objektu).

Výše zmíněná struktura PDU odpovídá nejpoužívanější verzi protokolu SNMPv1 a SNMPv2c pro pollingové zprávy. PDU SNMPv3 obsahuje více polí jak v hlavičce, tak v těle zprávy, která slouží k autentizaci a šifrování, více lze nalézt v knize (Kozierok, 2005, s. 1130). Speciálním případem je PDU Trap verze 1, který má odlišnou strukturu a obsahuje například samostatné pole s kódy chybových hlášení. Verze Trapv2 již používá stejnou strukturu PDU jako pollingové zprávy s tím, že chybová hlášení jsou obsažena v těle SNMP zprávy, více v knize (Clark, 2003, s. 390).

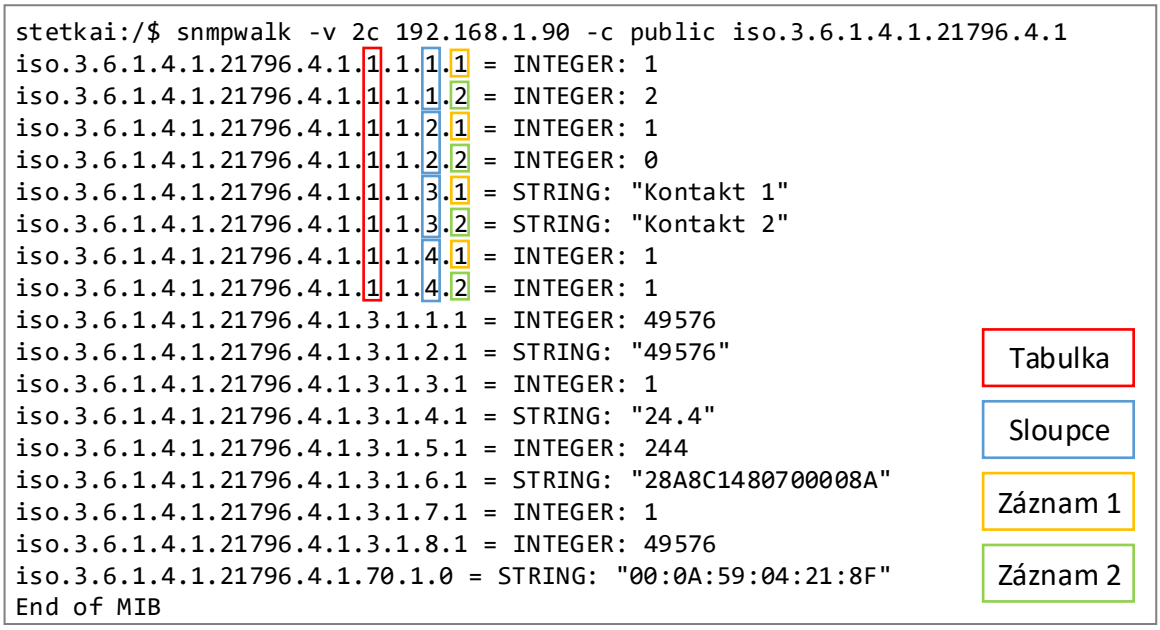
Databáze objektů MIB je strukturována hierarchicky v podobě obráceného stromu, kde listy představují konkrétní objekty s proměnnými. Objekt je identifikován buď textovým názvem nebo číselným OID. Například *iso.org.dod.internet.mgmt.mib-2.system.sysDescr* odpovídá číselným OID *1.3.6.1.2.1.1.1*, viz obrázek 12.



Obrázek 12 - Hierarchie objektů MIB
Zdroj: Autor dle (Orange SA, ©2018)

MIB je ve skutečnosti textový soubor, který obsahuje seznam SNMP objektů včetně popisu proměnných. Struktura MIB je v souladu se strukturou SMI (Structure of Management Information). SMI definuje jména objektů a specifikuje k nim příslušné datové typy a jejich chování. Datové typy a syntaxe objektů jsou definovány pomocí nezávislé abstraktní syntaxové notace ASN.1 (Abstact Syntax Notation One). To umožňuje

výměnu dat mezi Agentem a Manažerem bez ohledu na vnitřní datovou reprezentaci OS. Objekty MIB mohou obsahovat buď jednoduchou proměnnou, anebo vícenásobné proměnné, které lze prezentovat do tabulek a je možné je na první pohled rozeznat. Jednoduchá proměnná má na konci OID číslici 0. U tabulkových proměnných představuje předposlední číslo OID sloupce a poslední číslo začínající vždy číslicí 1 jednotlivé záznamy, viz obrázek 13.

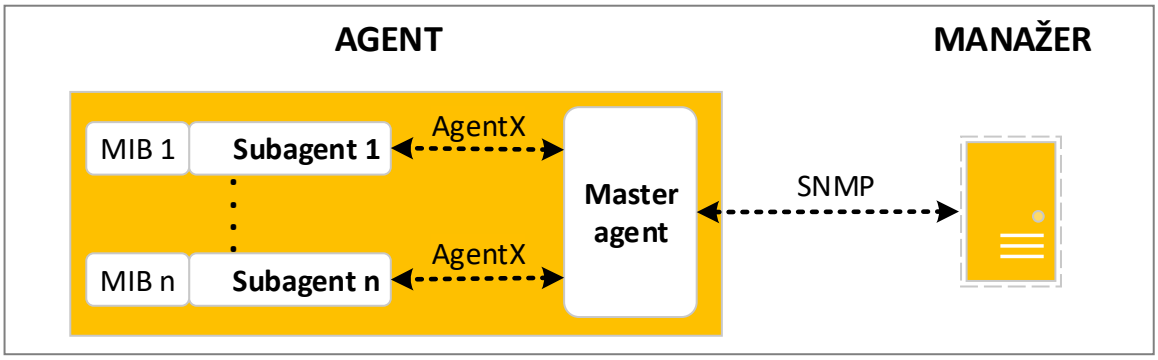


Obrázek 13 - Net-SNMP
Zdroj: Výstup z příkazového řádku Nagios pomocí programu snmpwalk (Net-SNMP)

MIB není pouze jedna databáze objektů. Všechna současná zařízení podporující protokol SNMP by měla obsahovat základní MIB, tzv. MIB-II, vydanou dle RFC 1213. Pozice MIB-II se nachází na OID 1.3.6.1.2.1 iso.org.dod.internet.mgmt.mib-2, viz obrázek 12. MIB-II obsahuje objekty se základními informacemi o OS, síťových rozhraních, směrovacích tabulkách a datech popisujících přenos protokolů IP, ICMP, TCP, UDP, EGP a SNMP. Monitorované zařízení může dále obsahovat i privátní MIB takzvané Vendor MIB. Tyto MIB vydávají samotní výrobci síťových zařízení a SW, ve kterých jsou popsány objekty potřebné ke správě a monitorování těchto zařízení a SW. Pro tyto privátní MIB je vyčleněná pozice uzlu začínající na OID 1.3.6.1.4.1 iso.org.dod.internet.private.enterprises. (Mauro, 2005, s. 23-27), (Matoušek, 2014, s. 373, 376)

Všechny doposud zmíněné doporučení RFC jsou dostupné na portálu IETF Tools (*IETF Documents*, 2017).

Pro potřeby rozšiřování funkcí SNMP Agenta byl IETF definován protokol ***AgentX*** (Agent Extensibility Protocol) dle RFC 2741. Rozšiřováním funkcí Agenta máme na mysli především přidání, změny MIB nebo získávání hodnot z externího zdroje. Architektura Agenta při použití protokolu Agentx je znázorněna na obrázku 14.



Obrázek 14 - Architektura AgentX
Zdroj: Autor dle (Daniele et al., 2000)

Hlavní Agent je zde označován jako Master agent, který komunikuje s jednotlivými Subagenty pomocí protokolu AgentX a s Manažerem standardně protokolem SNMP. Jednotliví Subagenti předávají informace o objektech konkrétní MIB. Master agent bývá běžně dostupný univerzální Agent, například od výrobců OS, který podporuje standardní MIB. Subagenti jsou vyvíjeni konkrétními výrobci zařízení nebo SW. Tito výrobci mají k dispozici sadu nástrojů na podporu vývoje Subagentů API (Application Programming Interface). Tato architektura tak umožňuje výrobcům dynamicky rozšiřovat nebo upravovat konkrétní MIB za běhu Master agenta, aniž by ovlivnili ostatní Subagenty a komunikaci Master agenta s Manažerem. (Daniele et al., 2000)

V následujících odstavcích jsou popsány nástroje využívající SNMP protokol, které budou použity v praktické části této DP.

Net-SNMP - balíček programů pro správu sítě pro prostředí příkazového řádku využívající protokol SNMP (SNMP v1, SNMP v2c a SNMP v3) a protokol AgentX. Net-SNMP je součástí mnoha OS včetně většiny distribucí Linuxu, FreeBSD, Solarisu a OS X. Existuje i verze pro OS Windows. Nejpoužívanější programy balíčku Net-SNMP:

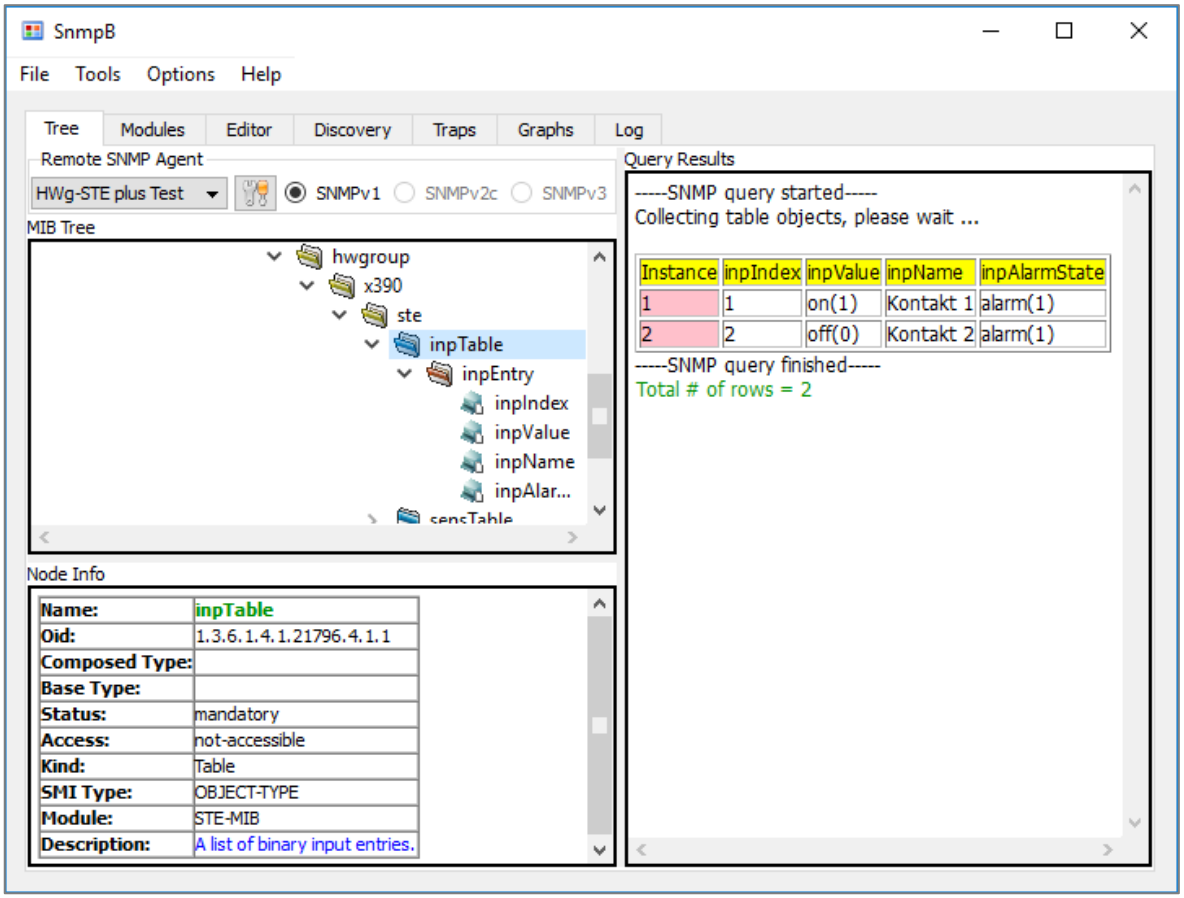
snmpget, snmpgetnext – získává informace o jednoduchých proměnných,
snmpwalk, ***snmptable***, ***snmpdelta*** – získává informace o vícenásobných proměnných,
snmpset – umožňuje nastavovat proměnné objektů,
snmpdf – získává informace o discích zařízení,

snmpstatus – vypíše obecné údaje o zařízení,

snmptranslate – zobrazí číselný a textový popis OID.

Na obrázku 13 je uveden příklad použití programu ***snmpwalk***, který umožňuje procházení více objektů MIB za sebou. Další informace o Net-SNMP lze získat v (Kretchmar, 2003, s. 37-46) a (*Net-SNMP*, 2013).

SnmpB - grafická aplikace pro prohlížení a editaci MIB databází s možností přidávání nových MIB souborů včetně SNMP dotazů na zařízení. Aplikace SnmpB také umožňuje vyhledávání SNMP Agentů v síti, přijímání SNMP Trapů, vykreslování grafů a zobrazení tabulek z přijatých hodnot. Aplikace je dostupná pro většinu OS Linux, FreeBSD, OS X a Windows (Slashdot Media, ©2018). Na obrázku 15 je uveden stejný příklad jako v případě příkladu programu snmpwalk Net-SNMP, nyní ale v grafické podobě aplikace SnmpB (včetně doplňujících textových popisů z MIB).



Obrázek 15 - SnmpB
Zdroj: Výstup aplikace SnmpB (Table View)

3.4 Monitorovací systém Nagios

Na úvodní definici systému Nagios nejlépe poslouží citace z webových stránek tohoto projektu: „*Nagios je výkonný monitorovací systém, který umožňuje organizacím identifikovat a vyřešit problémy s infrastrukturou informačních technologií před tím, než ovlivní kritické obchodní procesy*“ (Nagios Enterprises, ©2009-2018a).

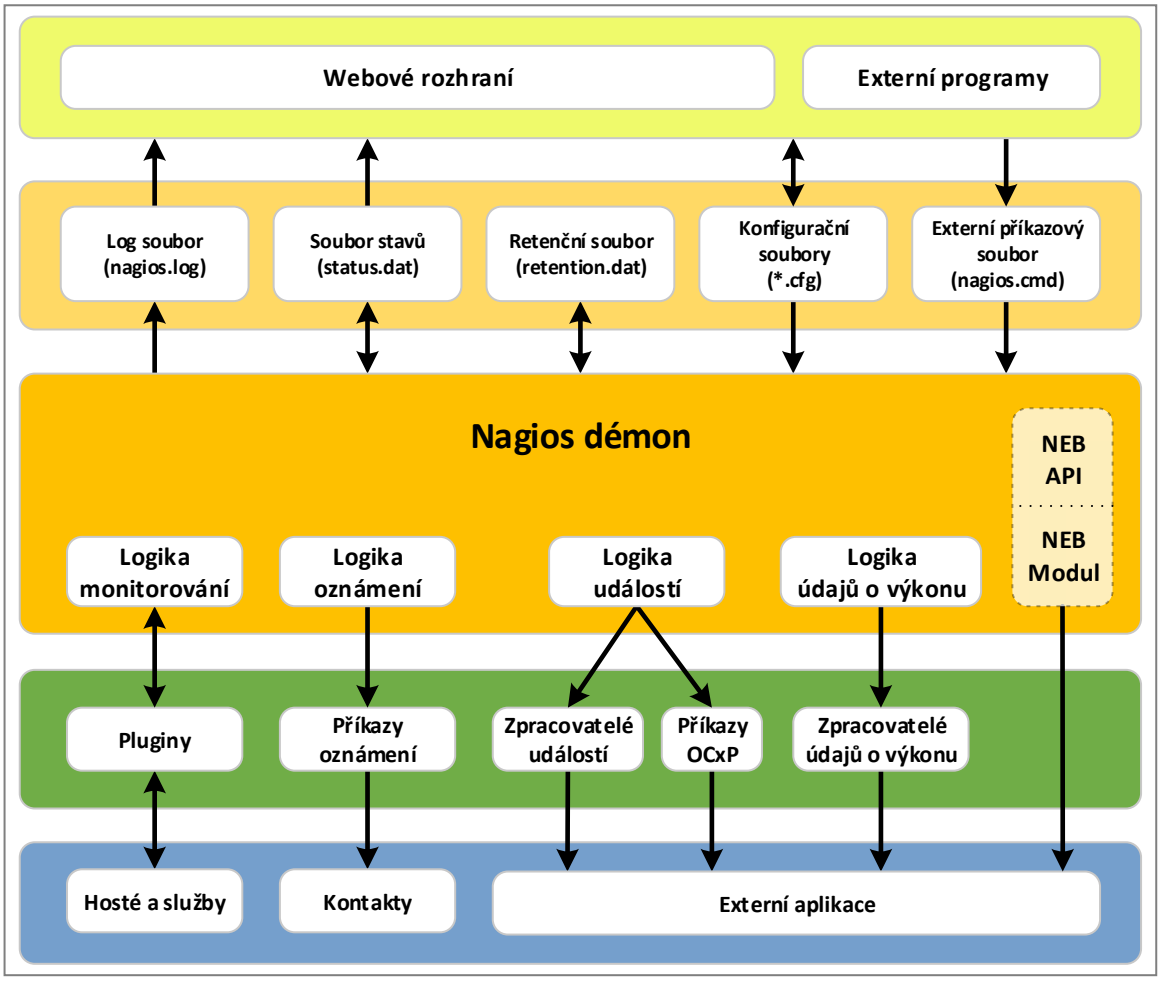
Vznik systému Nagios se datuje až do roku 1999, kdy ***Ethan Galstad*** přichází s Open Source projektem NetSaint, který byl v roce 2002 přejmenován na Nagios. Tento systém byl v následujících letech několikrát vyhlášen různými internetovými portály zabývajícími se tématikou Open Source řešení a Linux OS, jako „Síťový monitorovací systém roku“. V roce 2007 zakládá Galstad společnost Nagios Enterprises, kde dodnes působí jako prezident společnosti. Nagios je dostupný ve dvou variantách: Nagios Core a od roku 2009 Nagios XI. ***Nagios Core***, někdy také označován jako Nagios Core DIY (Do It Yourself), je Open Source systém licencovaný za podmínek GNU GPL verze 2. Tato varianta umožňuje poskládat si zdarma celý monitorovací systém, podle představ každého síťového administrátora, pomocí Open Source modulů a doplňků (tzv. Addons). Konfigurace systému je založena na textových konfiguračních souborech. Tím vyžaduje vyšší odbornou znalost a více úsilí při nasazování systému. V současnosti je dostupná verze Nagios Core 4. Naproti tomu ***Nagios XI*** představuje licencovanou placenou „krabicovou“ verzi s možnou podporou od společnosti Nagios Enterprises. Nagios XI obsahuje webovou grafickou nadstavbu s možností konfigurace celého systému a další moduly ulehčující konfiguraci a práci se systémem. (Nagios Enterprises, ©2009-2018a)

V následujících kapitolách se budeme především věnovat verzi systému Nagios Core (dále jen Nagios).

3.4.1 Architektura Nagiosu

Nagios byl navržen pro instalaci na OS Linux, ale je podporován i na většině Unixových OS. Na manuálových webových stránkách (Nagios Enterprises, ©2009-2018b) jsou uvedeny podrobné instalační návody Nagiosu na tyto OS: Red Hat Enterprise Linux (RHEL), CentOS, Oracle Linux, Ubuntu, SUSE SLES | openSUSE Leap, Debian, Raspbian, Fedora, Arch Linux, Gentoo, FreeBSD, Solaris a Apple OS X.

Nagios umožňuje neustálou kontrolu stavu síťových zařízení (hostitelů) a různých služeb na těchto hostitelích. Na základě těchto kontrol jsou pak generována hlášení. Jádrem Nagiosu je nagios démon (nagios daemon), který pomocí různých logik, konfiguračních souborů, API a skriptů umožňuje integraci dalších modulů, nadstaveb a externích aplikací. To z něj dělá velmi modulární a flexibilní řešení pro monitorování, viz obrázek 16.



Obrázek 16 - Nagios architektura
Zdroj: Autor dle (Nagios Enterprises, ©2009-2018b)

Log soubor – do tohoto souboru jsou zaznamenávány všechny změny stavů služeb a hostitelů, oznámení a události. Tento soubor je možné automaticky ukládat/rozdělovat do menších souborů dle zvolené periody (každou hodinu, denně, týdně a měsíčně).

Soubor stavů – obsahuje aktuální stavy všech sledovaných služeb a hostitelů. Jeho obsah je čten a zpracováván CGI (Common Gateway Interface), díky čemuž mohou být tyto stavy

prezentovány pomocí webového rozhraní. Stavy jsou v tomto souboru aktualizovány dle nastaveného intervalu v hlavním konfiguračním souboru (výchozí nastavení je 10 sekund). Obsah stavového souboru je odstraněn pokaždé, když je Nagios restartován.

Retenční soubor – slouží pro ukládání stavů předtím, než je Nagios zastaven. Při náběhu Nagiosu jsou použity informace uložené v tomto souboru pro nastavení počátečních stavů služeb a hostitelů, než začne Nagios aktivně cokoli sledovat.

Konfigurační soubory – existuje několik konfiguračních souborů, které lze rozdělit do čtyř skupin: hlavní konfigurační soubor, zdrojový soubor, konfigurační soubory objektů a konfigurační soubor CGI, podrobněji v kapitole „Konfigurační soubory“.

Externí příkazový soubor – do tohoto souboru mohou externí aplikace zapisovat příkazy (včetně CGI), které Nagios v pravidelných intervalech načítá a mění různé aspekty svých monitorovacích funkcí na základě těchto příkazů.

Logika monitorování – na rozdíl od mnoha dalších nástrojů pro monitorování, Nagios neobsahuje žádné vnitřní mechanismy pro kontrolu stavu hostitelů a služeb. Toto všechno pro Nagios zajišťují zásuvné moduly (tzv. Pluginy), podrobněji v kapitole „Pluginy“.

Logika oznámení (Notification) – zajišťuje zasílání oznámení při změně stavu hostitele a služby, nebo v případě, že stav, ve kterém hostitel nebo služba setrvává, není „OK“. Nagios může zasílat oznámení různými způsoby: pager, mobilní telefon, e-mail, okamžitá zpráva, zvukové upozornění apod. Jak jsou zasílána oznámení, závisí na příkazech oznámení (notification commands) definovaných v konfiguračních souborech objektů včetně komu mají být zasílána. Podrobnosti o příjemcích jsou pak definovány v objektech kontakty (contact) a skupiny kontaktů (contactgroup).

Logika událostí (Event) – pomocí Zpracovatelů událostí (Event Handlers), volitelnými systémovými příkazy (skripty nebo spustitelné soubory), je možné provádět akce, které se spouštějí při změně stavu hostitele nebo služby. Mezi tyto akce patří například restartování chybné služby, vytvoření záznamu v helpdeskovém systému nebo v databázi, odpojení hostitele nebo jeho části od zdroje (tzv. Power cycling). Funkce příkazů OCSP a OCHP (Obsessive Compulsive Service/Host Processor) umožňuje po spuštění kterékoliv kontroly

hostitele nebo služby spustit další příkaz (dle definice v konfiguračním souboru objektu), což je velmi užitečné při distribuovaném řešení monitorování.

Logika údajů o výkonu (Performance) – údaje o výkonu (Performance Data) jsou data, která jsou získána od hostitelů nebo služeb a je možné je dále zpracovávat. Může se jednat například o vytížení procesoru, vytížení operační paměti, obsazenost disku, počet přenesených dat a reakce na ping. Tato data mohou pak být zpracována externími aplikacemi do podoby grafů a statistických údajů.

NEB (Nagios Event Broker) – představuje API rozhraní, které umožňuje vývojářům vytvářet doplňky a moduly s přímou komunikací s jádrem Nagiosu. API definuje tzv. call-back funkce, které umožňují přístup k interním datům Nagiosu (stavech hostitelů a služeb, oznámeních, událostech apod.). (Nagios Enterprises, ©2009-2018b)

3.4.2 Distribuované monitorování

Cílem distribuovaného monitorování je umožnit Nagiosu monitorovat velkou síťovou infrastrukturu. Ovšem neexistuje univerzální nejlepší řešení distribuovaného monitorování. Je možné si vybrat ze čtyř základních typů distribuovaného monitorování:

Mod-Gearman je populární „load-balancing“ doplněk, který pracuje jak s Nagios Core, tak s Nagios XI. Je vhodný pro poměrně velké síťové infrastruktury, kde je zapotřebí snížit zatížení na monitorovacím serveru tím, že se předávají kontroly na další servery. Mod-Gearman využívá model hlavní server (master) / pracovní uzly (worker nodes), kde jeden hlavní server Nagios rozdělí úlohy do více pracovních uzlů. Hlavní server obsahuje všechny konfigurační a kontrolní definice. Pracovní uzly obsahují pouze pluginy. To zjednodušuje správu systému. Pokaždé, když musí být provedena kontrola, hlavní server ji předá do jednoho z pracovních uzlů, který pak hlásí výsledky zpět. Pracovní uzly mohou být přidány nebo odebrány podle potřeby, jelikož hlavní server zaznamenává, které z nich jsou k dispozici, a odpovídajícím způsobem upraví své přiřazení. Mod-German využívá modul NEB.

Nagios Fusion je komerční distribuované monitorovací řešení, které poskytuje centrální uživatelské prostředí pro více serverů Nagios Core nebo Nagios XI. Fusion umožňuje měnit monitorovací prostředí nasazením dalších serverů Nagios XI nebo Nagios Core pro

sledování dalších hostitelů, služeb a aplikací. Každý server monitoruje část infrastruktury a Fusion poskytuje centrální uživatelské prostředí, které umožní rychle zobrazit stav celé infrastruktury na jednom místě.

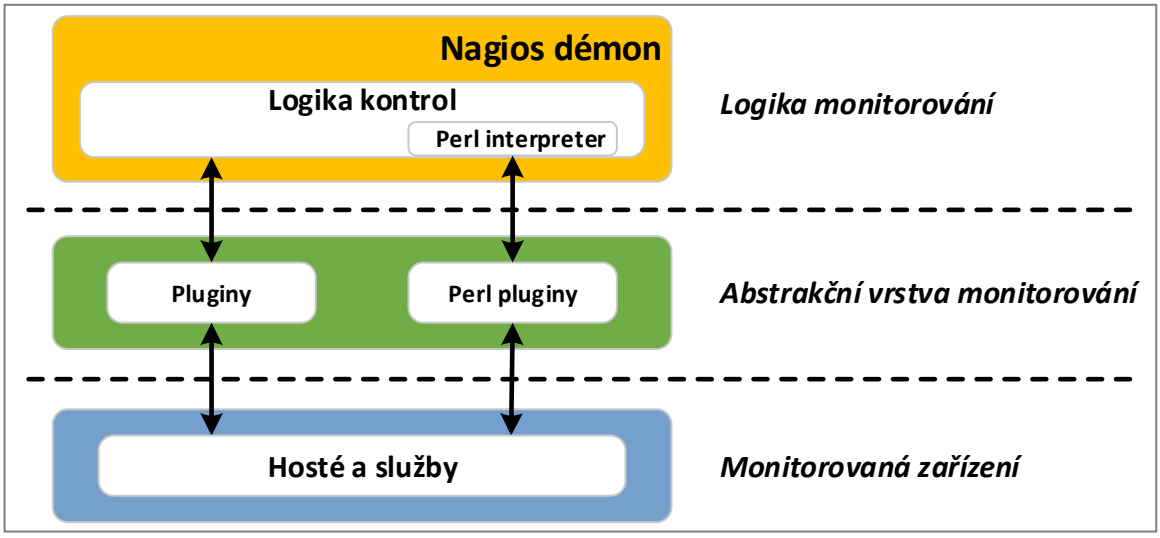
MNTOS (Multi-Nagios Tactical Overview System) je funkčně stejné řešení jako Fusion, jen je vydané pod licencí BSD (svobodný SW). Toto řešení je založené na shromažďování dat z více taktických přehledových webových stránek systémů Nagios, které kompiluje do jediného souboru XML. Soubor XML lze použít libovolným rozhraním podporujícím XML a tím zobrazit data na jedné webové stránce.

Federated Monitoring je řešení pro klienty, kteří využívají služeb externích společností MSP (Managed Service Providers) pro monitorování/správu počítačových sítí. V tomto modelu jsou vzdálené sítě a jejich prvky monitorovány dedikovanými servery Nagios. Každý vzdálený server Nagios může být spravován personálem NOC (Network Operations Center) nebo klientem. Oznámení a konfigurace jsou obecně řešeny každým vzdáleným serverem Nagios. Vzdálené servery Nagios mohou být nakonfigurovány tak, aby přenášely výsledky kontrol zpět na centrální server Nagios XI v NOC. To umožňuje personálu NOC mít pohled z ptačí perspektivy na celou síť a poskytuje jim centralizovaný reporting a volitelná oznámení.

Více o těchto a dalších typech distribuovaného monitorování lze získat na (Nagios Enterprises, ©2009-2018b) a (Nagios Enterprises, ©2009-2018c).

3.4.3 Pluginy

Pluginy jsou kompilované spustitelné soubory nebo skripty (skripty v jazyce Perl, shell skripty apod.), které lze spustit z příkazového řádku. Výstupy z pluginů slouží k určení aktuálního stavu hostitelů a jejich služeb. Nagios spustí plugin (pomocí logiky kontrol dle časových nastavení), plugin vrátí výsledek (na základě vnitřních podmínek a komunikace s hostitelem), Nagios pak zpracuje výsledky a provede veškerá potřebná opatření (odešle oznámení uživateli, skupině uživatelů nebo externí aplikaci). Pluginy fungují jako abstrakční vrstva mezi logikou monitorování přítomnou v démonu Nagiosu a monitorovanými hostiteli a službami, viz obrázek 17.



Obrázek 17 - Nagios abstrakční vrstva pluginů
Zdroj: Autor dle (Nagios Enterprises, ©2009-2018b)

Nevýhodou tohoto typu architektury je skutečnost, že Nagios nemá informace o tom, co monitoruje. Nagios pouze sleduje změny stavů monitorovaných zařízení. Jen v samotných pluginech je nadefinováno, co se monitoruje a jak provádět skutečné kontroly. (Nagios Enterprises, ©2009-2018b)

Existuje oficiální balíček cca 50 pluginů, které byly vytvořeny, a jsou stále vyvíjeny za účelem sledování základních služeb (HTTP, DNS, NTP, MySQL, Ping, SNMP atd.) a údajů o výkonu hostitelů (zatížení procesoru, využití disku atd.). Tyto pluginy jsou dostupné na (Nagios Enterprises, ©2009-2018d). Další pluginy a doplňky jsou dostupné na (Nagios Enterprises, ©2009-2018c), kde komunita vývojářů a programátorů sdílí své projekty. Pokud žádný plugin na zmíněných portálech není vyhovující, je možnost si vytvořit plugin vlastní. Pluginy musí obsahovat minimálně tyto dvě základní vlastnosti:

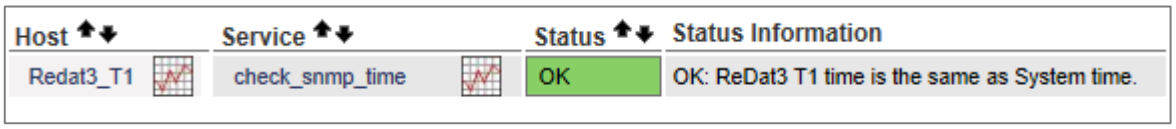
- ukončit svůj běh s návratovou hodnotou, podle které Nagios určuje stav hostitele nebo služby, viz tabulka 3,
- vrátit minimálně jeden řádek textu do STDOUT (standardní výstup Linux/Unix OS).

Tabulka 3 - Návratové hodnoty pluginu

Návratová hodnota pluginu	Stav služby	Stav hostitele
0	OK	UP
1	WARNING	UP nebo DOWN/UNREACHABLE
2	CRITICAL	DOWN/UNREACHABLE
3	UNKNOWN	DOWN/UNREACHABLE

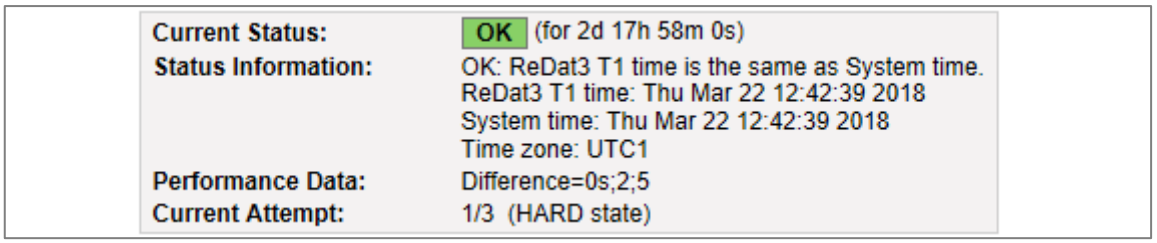
Zdroj: Autor dle (Nagios Enterprises, ©2009-2018b)

Výstupní text pluginu může mimo požadovaný jeden řádek obsahovat údaje o výkonu (Performance Data) a od verze Nagios 3 i víceřádkový výstupní text. Jednotlivé řádky výstupního textu se v pluginu oddělují zalomením na nový řádek pomocí znaku „/n“. Údaje o výkonu následují napravo od oddělovacího znaku svislé čáry „|“. První řádek výstupního textu se na základním webovém rozhraní Nagiosu zobrazuje na stránce hostitele (Status Information), viz obrázek 18. (Nagios Enterprises, ©2009-2018b)



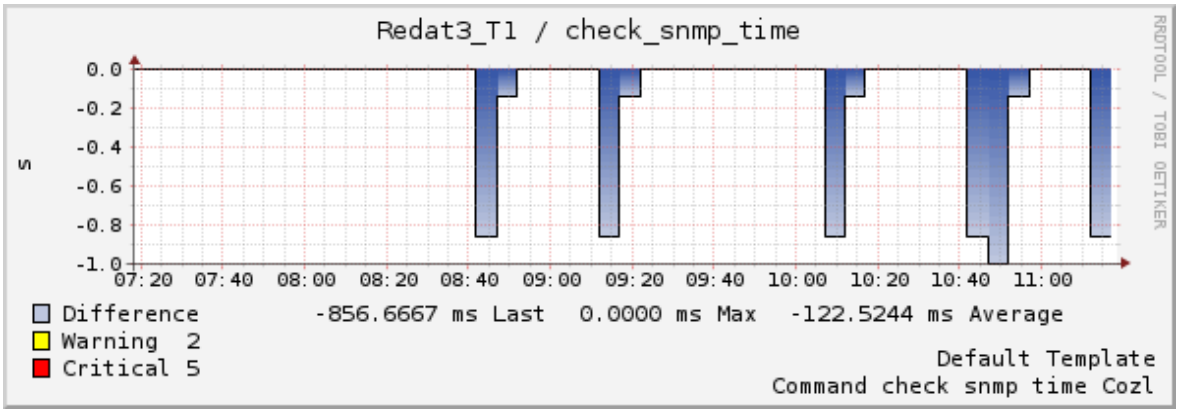
Obrázek 18 - Nagios webová stránka hostitele
Zdroj: Autor, webové rozhraní Nagios (výřez)

Na webové stránce konkrétní služby jsou pak zobrazeny všechny řádky výstupního textu a aktuální údaj o výkonu, viz obrázek 19.



Obrázek 19 - Nagios webová stránka služby
Zdroj: Autor, webové rozhraní Nagios (výřez)

Historii údajů o výkonu lze pak například pomocí doplňku PNP4Nagios zobrazit v grafické podobě, viz následující obrázek 20.



Obrázek 20 - Nagios webová stránka údajů o výkonu
Zdroj: Autor, webové rozhraní Nagios (výřez)

Pro správnou funkci doplňku PNP je nutné, aby plugin vracel údaje o výkonu v požadovaném formátu: ***'label'=value[UOM];[warn];[crit];[min];[max],*** kde

label – název sledovaného údaje, jednoduché uvozovky jsou požadovány, pokud název obsahuje mezery,

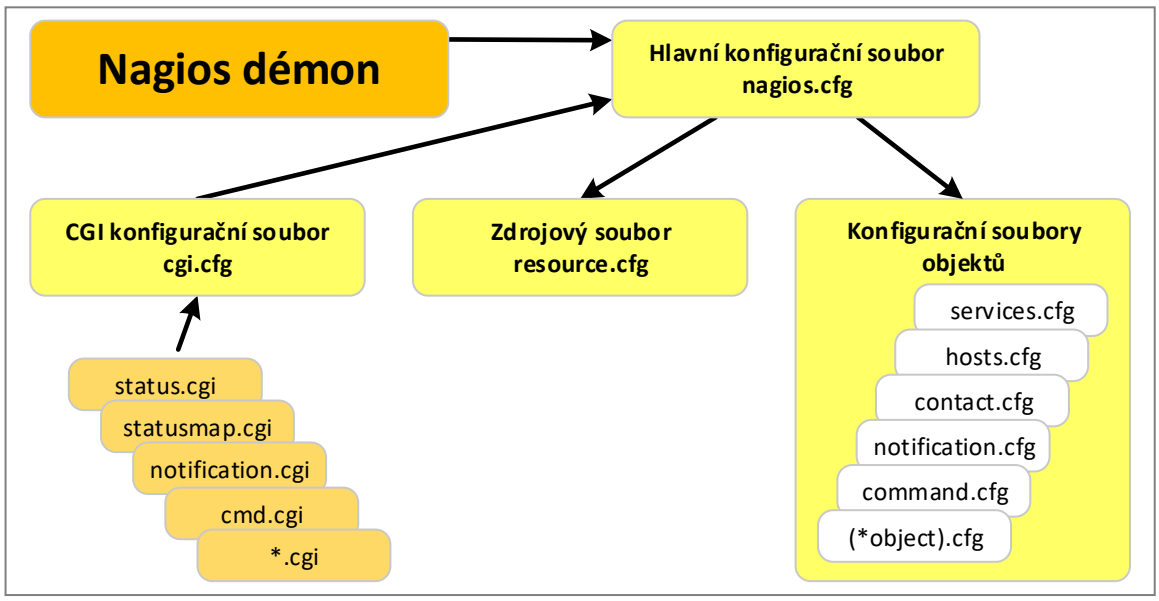
value – číselná hodnota sledovaného údaje,

UOM *(Unit Of Measurement)* – měrná jednotka, může být nespecifikována (počet, průměr apod.) nebo je použita jedna z následujících jednotek: s – sekundy, % - procenta, B – bajty (KB, MB, RB), c – kontinuální čítač (přenášené bajty za časovou jednotku),

varn, ***crit***, ***min***, ***max*** – tyto prahové hodnoty nejsou povinné, více o tomto doplňku se lze dočíst na stránkách projektu. (Linge, 2017)

3.4.4 Konfigurační soubory

Pokud bude při instalaci postupováno dle pokynů instalačních návodů, pak všechny výchozí konfigurační soubory se budou nalézat ve složce „/usr/local/nagios/etc/“. Před samotným spuštěním monitorování je třeba tyto soubory vytvořit nebo upravit. Konfigurační soubory lze rozdělit do čtyř skupin, viz obrázek 21.



Obrázek 21 - Nagios konfigurační soubory
Zdroj: Autor dle (Nagios Enterprises, ©2009-2018b)

Hlavní konfigurační soubor obsahuje řadu direktiv, které ovlivňují fungování démona (základní konfiguraci systému a odkazy na další konfigurační soubory a složky). Tento konfigurační soubor je při spuštění systému Nagios načten jak démonem Nagios, tak CGI.

Zdrojový soubor lze použít k ukládání maker definovaných uživatelem. Hlavním účelem tohoto souboru je ukládání citlivých informací o konfiguraci (jako jsou hesla), aniž by byly k dispozici pro CGI. Je možné definovat jeden nebo více volitelných zdrojových souborů pomocí direktivy resource_file v hlavním konfiguračním souboru.

CGI konfigurační soubor obsahuje řadu direktiv, které ovlivňují činnost CGI souborů, pomocí kterých jsou data prezentována na webovém rozhraní Nagiosu. Obsahuje také odkaz na hlavní konfigurační soubor, takže CGI mají informace o konfiguraci Nagiosu a o konfiguračních souborech objektů. Dále obsahuje výchozí html cestu pro webové stránky Nagiosu, výchozí fyzickou cestu pro html soubory, autentifikační pravidla pro zobrazení webových stránek apod.

Konfigurační soubory objektů se používají k definování hostitelů, služeb, skupin hostitelů a služeb, kontaktů, skupin kontaktů, oznámení, příkazů apod. Jedná se o všechny prvky (objekty), které se podílejí na monitorovací a oznamovací logice. Pomocí nastavení ***cfg_file*** nebo ***cfg_dir*** v hlavním konfiguračním souboru je možné zadat jeden nebo více konfiguračních souborů objektu. (Nagios Enterprises, ©2009-2018b)

3.4.5 Popis vybraných funkcí a nastavení

3.4.5.1 Makra

Jednou z hlavních funkcí, díky nimž je Nagios tak flexibilní, je schopnost používat makra v definicích příkazů. Makra umožňují odkazovat na informace z hostitelů, služeb a dalších zdrojů v příkazech. Před provedením příkazu Nagios nahradí všechna makra, která nalezne v definici příkazu, odpovídajícími hodnotami. (Nagios Enterprises, ©2009-2018b)

Například při definici hostitele, kde jeho dostupnost kontrolujeme příkazem check_ping, přenášíme parametr IP adresy hostitele:

```
define host {
        host_name       pbx_1
        address         192.168.1.50
        check_command   check_ping
        ...
        }
```

pomocí makra $HOSTADDRESS$ do definice příkazu:

```
define command {
        command_name    check_ping
        command_line    /... /check_ping  -H $HOSTADDRESS$ -w 150.0,30% -c 200.0,50%
        }
```

Pokud bychom potřebovali definovat u každého hostitele například jiné parametry –w (warning) a –c (critical) je možné využít maker $ARGn$:

```
define command {
        command_name    check_ping
        command_line    /... /check_ping -H $HOSTADDRESS$ -w $ARG1$ -c $ARG2$
        }
```

Definice hostitele by pak v tomto případě vypadala takto:

```
define host {
        host_name           pbx_1
        address             192.168.1.50
        check_command       check_ping!200.0,80%!300.0,90%
        ...
        }
```

Oddělovacím znakem mezi příkazem a jednotlivými argumenty je pak vykřičník „!“. Při definici služeb bývá právě makro $ARGn$ nejvíce používané, jak si ukážeme v praktické části této DP.

3.4.5.2 Dědičnost

Dědičnost nám umožňuje, při definici objektů, zjednodušit konfigurační soubory na místech, kde používáme ve více objektech stejné vlastnosti. Existují tři proměnné ovlivňující dědičnost, které se vyskytují ve všech definicích objektů:

```
define someobjecttype {
        object-specific     variables
        ...
        name                template_name
        use                 name_of_template_to_use
        register            [0/1]
        }
```

První proměnná „name“ je jméno šablony, na které lze odkazovat v jiných definicích objektů, a tím mohou objekty dědit vlastnosti/proměnné ze šablony. Názvy šablon musí být jedinečné mezi objekty stejného typu, takže nemůžeme mít dvě nebo více definic hostitele, které mají například název šablony „host_template“.

Pomocí druhé proměnné „use“ se zadává název objektu šablony, ze kterého chceme dědit vlastnosti/proměnné. Název, který určíme pro tuto proměnnou, musí být definován jako šablona jiného objektu s názvem (pomocí proměnné „name“).

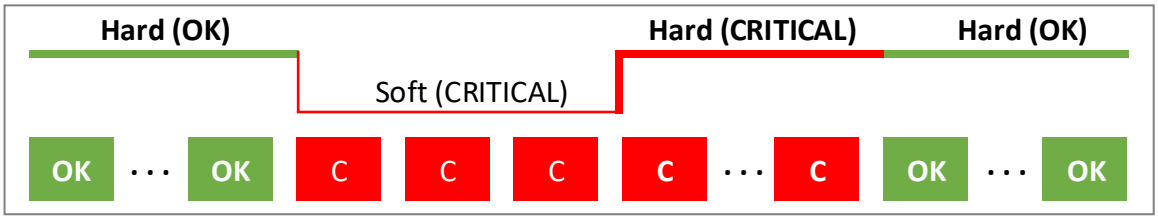
Třetí proměnná „register“ slouží k označení, zda má být definice objektu "registrována" Nagiosem. Ve výchozím nastavení jsou všechny definice objektů registrovány. Pokud používáme definici částečného objektu, jako jsou šablony, musí se tato proměnná nastavit na hodnotu „0“. Je pochopitelné, že se tato hodnota jako jediná nedědí na jiné objekty.

Při používání dědičnosti objektů je důležité pochopit, že lokální vlastnosti/proměnné objektů mají vždy přednost před vlastnostmi/proměnnými definovanými v objektu šablony. (Nagios Enterprises, ©2009-2018b)

3.4.5.3 Soft a Hard state

Při zasílání oznámení používá Nagios funkci tzv. ***Soft state*** a ***Hard state***. Funkce ***Soft state*** umožňuje ignorovat dočasné nebo náhodné výpadky služby nebo hostitele. Je také velmi užitečná pro provádění kontrol, které mohou pravidelně selhat i v případě, že vše funguje správně. Například při sledování zařízení pomocí protokolu SNMP může sice selhat první kontrola, ale nakonec uspěje během druhé nebo třetí kontroly.

Soft State nastává, pokud dojde ke změně ze stavu OK na stav CRITICAL nebo WARNING. Nagios několikrát zkontroluje hostitele nebo službu, aby se ujistil, že změna stavu je trvalá. Počet kontrol je specifikován v konfiguraci hostitele nebo služby. Po vyčerpání tohoto počtu kontrol vyhodnotí Nagios jako ***Hard state*** a může být zasláno oznámení. Při nastaveném počtu kontrol na „1“ bude okamžitě považováno za ***Hard state*** (Kocjan, 2008, s. 14). Opačně pak ze stavu CRITICAL nebo WARNING na stav OK dojde vždy k ***Hard state***. Na následujícím obrázku jsou znázorněny změny ***Soft*** a ***Hard state*** při nastaveném počtu „4“ kontrol (při změně stavů OK na CRITICAL a opačně).



Obrázek 22 - Nagios Soft a Hard state
Zdroj: Autor dle (Nagios Enterprises, ©2009-2018b)

3.4.6 Programovací jazyk Perl pro tvorbu vlastních pluginů

Převážná většina dostupných pluginů od komunity vývojářů je tvořena v programovacím jazyce Perl. Proto byl tento jazyk zvolen i pro vlastní pluginy v této DP.

Perl je interpretovaný programovací jazyk, což znamená, že vytvořený program (plugin) se nepřekládá, ale vykonává se přímo ze zdrojového kódu. To má výhodu především při modifikaci a testování pluginu. Aby bylo možné plugin spustit, je nutné mít nainstalovaný interpret Perlu. Ve většině linuxových distribucí je tento interpret přítomný. Pro testování pluginu na OS Windows je možné nainstalovat interpret Strawberry Perl (*Strawberry Perl*, 2017) nebo přímo vývojové prostředí Padre IDE (Szabo et al., 2016). Dalšími užitečnými nástroji pro editaci pluginu jsou programátorské textové editory jako například PSPad (Fiala, 2017) a Notepad++ (Ho, ©2018).

Perl je velmi benevolentní programovací jazyk. Proměnné se nedeklarují. Chování různých prvků závisí na kontextu použití. Syntaxe je velmi proměnlivá. Hlavní předností Perlu jsou pak regulární výrazy a asociativní pole (hashe), které z něj dělají silný nástroj pro práci s textovými informacemi. Při spouštění pluginu lze jako ochranu proti zmíněné benevolentnosti zadat přepínač „-w“ (direktiva warning) a do těla pluginu vložit direktivu „strict“ pomocí funkce „use“. Parametr „-w“ nastaví interpret tak, aby při každé podezřelé konstrukci vypsal varovné hlášení (například při jediném výskytu proměnné nebo použití proměnné, které nebyla přiřazena hodnota). Velmi užitečné a doporučené je zadat direktivu warning již na začátek těla (hlavičky) pluginu, kde se zadává cesta k interpretu Perl. (Satrapa, 2001, s. 15-17)

```
#!/usr/bin/perl –w

use strict;
```

Takto se bude vytvořený plugin vždy spouštět v módu warning. Další direktiva „strict“ (striktní režim běhu pluginu) zakazuje použití potenciálně nebezpečných konstrukcí (Satrapa, 2001, s. 110):

- nedeklarované proměnné – všechny proměnné, které nejsou deklarované pomocí funkcí my, our, vars nebo importované z modulu,
- holá slova – tzn. neuvozené řetězce a nedeklarované podprogramy,

- symbolické odkazy – proměnná, která je zapsána jako pevný odkaz (reference), ale místo aby obsahovala odkaz na adresu v paměti, obsahuje řetězec.

Při tvorbě pluginu budeme využívat jak jednoduché proměnné (skaláry) tak i složené proměnné (pole a hashe). Všechny proměnné se ve striktním režimu deklarují pomocí funkce „my“. Skaláry mohou obsahovat číslo, řetězec znaků nebo odkaz na jinou proměnnou. Skaláry se označují znakem dolaru „$“ před jménem:

```
my $skalar=3;
```

Pole neboli seznamy mohou obsahovat více hodnot (prvků). Tyto prvky se zapisují oddělené čárkami uzavřené do kulatých závorek a pole se označuje znakem zavináče „@“ před jménem:

```
@pole = ("prvek_1", "prvek_2 ", ..., "prvek_n");
```

Prvky pole jsou určeny tzv. indexem, číselnou hodnotou začínající od „0“. Tzn. že prvek_1 má index „0“, prvek_2 má index „1“ atd. Naproti tomu hashe mohou obsahovat v indexu řetězec. V takovém případě již nemluvíme o indexech, ale o klíčích. Hashe se označují znakem procenta „%“ před jménem a zapisují se takto (Satrapa, 2001, s. 21,57,127):

```
%hash = ("klic_1", "hodnota_1", "klic_2", "hodnota_2", ..., "klic_n", "hodnota_n");
```

Funkce „use“ se nepoužívá pouze pro implementaci různých direktiv, ale i k zavedení modulů. Modul je logicky ucelená skupina podprogramů a proměnných, která řeší určitou část problému (Satrapa, 2001, s. 126-127). Velmi užitečný je archiv modulů CPAN (Comprehensive Perl Archive Network) (Perl.org, ©2005-2018), odkud budou čerpány moduly pro tvorbu pluginů.

Nepostradatelnou dokumentací a inspirací při tvorbě pluginu bude Perl Programming Documentation (*Perl Programming Documentation*, 2017) a PerlMonks (*PerlMonks*, 2018). Při tvorbě regulárních výrazů v pluginu budou tyto výrazy tvořeny a testovány pomocí webového portálu RegExr (Skinner, nedatováno). Pro grafické znázornění algoritmu pluginů budou použity vývojové diagramy dle ČSN ISO 5807 a (Pšenčíková, 2009).

3.4.7 NagVis

NagVis je grafická nadstavba Nagiosu, která umožňuje zobrazovat stavy objektů (Host, Service, Hostgroup, Servicegroup) na tzv. mapách s volitelným grafickým pozadím ve formátu PNG, GIF nebo JPEG. Je čistě na administrátorovi systému, jaké pozadí zvolí, zda použije fotografii konkrétního systému, geografickou mapu, topologii systému nebo například vlastní diagram. Každá mapa může být konfigurována pomocí vlastního konfiguračního souboru. Konfigurační soubory je možné upravit přímo pomocí textového editoru nebo pomocí webového rozhraní, které je vhodnější minimálně pro vkládání ikon samotných objektů. Výchozí ikona pro objekty Host a Hostgroup je čtvercového tvaru. Ikona objektů Service a Servicegroup je kruhového tvaru. Barva ikon se mění v závislosti na stavu objektu: červená pro stav CRITICAL, žlutá pro WARNING, zelená pro OK a otazník na šedém pozadí pro UNKNOWN. Ikony lze nastavit ve dvou velikostech, 16x16 a 9x9 bodů. Je možné do NagVisu také importovat i vlastní ikony. Pro zobrazení, například objemu přenesených dat mezi aktivními prvky, lze použít objekt typu Line. Line lze nastavit tak, aby měnily svou barvu dle procentuální dosažené úrovně. Mimo Line lze do map přidat další speciální objekty NagVis jako jsou Shape, Textbox a Map. Ikona objektu Map slouží jako hypertextový odkaz na jinou mapu, tím lze jednotlivé mapy mezi sebou propojovat. Pokud jsou na odkazované mapě pouze objekty Service, tak má ikona Map podobu ikony Service, v jiném případě má podobu ikony Host.

Pomocí dat dodaných rozhraním z Nagiosu tzv. backendem aktualizuje NagVis objekty umístěné na mapách v určitých intervalech tak, aby odrážely aktuální stav. Je k dispozici několik backendů:

- MKLivestatus,
- NDOUtils / IDOUtils,
- merlin.

Výchozí backend MKLivestatus od vývojáře Mathiase Kettnera je velmi jednoduchý a inteligentní NEB modul, který je koncipován a naladěn pro snížení zatížení disků, paměti a procesorů způsobeného zpracováním živých dat v systému Nagios. Pomocí API NEB jsou data načtena přímo z procesu Nagiosu, proto NagVis pracuje s aktuálnějšími daty, než kdyby

je vyčítal ze stavového souboru Nagiosu. Ostatní backendy vyžadují na straně Nagiosu databázi MySQL, odkud vyčítají data. (Barth, 2008, s389), (NagVis Project, ©2013)

3.5 Telekomunikační systémy

Hlavním úkolem telekomunikačních systémů je přenos informace mezi dvěma či více body. Mezi první takovéto systémy můžeme například zařadit kouřové signály indiánských kmenů, světelné signály v podobě majáků pro námořnictvo a akustické signály domorodých kmenů. Telekomunikační systém v moderním pojetí představuje především elektronickou komunikaci, do které patří telegrafie, radiofonie, rozhlas, televize, počítačové sítě a telefonie. (Prokeš, 2000, s. 15)

3.5.1 Telefonie a telefonní ústředny

Telefonie představuje souhrnný název pro obousměrný způsob přenosu lidského hlasu na velkou vzdálenost v reálném čase. Počátky telefonie sahají do druhé poloviny 19. století. Všeobecně je známo, že v roce 1876 podal patent na první telefon Alexander Graham Bell ve stejný den a o pár hodin dříve než Elisha Grey. Ovšem rezolucí 269 z 11. června 2002 byl Kongresem Spojených států uznán skutečným vynálezcem telefonu ***Antonio Meucci***, jehož telefon byl poprvé předveden již v roce 1860 v New Yorku. Telefonie se uskutečňuje prostřednictvím telefonní technologie. Ta zahrnuje koncová zařízení (telefony, faxy atd.) a přenosové prostředí neboli telefonní síť skládající se z telefonních vedení a telefonních přepínačů či přepínacích systémů – ústředen. (*H. Res. 269*, 2002), (Vozňák, 2009, s. 13), (Strach, 2009)

Ústředna (v telekomunikacích) je souhrn spojovacího a přídavného zařízení uzlu telekomunikační sítě umožňující sestavování jednotlivých spojení podle požadavků individuálních uživatelů (Kapoun, 2000, s. 3). Telefonní ústředny lze dělit v tom nejobecnějším formátu na generace. Generace lze svázat s časovým vývojem celé historie telefonie 20. století (Bazala, 2006, s. 12).

Do ***nulté generace*** řadíme tzv. manuální ústředny, kde řízení spojení provádí spojovatelka. Volající je po vyzvednutí sluchátka spojen se spojovatelkou a sdělí spojovatelce číslo (jméno) volaného. Spojovatelka vyzvoní volaného a po jeho vyzvednutí

ručně propojí dráty volaného a volajícího spojovacím kabelem na ovládacím pultu. Tento typ spojování hovorů se používal koncem 19. a začátkem 20. století. (Bazala, 2006, s. 12)

Ústředny ***první generace*** na základě volby automaticky vyzvoní volaného, spojí a zruší hovor. Jako hlavní spojovací zařízení se zde využívala relátka, automatický krokový volič Almona Strowgera patentovaný již v roce 1891 a další elektromagnetické prvky. Krokový volič funguje na elektromechanickém principu, každým přijatým impulzem z účastnického vedení se rameno voliče přesune do další polohy, krokuje tím v kontaktním poli, kupříkladu po vytočení číslice 8 provede osm kroků (Vozňák, 2009, s. 27). Volba je synchronní, tzn., že budování spojové cesty (účastnického okruhu) probíhá současně s volbou čísla. Účastnický okruh je v telekomunikacích kombinace dvou přenosových kanálů umožňujících přenos mezi dvěma body v obou směrech (Kapoun, 2000, s. 3). Pro propojení více ústředen se zde objevují první přenašeče a jednoduché spojové systémy. Spadá do první poloviny 20. století. (Bazala, 2006, s. 12)

Ústředny ***druhé generace*** jsou zařazeny přibližně do období 70. let a zpravidla se jedná o výkonnější, v praxi nejpoužívanější kapacitní elektromechanické ústředny na bázi křížových spínačů. Křížový spínač je dvoupohybový elektromechanický spínač s elektromagnetickým ovládáním. Ovládání se děje soustavou vertikálních a horizontálních tyčí v místě jejich křížení (Boldiš, 1985, s. 23). Volba je zde již asynchronní, nejprve se uloží do pamětí (řádově desítky bitů pro jedno spojení) a posléze se vytvoří spojová cesta. K velkému rozvoji zde dochází v oblasti přenosových systémů mezi vzdálenými ústřednami, které umožňují automatizaci a integraci národních sítí a vznikají tak prvotní dokonalé telefonní sítě. (Bazala, 2006, s. 13)

Charakteristickým prvkem ústředen ***třetí generace*** je úplné nahrazení elektromechanických součástek za polovodičové elektrosoučástky. Ústředny mají již centrální řízení mikropočítačem. Telefonní hovor je ale stále analogový. Přesto tyto ústředny již umožňují přenos čísla volajícího, optimalizaci sestavování spojení dle analýzy čísla, dokonalou tarifikaci apod. (Bazala, 2006, s. 14)

Ústředny ***čtvrté generace*** jsou již plně digitální a založené na principu časového multiplexu TDM (Time Division Multiplex). K telefonní ústředně lze připojit jak klasické analogové, tak i digitální ISDN (Integrated Services Digital Network) koncové přístroje.

ISDN umožňují přenos hlasu i dat. Tato generace ústředen umožnila velký rozmach „podnikových“ pobočkových ústředen PBX (Private Branch Exchange). Účastník má možnost využívat telefonní služby jako například identifikaci volajícího, přesměrování, předání a konferenční hovor. Propojení mezi jednotlivými PBX mezi sebou a do VTS (Veřejné telekomunikační sítě) je realizováno pomocí přípojky ISDN PRI (Primary Rate Interface). (Bazala, 2006, s. 162)

Telefonní ústředny ***páté generace*** již nepředstavují telefonii v klasickém pojetí, ale jedná se o telefonii v prostředí počítačových sítí tzv. VoIP (Voice over Internet Protocol). Spojování hovorů je na úrovni softwarové aplikace pomocí tzv. Softswitche. Na rozdíl od všech předchozích generací ústředen již není účastnický okruh vystavěn po celou dobu hovoru, ale hovor je přenášen pomocí paketů v IP síti. Propojení mezi jednotlivými PBX mezi sebou a do VTS je realizováno buď pomocí ISDN PRI přes převodníky nebo přímo pomocí protokolu SIP (Session Initiation Protocol). (Bazala, 2006, s. 15,200)

3.5.2 Nadstavby telekomunikačních systémů

Součástí telekomunikačních systémů bývají různé aplikační nadstavby a doplňkové technologie jako například tarifikační aplikace, kontaktní centra, dispečerské nadstavby, konferenční systémy, databáze telefonního seznamu, hlasová pošta, fax server, přenosové systémy a proudové zdroje.

Tarifikační aplikace - jedná se o specializovanou aplikaci umožňující vyhodnocování a sledování telefonního provozu. Vyhodnocení provozu může být členěno dle jednotlivých uživatelů, skupin, systémů nebo napříč vybranými uživateli sítě. Detailní výpisy hovorů poskytují údaje o jednotlivých telefonních hovorech (číslo volaného a volajícího, cena, délka hovoru apod.) Součástí aplikace bývají různé statistiky, grafická vyhodnocení, případně API pro zpracování dat aplikacemi třetích stran. Tarifikační aplikace bývá nainstalována na PC připojeném k PBX pomocí sériového rozhraní, TAPI (Telephony Application Program Interface) nebo Ethernet.

Kontaktní centrum, ACD a IVR - systémy kontaktních center (dále jen KC) jsou samostatnou aplikační nadstavbou PBX nebo systému ACD, viz níže. Tyto systémy se provozují na pracovištích, kde agenti komunikují se zákazníky (vyřizují různé požadavky

a poskytují služby). Mezi tyto služby například patří poskytování informací o telefonních kontaktech, podpora tísňových volání, technická podpora, podpora zákaznických služeb (předprodejní a poprodejní péče zákazníka), telemarketing a internet banking. Systémy KC bývají provozovány na straně agenta v podobě tenkého klienta (instalované aplikace nebo webového rozhraní) komunikujícího se serverem, který zajišťuje veškeré funkce systému a datové uložiště. KC umožňují distribuovat příchozí hovory s prioritizací ve více úrovních: na úrovni vstupních skriptů například podle detekce signalizace (čísla volaného nebo čísla volajícího), na úrovni skillů (dle jazykových dovedností agenta, specializací agenta na určitou skupinu zákazníků apod.) a na úrovni samotného agenta (dle časového vytížení, dle výchozí přidělené priority apod.). Systémy KC většinou poskytují rozsáhlé reportovací nástroje (historie aktivit agentů, podrobné výpisy hovorů a reporting v reálném čase poskytující informace o stavech agentů, příchozích a probíhajících hovorech). Data reportingu v reálném čase je pak možné zobrazit na nástěnném monitoru, tzv. Wallboardu. Systémy KC by měly být nastaveny tak, že v případě nedostupnosti aplikace KC převezme směrování hovorů samotná PBX nebo ACD. ACD (Automatic Call Distribution) je funkce, implementovaná většinou přímo v PBX, která umožňuje základní automatickou distribuci příchozích hovorů. ACD je schopno rovnoměrně vytěžovat agenty přihlášené do systému. Vedoucí pracovník agentů tzv. supervizor by měl mít možnost sledovat na telefonním přístroji počet přihlášených agentů, počet účastníků ve frontě, odposlouchávat hovory na jednotlivých agentech a jednoduchým způsobem se napojit na probíhající hovor agenta. Pokud jsou všichni agenti odhlášeni, ACD by mělo mít možnost přesměrovat hovory například na jiné ACD, konkrétní telefonní číslo nebo do hlasové schránky. Mimo výše zmíněné hovorové funkcionality umožňují KC systémy poskytovat služby i přes další přístupové kanály jako například e-maily, faxy, sms zprávy a dnes velmi oblíbené internetové komunikační kanály: Web chat, Skype, Facebook a Twitter. Někdy se v kombinaci s KC, ACD nebo i samostatně používá automatický hlasový systém IVR (Interactive Voice Response). Tento systém se také někdy nazývá systém hlasového stromu, kde si volající může interaktivně vybírat zvolené informace pomocí klávesnice na svém telefonu nebo hlasových příkazů. IVR často dokáže přečíst volajícímu i dynamické informace, například položku v databázi. Častým úkolem IVR je zbavit agenty KC rutinních činností.

Telefonní seznam - aplikace telefonního seznamu obsahuje databázi účastníků telekomunikační sítě. Databáze obsahuje jméno, telefonní číslo, popřípadě název společnosti, adresu, členění na různé kategorie apod. Pomocí webového rozhraní se zobrazují a administrují data telefonního seznamu. Import kontaktů z jiných podnikových aplikací a PBX bývá zajištěn pomocí protokolu LDAP (Lightweight Directory Access Protocol).

Dispečerské systémy - jsou určeny pro komplexní řízení úkolů, plánování, rychlou komunikaci a zpracování hovorů. Tyto systémy jsou určeny především pro pracoviště krizových a finančních center, pro která je důležitá rychlost komunikace a rozhodovaní, jako například ve veřejné a letecké dopravě, u složek záchranných sborů, v energetice, v bezpečnostních službách a v bankovnictví. Koncovým telefonním zařízením bývají zpravidla PC s dotykovým monitorem a sluchátkem (náhlavní soupravou, mikrofonem, reproduktorem). Tato zařízení umožňují uložení velkého počtu cílových a linkových tlačítek, přístup do centrálního telefonního seznamu, sdílení linek mezi jednotlivými zařízeními, sestavování konferenčních hovorů s větším počtem účastníků apod.

Konferenční systémy - umožňují vytvářet skupinová volání mezi více účastníky. Existují různé typy konferencí:

- Ad-hoc – konferenční hovor se vytváří na telefonu postupným přidáváním účastníků. Počet účastníků je omezen kapacitou PBX nebo konferenčního serveru.
- Plánovaná - konferenci lze plánovat například jako schůzku v kalendáři MS Outlook. Účastníci dostanou spolu s běžnou pozvánkou i telefonní číslo, na které mají zavolat, včetně PINu pro vstup do konference.
- Videokonference - při tomto typu konference je s hlasem přenášen i video obraz. Zpravidla se jedná o zajištění videokonferenčního řešení včetně vybavení pro vybrané zasedací místnosti (například: zasedací místnosti pro management, provozní složky a krizové místnosti).

Hlasová pošta - umožňuje přesměrování telefonní linky do hlasové schránky (pro zanechání zprávy od volajícího) s funkcí indikace nové zprávy na telefonu. Měla by být také možnost nahrání uvítací hlášky uživatelem pomocí telefonu.

Fax server **-** zajišťuje virtuální faxové linky pro příjem a odesílání faxů s možností přeposílat faxové zprávy do emailové schránky. Odesílání faxů je většinou realizováno virtuální tiskárnou v PC uživatele.

IM (Instant messaging) klient **–** jedná se o SW aplikaci, kterou je možné využívat na PC a v mobilních zařízeních. Představuje centrální přístupový bod ke všem komunikačním kanálům (hlas, video, chat). Umožňuje navázat okamžitou komunikaci. Aplikace umožňuje zobrazení stavu uživatele (prezence), který je uložen v kontaktech (aktivní, nepřítomen, na schůzce, nerušit apod.). Další užitečnou funkcionalitou je sdílení plochy a aplikací (dle potřeby je možné sdílet plochu, prezentaci, softwarovou aplikaci, obrázky nebo jakákoliv jiná data).

Přenosové telekomunikační systémy - umožňují přenos příčkových vedení založených například na standardu ISDN PRI na velké vzdálenosti. Při přenosu signálu se často využívá sdružení a převod více metalických vedení PRI do jednoho datového toku na optické vedení.

Proudové zdroje - zajišťují napájení telekomunikačních systémů elektrickým proudem. Tyto zdroje dále zajišťují pomocí usměrňovačů a baterií filtraci proudových špiček a v době výpadku přívodního napájení i zálohu napájení. Telekomunikační systémy jsou obvykle napájeny buď stejnosměrným napájením z baterií (obvykle 48V) nebo střídavým napětím 230V ze střídačů.

3.6 ReDat

Specifickou částí nadstaveb telekomunikačních systémů jsou hovorové záznamové systémy, které jsou důležité pro vyhodnocování vzniklých krizových situací, mimořádných událostí, nestandardních provozních situací, požárních zásahů, bezpečnostních incidentů a v neposlední řadě řešení reklamací a stížností zákazníků. Záznamové systémy jsou využívány například v železniční a letecké dopravě, u složek záchranných sborů, v energetice, bankovnictví a kontaktních centrech. Mezi nejvýznamnější záznamové systémy v České republice patří systém ReDat od pardubické společnosti Retia, a.s.

Základem každého systému ReDat je záznamová jednotka - logger a aplikační platforma ReDat eXperience se svými moduly a periferiemi. Základní funkcí systému ReDat

je spolehlivý záznam nejrůznějších komunikačních kanálů přenášených na variabilních technologiích (různých typů zařízení od různých výrobců). ReDat loggery dokáží zaznamenat nejen hlas, ale i data nebo obraz. Každý logger je navíc vybaven nástroji pro integraci se systémy třetích stran. Díky tomu patří ReDat mezi velmi univerzální systémy svého druhu. (RETIA, a.s., ©1993-2018)

3.6.1 ReDat loggery

ReDat3 (dále jen R3) - univerzální digitální záznamové zařízení pro integrovaný záznam různých typů dat ze specifikovaných rozhraní. Systém je rozšířen o integrační nástroje pro spolupráci s nadstavbovými aplikacemi pro jednotné zpracování informace. Záznamové zařízení R3 je založeno na principu osobního počítače s monitorem, klávesnicí, myší a reproduktorovou soupravou. Provedení počítače je komerční nebo průmyslové s OS QNX, pevným diskem velké kapacity, respektive modulem řadiče RAID1 pro zrcadlení disků s výměnnými zásuvkami pro disky nebo archivační mechanikou (USB, DVD) a speciálními kartami pro digitalizaci a zpracování zvuku analogové, digitální a VoIP telefonie. Záznamová jednotka dále umožňuje pracovat i se záznamy PC obrazovek a datové komunikace.

ReDat3 P - pasivní záznamová jednotka je kompaktní logger malých rozměrů. Implementuje se do systémů a prostředí vyžadujících vysokou úroveň spolehlivosti s ohledem na minimalizaci nákladů na servis a údržbu. Zařízení je bezobslužné, neobsahuje točivé části a nevykazuje hluk.

ReDat DVI/DP - jednotka slouží k hardwarovému záznamu datové komunikace na rozhraních DVI-D, DisplayPort, USB (HID) a audio záznamu z prostorových mikrofonů. Je převážně určena na pracoviště pro řízení technologických a dopravních procesů.

ReDat KMM - jednotka, která je určena pro záznam aktivit na periferních zařízeních PC a zvuku z prostorových mikrofonů. Tato jednotka monitoruje komunikaci na USB sběrnici periferií (myš, klávesnice nebo touchscreen, souhrnně označované jako HID - Human Interface Devices) nebo mikrofonu.

ReDat VoIP Recorder (dále jen RVR) - softwarové řešení pro záznam hlasové komunikace z různých komunikačních technologií, založených na principu VoIP, včetně záznamu PC obrazovek.

RVRG (ReDat Voice Recording Gateway) - koncepčně nový produkt přinášející možnost nezávislosti záznamových rozhraní vůči serverovým a PC systémům. RVRG umožňuje provozování celého záznamového systému na serverové technologii zákazníka a podporuje i řešení plné virtualizace záznamového systému. (RETIA, a.s., ©1993-2018)

3.6.2 ReDat eXperience

ReDat eXperience (dále jen REX) je serverovou aplikační nadstavbou loggerů R3 a RVR, která umožňuje vytvoření uceleného záznamového systému pro kontaktní centra nebo centrálního archivu záznamů. REX je provozován na operačním systému Windows, využívá databázi MS SQL a webový server Apache. Veškeré činnosti uživatele jsou vykonávány pomocí webového prohlížeče. REX dále slouží k nastavení uživatelských přístupů, definici archivace záznamů a administraci ReDat loggerů. REX uchovává logy všech událostí, tzn., že každé přihlášení do systému a jakýkoliv přístup k uloženým záznamům je zapisován do aplikačního logu. REX se skládá z několika aplikací - modulů:

Catalog je základní aplikací pro práci se záznamy. Do aplikace Catalog mohou mít přístup všichni uživatelé. Oprávnění pro přístup k záznamům je dán rolí a umístěním uživatele do hierarchie.

Monitoring slouží ke sledování stavu záznamových kanálů, poboček či agentů. Umožňuje přehrávání, příposlech a rychlý výpis posledních záznamů daného zdroje. Do aplikace Monitoring mohou mít přístup všichni uživatelé. Oprávnění k jednotlivým funkcím aplikace je dáno rolí uživatele.

Audit shromažďuje záznamy o všech činnostech uživatelů, stavu systému i záznamových jednotek. Do aplikace Audit mohou přistupovat všichni uživatelé. Oprávnění k událostem určitých osob nebo systému je dáno rolí, kterou uživatel používá.

System slouží k systémové konfiguraci. Pro práci s aplikací System je nutné mít zaškrtnuté oprávnění v roli, přičemž různé sekce konfigurace mohou být individuálně povoleny

nebo zakázány. REX je dodáván s defaultním uživatelem, který nemá žádné omezení. S jeho pomocí je možné tvořit další uživatele a konfigurovat systém.

Users slouží pro konfiguraci uživatelů, pracovních skupin a rolí. Je možné zde vytvořit kompletní hierarchii zaměstnanců a přidělit jim požadovaná práva.

MySpace slouží k individuálnímu přizpůsobení pracovního prostředí systému REX. Do aplikace MySpace mají přístup všichni uživatelé bez jakéhokoli omezení.

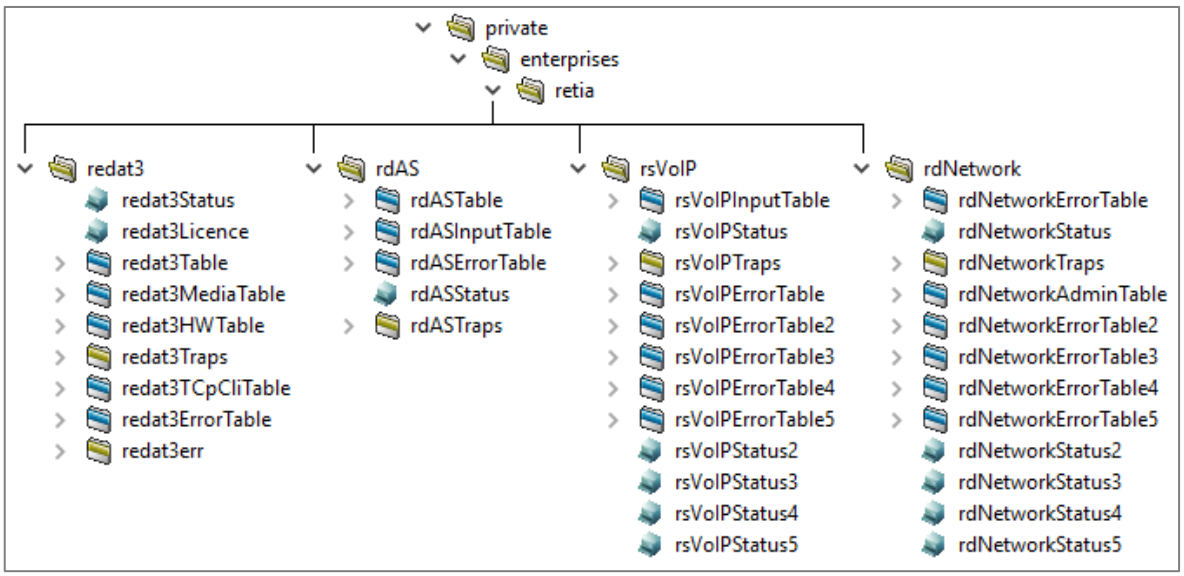
Reporting je analyticko - reportovací rozšíření určené k sestavování pokročilých analýz a k jejich následnému zobrazení, archivaci nebo k odeslání na příslušný email. Oprávnění k používání rozšíření Reporting se řídí uživatelovou rolí a jeho skupinou. Uživatelům je možné povolit konfiguraci (předpisů) reportů a následně mazání výsledků. Rozšíření Reporting se konfiguruje v aplikaci ***Rules*** na záložce Konfigurace reportů a výsledky jsou prezentovány v aplikaci ***Results*** na záložce Reporting. (RETIA, a.s., ©1993-2018)

V roce 2014 ukončila společnost Retia podporu a rozvoj aplikací ReDat Explorer (monitorování a správa loggerů) a ReDat Aplikační server (webový server pro přístup a archiv záznamů). REX zcela nahradil ReDat Aplikační server a částečně i ReDat Explorer (správa loggerů). Zbývalo tedy vyřešit centrální monitorování. To výrobce vyřešil spuštěním podpory SNMP na všech produktech ReDat a tím otevřel možnost využívat monitorovací systémy třetích stran podporujících SNMP protokol.

3.6.3 Podpora SNMP v systému ReDat

Všechny současné produkty ReDat podporují SNMP polling a SNMP Trap. Mimo základní MIB II jsou podporovány MIB Host Resources dle RFC1213 a proprietární RETIA-cz-MIB. MIB Host Resources začíná na pozici OID 1.3.6.1.2.1.25 a obsahuje základní údaje o výkonu zařízení jako například vytížení procesoru a operační paměti, systémovém čase a běžících procesech. Retia MIB začíná na pozici OID 1.3.6.1.4.1.17607 a je členěna do čtyř základních větví, viz obrázek 23. Všechny větve obsahují globální informace o celkovém stavu zařízení a tabulku s podrobným přehledem chybových stavů. První větev s názvem ***redat3*** se týká pouze záznamového zařízení R3, mimo tabulek s daty o zařízení obsahuje tato větev ještě informace o licenci. Z historických důvodů

je součástí MIB i druhá větev ***rdAS***, která se týká pouze systému ReDat Aplikační server. V této větvi se opět nachází tabulky s informacemi o činnosti systému ReDat Aplikační server, o běhu jednotlivých služeb a o stavech kanálů na připojených záznamových zařízeních. Třetí větev ***rsVoIP*** poskytuje informace o záznamovém zařízení RVR. Poslední větev ***rdNetwork*** obsahuje informace o ostatních službách, nazvaných ReDat Network. Tímto termínem společnost Retia označuje procesy běžící na operačním systému rodiny Windows (mimo RVR) a patřící do záznamového systému ReDat. Jedná se například o procesy CTI řízení a procesy systému REX.



Obrázek 23 - Retia MIB
Zdroj: Výstup aplikace SnmpB (výřez, upraveno)

Zařízení R3 využívají model Master agenta a Subagenta pomocí protokolu ***AgentX***. Master agent ***snmpd*** je součástí OS QNX a Subagent ***sqclsnmp,*** který je dodaný přímo od společnosti Retia. Konfigurace SNMP se provádí pomocí souborů ***snmpd.conf*** a ***snmp_status.cfg,*** jak si ukážeme v praktické části.

Zařízení RVR a REX využívají stejný model jako zařízení R3. Master agent ***snmp.exe*** je součástí OS Windows a knihovna ***snmpretia.dll*** plní úkol Subagenta.

4 Praktická část

Praktická část DP se zabývá zajištěním monitorování telekomunikačního systému Zadavatele, který provozuje telekomunikační systém (dále jen PBX systém) pro více než 3000 uživatelů. Zadavatel se řadí, co se týče velikosti a počtu zaměstnanců, mezi střední až velké společnosti. Telekomunikační a počítačovou síť provozuje na území o rozloze cca 1000 ha, na kterém pracuje více jak 4000 zaměstnanců. Odběratelem telekomunikačních služeb nejsou pouze zaměstnanci Zadavatele, ale i společnosti působící na jeho území.

Z důvodu zajištění anonymity a ochrany údajů nebude uveden skutečný název Zadavatele a všechny technické údaje budou v následujících kapitolách upraveny (IP adresy, doménová jména, topologie systémů apod.). Úprava těchto údajů byla zvolena s ohledem na zachování integrity sdělených informací a logické souvislosti.

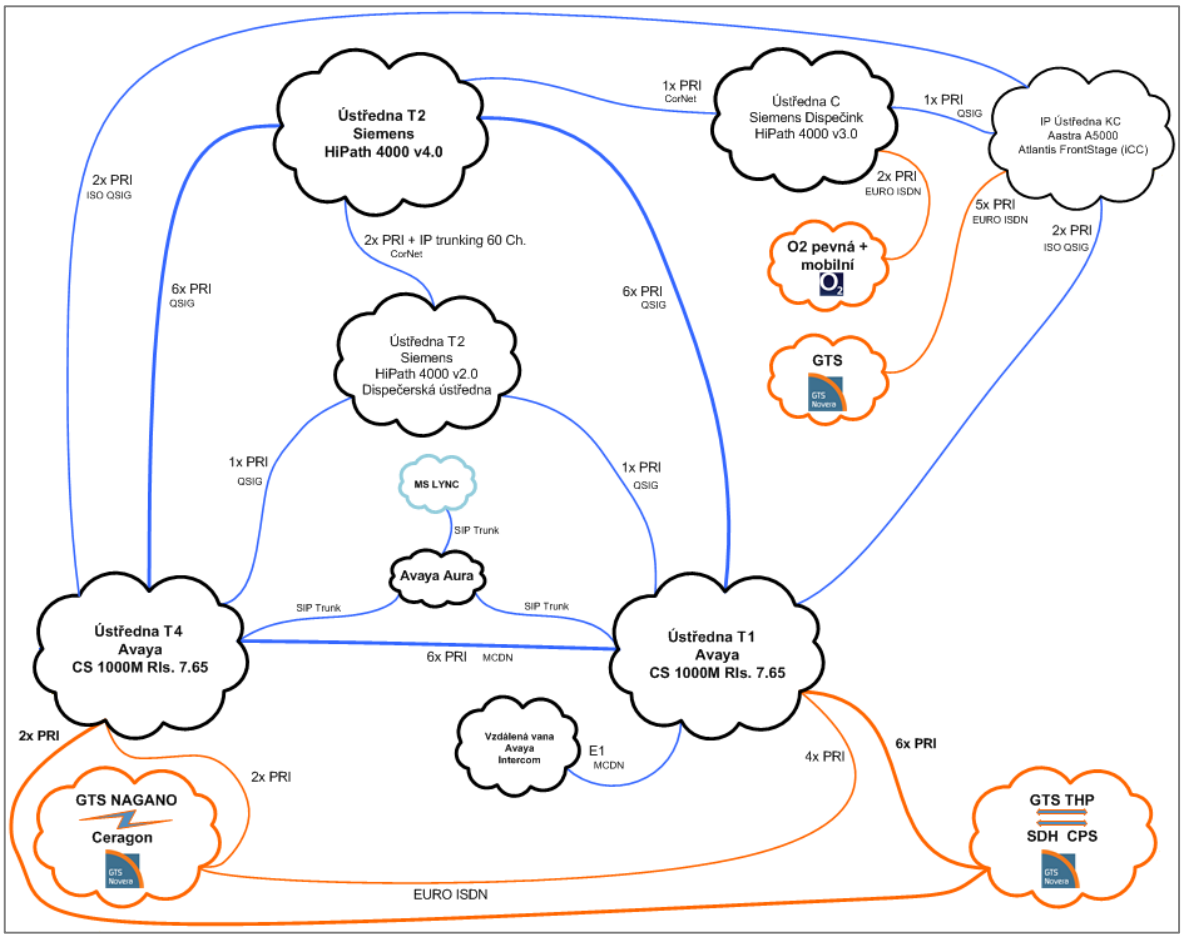
4.1 Analýza výchozího stavu systémů Zadavatele

4.1.1 Analýza a popis PBX systému

PBX systém Zadavatele je složen z několika ústředen 4. až 5. generace od výrobců Avaya (dříve Nortel) a Unify (dříve Siemens), které jsou umístěny do tří lokalit. Tento systém zajišťuje provoz veškerých hlasových služeb provozovaných na pobočkových linkách na území Zadavatele, včetně provozu dispečerského systému Siemens Trading pro provozní a bezpečnostní složky. Dále Zadavatel provozuje samostatný systém kontaktního centra Aastra, viz níže. Koncové body telefonní sítě mohou být osazovány dle požadovaného typu linky (analogové, digitální, dispečerské nebo VoIP).

Z topologie PBX systému znázorněné na obrázku 24 je patrná značná heterogenita, která byla v minulosti způsobena konsolidací více organizací v jednu společnost Zadavatele. Nejvíce to však ovlivnila snaha původního managementu zajistit konkurenční prostředí pro dodavatele jednotlivých systémů. Tato heterogenita (dva resp. tři různé systémy na úrovni výrobců PBX) má za následek několik negativních dopadů: více administrátorských rozhraní, různé typy proprietárních koncových digitálních telefonů nekompatibilních s ostatními PBX, nutnost více servisních smluv apod. Dalším negativním

aspektem je dovršení životnosti dispečerského systému Siemens Trading. Současně dochází ze strany výrobce Avaya k ukončení podpory produktu u systému CS1000M.



Obrázek 24 - PBX systém Zadavatele
Zdroj: **Autor**

Součástí PBX systému Zadavatele jsou následující nadstavby a doplňující systémy:

Kontaktní centrum Avaya AACC (Avaya Aura Contact Center) je aplikační nadstavbou funkce ACD v PBX Avaya, která slouží pro pracoviště centrálního ICT Helpdesku zajišťujícího technickou podporu. Toto pracoviště je vybaveno 6 agentskými pracovišti využívajícími VoIP a digitální linky. Systém AACC je především využíván pro vyřizování telefonických požadavků a reportingu včetně wallboardu. Přístup do AACC je realizován pomocí webové aplikace. Samotný systém AACC je provozovaný na serveru Windows, kde je spuštěn webový server, databáze Cache a samotné moduly AACC jako služby.

Kontaktní centrum Aastra je provozováno pro 70 agentských pracovišť. Tito agenti poskytují informace o telefonních kontaktech Zadavatele, zajišťují podporu zákaznických služeb, telemarketing apod. Systém KC se skládá z PBX 5. generace Aastra, IVR, aplikace

iCC (Intelligent Contact Centre) a vlastní záznamové aplikace SmartRecord. Systém IVR zajišťuje uvítací hlášky odpovídající vstupním telefonním číslům, nebo identifikaci volajícího v případě, že je telefonní číslo volajícího nalezeno ve věrnostní databázi zákazníků Zadavatele. Hovory jsou dále tříděny iCC dle jazyků, požadovaných služeb zvolených v IVR volajícím klientem a následně jsou spojeny na agenta s odpovídající znalostní skupinou. V rámci KC je zpracovávána mimo hovorů i obousměrná emailová a faxová korespondence, systém je schopen komunikovat i prostřednictvím SMS. Aplikace SmartRecord umožňuje záznam hlasu a obrazovek jednotlivých agentských pracovišť. Data se nahrávají zvlášť a poté se párují. Celý systém KC je provozovaný na několika serverech Windows, kde jsou spuštěny webové servery IIS, databáze MS SQL a samotné moduly iCC, IVR a SmartRecord jako služby.

Tarifikační aplikace Ateco umožňuje sběr a vyhodnocování tarifikačních dat z několika PBX Zadavatele. Sběr dat probíhá na několika stolních PC Ateco pomocí sériového rozhraní RS232 přímo nebo přes RS232/ IP buffer. Data jsou ukládána na lokální disky do textových souborů pomocí služby Ateco Zápis. Aplikace umožňuje nastavit lokální akustickou signalizaci při překročení určitého časového úseku od posledního přenosu dat.

Dispečerský systém Siemens Trading je provozovaný jako nadstavba na samostatné PBX Siemens HiPath odpovídající 4. generaci. Správa a konfigurace Tradingu probíhá přes lokální PC pomocí aplikace SysMan, která je spuštěna v MS DOS. Komunikace PC s PBX probíhá přes rozhraní ISDN linky. Tím je velmi omezen dálkový dohled dispečerské nadstavby. Samotná PBX je připojena do LAN sítě. Koncová dispečerská zařízení jsou velmi odolné dotykové LCD terminály s možností připojení až dvou sluchátek a externího mikrofonu. Tyto terminály jsou k PBX připojeny pomocí optického vedení. Monitorování terminálů je možné pouze přes aplikaci SysMan.

Telefonní Seznam SamWin obsahuje databázi účastníků telekomunikační sítě Zadavatele. Databáze obsahuje jméno, název společnosti, umístění, telefonní číslo a klasifikaci uveřejnění kontaktu (veřejný/neveřejný). Databáze se převážně využívá jako primární zdroj pro telefonní seznam KC Aastra. Na straně Technologie KC jsou tato data používána v agentské aplikaci. KC zajišťuje službu „telefonních spojovatelek“ (tj. služby poskytování informací o telefonních číslech Zadavatele a přepojování příchozích telefonních hovorů). Přístup k telefonnímu seznamu je zajištěn přes webové rozhraní, pomocí kterého se také

nastavují uživatelské přístupy. Server SamWin je umístěn na virtuálním prostředí VMware Zadavatele s OS Windows, kde je spuštěn webový server IIS a databáze MS SQL.

Hlasovou poštu zabezpečuje aplikace ***CallPilot*** od společnosti Avaya. Toto zařízení umožňuje aktivaci virtuálních hlasových schránek k telefonním pobočkám sítě Zadavatele. Dále tento systém zajišťuje funkce IVR, virtuální faxy a hlášky pro IVR. CallPilot je provozovaný na serveru MS Windows, kde je spuštěn webový server IIS, databáze SQL Anywhere a samotné moduly CallPilotu jako služby.

Provisioning server zajišťuje pomocí protokolu FTP (File Transfer Protocol) automatickou konfiguraci VoIP telefonů.

VocalBox announcement zajištuje přehrávání hlášek o nahrávání. Tyto hlášky jsou přehrávány volajícímu před spojením na nahrávanou linku. Aplikace VocalBox je provozována na čtyřech průmyslových PC s OS Windows XP Professional. Jednotlivé programy aplikace jsou spuštěny jako služby.

Přenosový systém TTC zajišťuje přenos příčkových vedení, založených na standardu ISDN PRI, mezi jednotlivými PBX Zadavatele. Pro velké vzdálenosti mezi PBX jsou pro přenos PRI použity optické kabely a metalicko-optické převodníky TTC včetně dohledového systému Doris, který umožňuje zasílání chybových zpráv pomocí protokolu SMTP do emailové schránky.

Proudové zdroje Benning zajišťují napájení PBX systému. Proudové zdroje se skládají z baterií 48V, usměrňovačů a střídačů. Samotné ústředny jsou napájeny stejnosměrným napětím 48V (zajišťujícím filtraci proudových špiček a v době výpadku proudu i zálohu systému). Střídače Benning 48VDC/230VAC slouží převážně jako záložní zdroj pro technologii nahrávání hovorů, hlasovou poštu a tarifikaci. Ve dvou lokalitách je provoz usměrňovačů monitorován lokálním signalizačním tablem, které akusticky signalizuje odpojení usměrňovačů od přívodního napájení nebo defekt některého z usměrňovačů.

Nově patří do správy administrátorů PBX systému nástěnné ***IP hodiny*** zobrazující aktuální čas v prostorách Zadavatele. Zatím se jedná pouze o 4ks hodin od výrobců Mobatime a RGB Technology. Mobatime podporují SNMP protokol a RGB Technology umožňují pouze webový přístup pro správu a kontrolu času.

4.1.1.1 Monitorování PBX systému

Všechny hlavní části ústředen (hovorové a řídící karty/servery) jsou připojené do sítě LAN přímo nebo pomocí modemu přes terminálový PC. Pomocí příkazového řádku nebo grafických nadstaveb mohou administrátoři PBX systému (oddělení STS – Správa Telekomunikačních Systémů) provádět pravidelné kontroly a správu systému. Některé systémy umožňují zasílání chybových zpráv pomocí protokolu SMTP do sdílené emailové schránky administrátorů STS. Administrátoři STS v rámci služeb H24 také provádějí fyzickou kontrolu PBX systému. Na základě požadavku Zadavatele snížit náklady dojde ke snížení počtu administrátorů STS ze šesti na čtyři. V souvislosti s tím dojde ke zrušení nočních služeb, služeb o víkendech a svátcích. Nově bude zavedeno držení pohotovostí mimo pracovní dobu.

4.1.2 Analýza a popis systému ReDat

Zadavatel provozuje záznamový systém ReDat složený ze čtyř loggerů R3, dvou Aplikačních serverů a aplikace Explorer. Loggery R3 zajišťují hlasový záznam na jednotlivých lokalitách s širokou paletou typů záznamu (analog, digitál, VoIP, PCM a radiové relace). Všechny loggery jsou v průmyslovém provedení s duálními napájecími zdroji a diskovým polem RAID1. Hlavní Aplikační server slouží pro přehrávání záznamů ze všech loggerů pomocí webového rozhraní, k nastavení uživatelských přístupů a k definici archivace záznamů. Druhý Aplikační server pak slouží v jedné lokalitě pro přímé přehrávání záznamů přes dispečerský pult pomocí API.

4.1.2.1 Monitorování systému ReDat

Pro vzdálenou správu jednotlivých loggerů a monitorování celého systému ReDat slouží ReDat Explorer. Explorer je instalovaný na OS Windows XP a od uvedení do provozu vykazuje aplikace problémy. K aplikaci bylo nutné nasadit skripty na pravidelný restart služby Exploreru a automatické dělení log souborů. Monitorování ale poskytuje velmi podrobné informace o chodu jednotlivých systémů. Chybové stavy pak dle závažnosti chyby zasílá pomocí protokolu SMTP do sdílené emailové schránky administrátorů STS.

4.1.3 Analýza a popis monitorovacích systémů Zadavatele

Zadavatel provozuje v rámci celého ICT několik monitorovacích systémů. Mezi největší patří MS SCOM a dva systémy Nagios.

MS SCOM (Microsoft System Center Operations Manager) je komerční monitorovací systém od společnosti Microsoft a je určený především pro produkty od této společnosti. SCOM využívá a spravuje oddělení SWA (Správa Windows Aplikací).

Oba systémy Nagios spravuje oddělení SSI (Správa Síťové Infrastruktury). První Nagios je určen právě pro toto oddělení pro monitorování aktivních a pasivních prvků počítačové sítě. Tento systém monitoruje cca 190 hostitelů a 1500 služeb. Druhý systém Nagios využívají oddělení SPA (Správa Provozních Aplikací) a SUS (Správa Unix/Linux Systémů). Tento systém monitoruje cca 220 hostitelů a 1800 služeb. Každý systém Nagios je provozován v jiném datovém centru (VMware) Zadavatele a navzájem si sledují dostupnost monitorovacích služeb a údaje o výkonu. Oba systémy jsou využívány ve variantě DIY Nagios Core včetně rozložení zátěže pomocí Mod-Gearman (každý systém dva workery) a vizualizační nadstavby NagVis. Každý server je provozován na OS Linux Debian a HW: 2x CPU Intel Xeon X5670, RAM 2GB a HDD 10GB. Konfigurace systémů je prováděna přímo editací textových konfiguračních souborů. Pro základní administraci a vizualizaci dat se využívá výchozí webové rozhraní. Systémy Nagios dále umožňují pomocí Event Handleru generování incidentů v aplikaci HP OpenView Service Desk (dále jen OVSD) na pracoviště centrálního ICT Helpdesku. Na tomto pracovišti se také zobrazují NagVis mapy z prvního Nagiosu.

Počítačová síť Zadavatele je, co se týká velikosti rozlohy, na úrovni sítí CAN. Interní síť je oddělena od Internetu a sítí jiných společností pomocí DMZ (Demilitarized Zone) s dvojitým firewallem. Tato síť je pak interně rozdělena na serverovou a uživatelskou síť pomocí firewallu. Tyto sítě jsou pak dále logicky členěny do virtuálních podsítí VLAN podle zaměření koncových zařízení a systémů. Komunikace systémů Nagios s monitorovanými hostiteli tedy probíhá buď přes firewall nebo pomocí pravidel mezi VLAN.

4.2 Navrhované řešení

Vzhledem k tomu, že již není podporována monitorovací aplikace ReDat Explorer a dojde ke zrušením nočních služeb administrátorů STS, je nutné nasadit jednotné monitorování celého PBX systému. Z důvodu neustálého snižování nákladů Zadavatele bylo zvoleno využití stávajícího systému Nagios (již využívaného odděleními SPA a SUS). Tento systém je pro svou modulárnost a flexibilitu velmi vhodným řešením pro heterogenní PBX systém. V Nagiosu budou definovány nové:

- kontakty a skupina kontaktů pro administrátory STS,
- konfigurační soubory hostitelů a služeb a jejich skupiny,
- šablony definic hostitelů a služeb,
- definice příkazů pro nové typy kontrol,
- šablony hostitelů a služeb pro různé typy generování incidentů v aplikaci OVSD pro skupinu STS.

Dále bude podle typu koncového zařízení (hostitele) vybrán nejvhodnější typ Nagios pluginu a typ Agenta na straně hostitele (SNMP, NRPE a NSClient++). V případě, že nebude nalezen vhodný dostupný plugin, bude vytvořen plugin nový. V NagVisu budou vytvořeny nové mapy PBX systému, které budou přístupné jak pracovišti STS, tak i ICT Helpdesku.

Pro zajištění kompletního monitorování PBX systému by bylo nutné z důvodu zastaralé technologie a jeho heterogenity provést upgrade jednotlivých systémů. V některých případech to již není z důvodu ukončení podpory výrobcem možné. Dnešní management již prosazuje obnovu PBX na homogenní systém tak, aby byla zajištěna centralizovaná správa, monitorování a byla minimalizována potřeba několika proprietárních koncových přístrojů od různých výrobců. Bohužel se tato obnova systému, z důvodu očekávaných vysokých finančních nákladů, několikrát posunula. Proto není možné nyní zajistit přímé monitorování všech systémů, jejich služeb a dispečerských koncových zařízení, které nemají konektivitu na LAN nebo v současnosti nepodporují protokol SNMP.

Specifikace monitorování PBX systému

Vzhledem k výše zmíněným skutečnostem, dokud nebude PBX systém kompletně obnoven, bude nasazeno monitorování alespoň na část telekomunikačního systému, který je dostupný z LAN sítě. Na některých částech bude sledována minimálně dostupnost

hostitelů samotným pingem. Na téměř všech nadstavbách telekomunikačního systému budou pomocí systému Nagios sledovány služby a prováděny následující kontroly:

- zaplnění disků,
- vytížení procesoru,
- vytížení operační paměti,
- systémového času,
- velikosti DB, datových souborů,
- posledního zápisu na disk, do DB nebo do datového souboru,
- logovacích souborů OS,
- běhu služeb (dle aplikace),
- platnosti https certifikátu (web servery),
- počtu přihlášených uživatelů v OS (servery).

Nově budou nasazena teplotní čidla HWg-STE Plus do technologických místností s telefonními ústřednami. Mimo teplotu budou tato čidla sledovat funkčnost proudových zdrojů pomocí signalizačních kontaktů.

4.2.1 Specifikace monitorování systému ReDat

Bude provedena náhrada Aplikačního serveru ReDat za REX a upgrade loggerů R3. Zároveň bude zprovozněn na celém systému ReDat SNMP protokol. ReDat Explorer bude zcela nahrazen Nagiosem.

Monitorování loggerů R3 bude obsahovat následující kontroly:

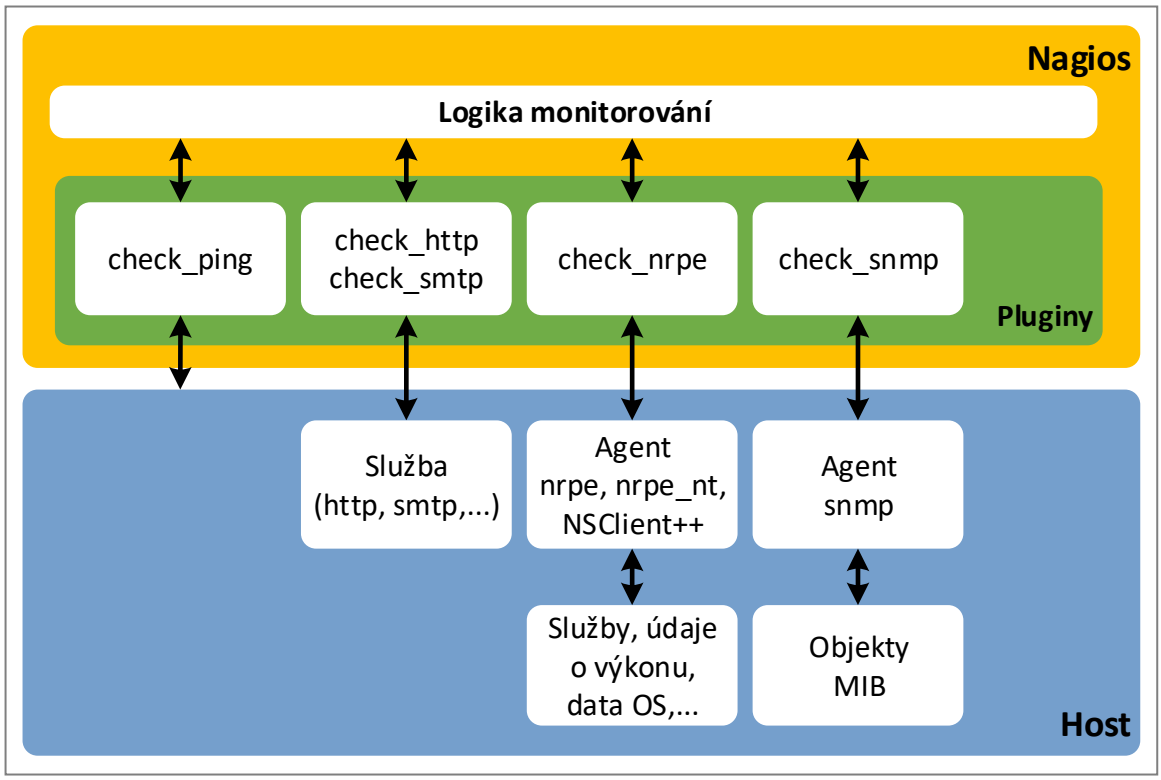
- odpojení kanálu,
- zatížení kanálu (podtečení/přetečení),
- tabulky chyb,
- zaplnění disku,
- posledního zápisu na disk (RAID),
- vytížení procesoru,
- vytížení operační paměti,
- systémového času,
- napájení (funkčnosti duálního zdroje, detekce vstupního napájení),
- napětí (kontrola výstupu zdrojů),
- RAID (funkčnosti HDD1/HDD2),
- konektivity na NTP server,
- funkčnosti ventilátorů.

Monitorování ReDat eXperience bude obsahovat tyto kontroly:

- tabulky chyb,
- zaplnění disků,
- posledního zápisu na disk (DB),
- vytížení procesoru,
- vytížení operační paměti,
- systémového času,
- existence souborů záloh DB,
- velikosti DB,
- běhu služeb (Apache, SQL, exp_*),
- platnosti https certifikátu,
- počtu přihlášených uživatelů v OS.

4.3 Konfigurace Nagiosu

Na základě specifikace požadavků na monitorování hostitelů a služeb PBX a ReDat systému je navrženo několik typů kontrol, viz obrázek 25.



Obrázek 25 - Typy kontrol PBX a ReDat systému
Zdroj: **Autor**

První typ kontrol je určen pro zjištění dostupnosti hostitele. Používá se k tomu plugin „check_ping“ ze základního balíčku pluginů a již z názvu je zřejmé, že plugin využívá příkazy ping. Jedná se o úplně nejzákladnější typ kontrol a definuje se v objektu Host. Všechny ostatní typy kontrol se definují v objektu Service.

U druhého typu kontrol, stejně jako u prvního typu, není zapotřebí na straně hostitele žádná instalace monitorovacího Agenta. Sama služba odpovídá na dotaz Nagios pluginu. V tomto případě se jedná o tzv. veřejně dostupné služby jako například http/https, ftp, smtp a ssh. Všechny potřebné pluginy jsou již obsaženy v základním balíčku pluginů.

Třetí typ kontrol využívá na straně Nagiosu velmi univerzální plugin „check_nrpe“. Na straně hostitele je spuštěn monitorovací Agent (dle typu OS nebo typu kontrol), který pomocí vlastních modulů, externích programů, pluginů nebo skriptů získává potřebná data pro určitý typ kontroly.

Poslední typ kontrol je v principu velmi podobný předchozímu typu, jen je celý založen na protokolu SNMP. Všechny předchozí typy kontrol využívají na straně Nagiosu pluginy, které jsou již obsaženy v základním balíčku pluginů. Do tohoto balíčku patří i plugin „check_snmp“. Tento plugin je určen pro základní SNMP kontroly, které umožňují vyhodnocovat OID data se základními datovými typy. Není tedy vhodný především pro vyhodnocování dat obsahujících datum a čas, vyhodnocování dat v SNMP tabulkách a vyhodnocování více OID hodnot s různými datovými typy. Pro tyto specifické typy kontrol je nutné použít pluginy od komunity vývojářů nebo vytvořit pluginy vlastní.

Než přistoupíme k podrobnému popisu typu kontrol (definic příkazů) a objektů (Host a Service), musíme nadefinovat kontakty a skupinu kontaktů pro administrátory STS.

4.3.1 Definice objektů Contact a Contactgroup

Definice objektu „contact“ slouží k identifikaci osoby, která by měla být v případě problému kontaktována (zasláno oznámení). V první třech direktivách jsou definovány jmenné a kontaktní údaje.

```
define contact{
        contact_name                    stetkai@ZADAVATEL.CZ
        alias                           Ivan Stetka
        email                           ivan.stetka@zadavatel.cz
        host_notification_period        24x7
        service_notification_period     24x7
        host_notification_options       d,u,r,f
        service_notification_options    w,u,c,r,f
        host_notification_commands      host-notify-by-email
        service_notification_commands   notify-by-email
        }
```

Následují direktivy definované zvlášť pro hostitele a službu:

notification_period - obsahuje název objektu „timeperiod“, ve kterém jsou definována časová období, kdy má být osoba kontaktována (24 hodin denně, 7 dní v týdnu).

notification_options - určuje, při jakém stavu má být oznámení zasláno kontaktu.

host: d (down) – vypnutý
u (unreachable) – nedostupný
r (recovery) – návrat do normálu (UP)
f (flapping) – mění-li host často stavy

service: w (warning) – stav varování
u (unknown) – neznámý stav
c (critical) – kritický stav
r (recovery) – návrat do normálu (OK)
f (flapping) – mění-li služba často stavy

notification_commands - obsahuje název příkazu, který se má spustit při aktivaci oznámení.

Definice příkazu „host-notify-by-email“ obsahuje příkaz pro zaslání emailové zprávy, včetně toho, co má zpráva obsahovat. Pro doplnění obsahu zprávy jsou použita makra a pro odeslání zprávy konzolová Unix aplikace „mailx“, viz níže.

```
define command{
        command_name                    host-notify-by-email
        command_line                    /usr/bin/printf "%b" "*Nagios*\n\nNotification Type: $NOTIFICATIONTYPE$\nHost:
//$HOSTNAME$\nState:       $HOSTSTATE$\nAddress:       $HOSTADDRESS$\nInfo:       $HOSTOUTPUT$\n\nDate/Time:
//$LONGDATETIME$\n" | /usr/bin/mailx -s "Host $HOSTSTATE$ alert for $HOSTNAME$!" $CONTACTEMAIL$
        }
```

V definici objektu „contactgroup“, pomocí direktivy „members“, zařadíme jednotlivé kontakty do skupiny s názvem „pbx_users“.

```
define contactgroup{
        contactgroup_name       pbx_users
        alias                   PBX contact group
        members                 stetkai@ZADAVATEL.CZ,kolega1@ZADAVATEL.CZ,kolega2@ZADAVATEL.CZ
        }
```

4.3.2 Definice objektů Host, Hostgroup a kontroly hostitelů

Při používání šablon nám do definice objektu „host“ postačí zadat pouze čtyři základní direktivy: název použité šablony, název hostitele, název skupiny, do které chceme hostitele zařadit, a IP adresu hostitele.

```
define host{
        use                     pbx_host
        host_name               Redat_eXperience
        hostgroups              PBX_ReDat
        address                 192.168.0.200
        }
```

Pokud u každého hostitele definujeme název skupiny, tak v samotné definici objektu „hostgroup“ stačí zadat pouze název a popis skupiny.

```
define hostgroup {
        hostgroup_name          PBX_ReDat
        alias                   PBX ReDat
        }
```

Všechny společné vlastnosti hostitelů nadefinujeme v šabloně, odkud budou hostitelé tyto vlastnosti dědit.

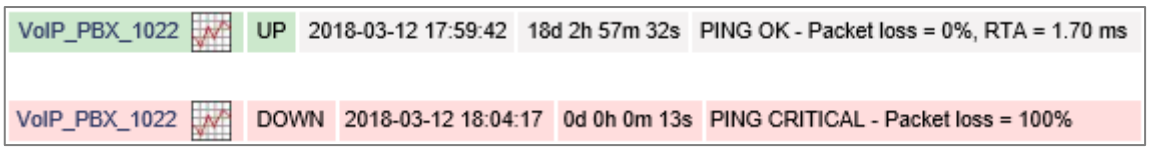
```
define host {
        name                    pbx_host        # název šablony
        register                0               # registrace objektu zakázána, viz ods. 3.4.4
        check_command           check_alive     # název příkazu pro kontrolu hostitele
        notifications_enabled   1               # oznámení povoleno
        flap_detection_enabled  1               # detekce měnícího se stavu povolena
        process_perf_data       1               # zpracování údajů o výkonu povoleno
        check_interval          5               # kontrola hostitele každých 5 minut při Hard state
        retry_interval          1               # kontrola hostitele každou minutu při Soft state
        max_check_attempts      3               # počet opakování kontrol při Soft state než dojde k Hard state
        notification_interval   0               # opakování oznámení vypnuto (oznámení zasláno jednou)
        notification_period     24x7            # timeperiod 24x7 pro samotné spuštění oznámení
        notification_options    d,u,r,f         # stavy hostitele spouštějící oznámení
        contact_groups          pbx_users       # skupina kontaktů pro oznámení
        event_handler_enabled   0               # zpracování událostí zakázáno
        }
```

Definice příkazu „check_alive“ obsahuje příkaz pro kontrolu hostitelů pomocí pluginu „check_ping“. IP adresa hostitele se doplňuje pomocí makra a prahové hodnoty warning a critical jsou pevně zadány. U každé prahové hodnoty představuje první údaj průměrnou

dobu odezvy na ping a druhý údaj (za čárkou) představuje počet ztracených paketů v procentech.

```
define command {
        command_name            check_alive
        command_line            /.../plugins/check_ping  -H $HOSTADDRESS$ -w 300.0,60% -c 500.0,100%
        }
```

Na následujícím obrázku jsou zobrazeny stavy hostitele UP a DOWN ve webovém rozhraní Nagiosu.



Obrázek 26 - Nagios Host (UP/DOWN)
Zdroj: Nagios web rozhraní (výřez, upraveno)

Pro generování incidentů v aplikaci OVSD pro skupinu STS nadefinujeme další šablonu.

```
define host {
        name                    pbx_host_OVSD_major
        use                     pbx_host
        alias                   #STS
        event_handler_enabled   1
        event_handler           OVSD_incident!YES!HKP!major
        }
```

Event handler nám zajistí spuštění příkazu „OVSD_incident“. Ke spuštění samotného Event handleru dojde, tak jako u oznámení, až při přechodu chybového stavu do ***Hard state***.

```
define command{
        command_name            OVSD_incident
        command_line            /.../OVSD_incident_script "$ARG1$" "$ARG2$" "$SERVICEDESC$"
"$SERVICESTATE$" //"$SERVICESTATETYPE$" "$SERVICEOUTPUT$" "$HOSTNAME$" "$HOSTADDRESS$" "$HOSTALIAS$"
//"$HOSTSTATE$" "$HOSTSTATETYPE$" "$HOSTOUTPUT$" "$LONGDATETIME$" "$ARG3$" "$HOSTDOWNTIME$"
//"$SERVICEDOWNTIME$" "$HOSTEVENTID$" "$SERVICEEVENTID$"
        }
```

Definice příkazu „OVSD_incident“ obsahuje příkaz pro vygenerování incidentu v aplikaci OVSD pomocí skriptu „OVSD_incident_script“. Samotný skript byl vytvořen administrátorem SSI a není součástí této DP. Ve skriptu jsou pro doplnění obsahu incidentu použita různá makra hostitele a služeb. Pro nastavení parametrů incidentu jsou požita makra „ARG1“ až „ARG3“, která se zadají do šablony hostitele v „event._handler“ následovně:

commnad_name!$ARG1$!$ARG2!$ARG3$

OVSD_incident!<Closeability>!<Type>!<Severity>

- Closeability určuje, zda se má uzavřít incident v OVSD při návratu do stavu UP/OK.
  YES - incident se automaticky uzavře
  NO - incident zůstává otevřen až do zásahu uživatele
- Type určuje, jakým způsobem je získávána KP (Konfigurační Položka) do OVSD.
  KP v OVSD představuje skupinu parametrů, se kterými se vytvoří incident, jako například skupinu zařízení, skupinu kontaktů a procesy.
  HKP - využívá se pro Host, KP se použije v definici Host, „alias“ za znakem #
  SKP - využívá se pro Service, ale KP se načte v definici Host, „alias“ za znakem #
- Severity určuje dopad incidentu v OVSD, viz tabulka 4.

Tabulka 4 - Dopad incidentů

Nagios definice	OVSD definice	Dopad
Normal	Minimální	30 dní
Warning	Nízký	10 dní
Minor	Střední	5 dní
Major	Vysoký	24 h
Critical	Kritický	1 h

Zdroj: Autor dle podkladů administrátorů SSI

Můžeme tedy vytvářet šablony hostitelů a služeb s různými argumenty příkazu OVSD_incident pro generování incidentů s různým dopadem, KP a typem uzavření.

4.3.3 Definice objektů Service, Servicegroup a kontrol veřejných služeb

Obdobně jako u objektu Host definujeme šablony pro objekt Service.

```
define service {
        name                    pbx_service     # název šablony
        register                0               # registrace objektu zakázána, viz ods. 3.4.4
        active_checks_enabled   1               # aktivní monitorování povoleno
        notifications_enabled   1               # oznámení povoleno
        flap_detection_enabled  1               # detekce měnícího se stavu povolena
        process_perf_data       1               # zpracování údajů o výkonu povoleno
        check_interval          5               # kontrola služby každých 5 minut při Hard state
        retry_interval          1               # kontrola služby každou minutu při Soft state
        max_check_attempts      3               # počet opakování kontrol při Soft state než dojde k Hard state
        notification_interval   0               # opakování oznámení vypnuto (oznámení zasláno jednou)
        notification_period     24x7            # timeperiod 24x7 pro samotné spuštění oznámení
        notification_options    w,u,c,r,f       # stavy služby spouštějící oznámení
        contact_groups          pbx_users       # skupina kontaktů pro oznámení
        event_handler_enabled   0               # zpracování událostí zakázáno
        }
```

Také pro služby definujeme další šablonu pro generování incidentů v aplikaci OVSD pro skupinu STS.

```
define service {
        name                    pbx_service_OVSD_minor
        use                     pbx_service
        alias                   #STS
        event_handler_enabled   1
        event_handler           OVSD_incident!YES!SKP!minor
        }
```

Nyní můžeme přistoupit k definici kontroly například webové služby https, konkrétně doby vypršení SSL (Secure Sockets Layer) certifikátu. Tuto kontrolu není třeba provádět každých 5 minut, proto pomocí direktivy „normal_check_interval“ změníme interval kontrol na jeden den (1440 minut).

```
define service{
        use                     pbx_service_OVSD_minor
        hosts                   Redat_eXperience
        servicegroups           PBX_ReDat
        service_description     check_https_certificate
        check_command           check_certificate_ssl!10,3!experience.zadavatel.cz
        normal_check_interval   1440
        }
```

Definice příkazu „check_certificate_ssl“ obsahuje příkaz pro kontrolu doby vypršení certifikátu pomocí pluginu „check_http“.

```
define command{
        command_name            check_certificate_ssl
        command_line            /…/plugins/check_http -S -C $ARG1$ -I $ARG2$
        }
```

Pomocí přepínače „-S“ zapneme SSL (port 443). Pro nastavení dalších argumentů pomocí přepínačů „-C“ a „-I“ jsou použita makra „ARG1“ a „ARG2“. Prvním argumentem nastavíme prahové hodnoty warning a critical oddělené čárku. Uvedené číslo „10,3“ představuje vyhlášení stavu warning při dosažení doby 10 dnů před vypršením certifikátu a vyhlášení stavu critical při dosažení doby 3 dnů před vypršením certifikátu. Druhým argumentem definujeme adresu webového serveru, na kterém chceme vypršení doby certifikátu kontrolovat.

4.3.4 Definice kontrol služeb pomocí NRPE_NT

Pomocí univerzálního pluginu „check_nrpe“ a monitorovacího Agenta na straně hostitele lze spouštět různé typy kontrol. Monitorovací Agent je dostupný jak pro OS Unix/Linux, tak i pro OS Windows. My se v této kapitole budeme zabývat Agentem ***nrpe_nt*** určeným pro OS Windows. Agent je dostupný na stránkách komunity vývojářů pluginů (Nagios Enterprises, ©2009-2018c). Pokud stáhneme zip soubor s označením „bin“, tak lze soubory pouze dekomprimovat a nakopírovat do nové složky. Obdobně lze získat i základní balíček pluginů „nrpe_nt_plugins.zip“ a další pluginy, které na straně hostitele provádějí samotné kontroly. V tabulce 5 je uveden seznam použitých pluginů.

Tabulka 5 - Pluginy nrpe_nt

Plugin	Typ kontroly
cpuload_nrpe_nt.exe	vytížení procesoru
memload_nrpe_nt.exe	vytížení operační paměti
diskspace_nrpe_nt.exe	zaplnění disku
service_nrpe_nt.exe	služeb
check_winprocess.exe	procesů
check_winevent.exe	protokolu událostí
check_winfile.exe	počtu, velikosti a stáří souborů a složek
check_user_count.bat	počtu přihlášených uživatelů (zobrazí i jméno)
check_windows_time.bat	systémového času hostitele

Zdroj: Autor

V konfiguračním souboru „nrpe.cfg“ nastavíme základní parametry Agenta.

```
server_port=5666                                # port pro komunikaci s Nagiosem
allowed_hosts=192.198.10.20, 192.198.10.22,... # povolení přístupu pouze Nagiosu a Workerům
dont_blame_nrpe=1                               # povolení možnosti zadávat argumenty na straně Nagiosu
```

Ve stejném souboru definujeme jednotlivé typy kontrol.

```
command[nt_check_cpu]=cpuload_nrpe_nt.exe 90 95
command[nt_check_memory]=memload_nrpe_nt.exe 90 95
command[nt_check_disk_c]=diskspace_nrpe_nt.exe c: 80 90
command[nt_check_services_NetWorker_res]=service_nrpe_nt.exe "NetWorker Remote Exec Service"
command[nt_check_process_wtels]=check_winprocess.exel --filter "imagename eq WTELS.EXE" --compare ne --critical 2
command[nt_check_event]=check_winevent --type error --window "2 hours" --critical 0
command[nt_check_age]=check_winfile.exe -t "D:\ATECO\DATA\DATA.TXT" --filter "age lt -2 hours" --critical 0
command[nt_check_users]=check_user_count.bat 1 2
command[nt_check_time]=check_windows_time.bat $ARG1$ $ARG2$ $ARG3$
```

Než přistoupíme k definici objektů na straně Nagiosu, tak je potřebné nainstalovat program Agenta „nrpe_nt.exe“ jako službu pomocí přepínače „-i“. A poté přes správce služeb v OS tuto službu spustit.

```
C:\NRPE\nrpe_nt.exe –i
```

Pokud vkládáme argumenty příkazů přímo do souboru „nrpe.cfg“, tak například definice služby pro kontrolu zaplnění disku C na straně Nagiosu bude vypadat následovně.

```
define service{
        use                     pbx_host_OVSD_major
        hosts                   Ateco_T1
        servicegroups           PBX_Ateco
        service_description     check_disk_usage_C
        check_command           check_nrpe_1arg!nt_check_disk_c
        }
```

Definice objektu příkazu pak bude obsahovat pouze makro „$HOSTADDRESS$“ a „$ARG1$“. Do makra „$ARG1$“ se pomocí přepínače „-c“ vkládá název příkazu, který se má spouštět na straně Agenta.

```
define command{
        command_name            check_nrpe_1arg
        command_line            /…/plugins/check_nrpe -H $HOSTADDRESS$ -c $ARG1$
        }
```

Pokud potřebujeme vkládat argumenty příkazů na straně Nagiosu, jako v případě kontroly systémového času hostitele, bude definice služby vypadat následovně.

```
define service{
        use                     pbx_host_OVSD_major
        hosts                   Ateco_T1
        servicegroups           PBX_Ateco
        service_description     check_time
        check_command           nt_check_time!nt_check_time!10!30
        }
```

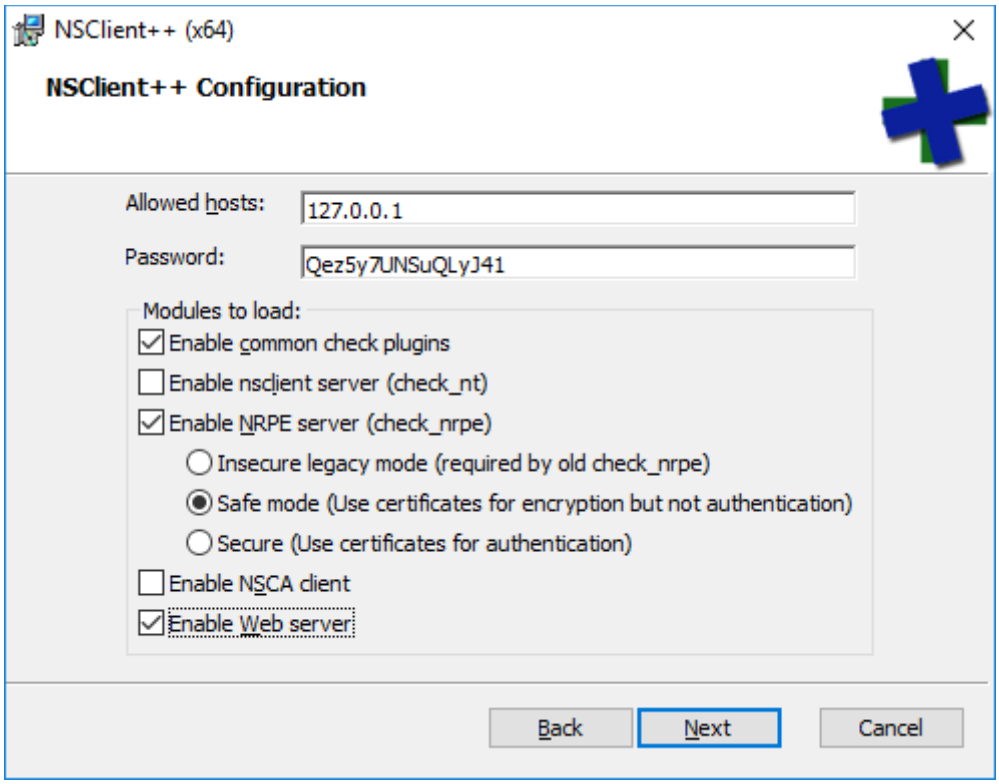
Definice objektu příkazu pak bude navíc obsahovat přepínač „-a“ sloužící pro přenos argumentů do příkazu na straně agenta. První argument obsahuje IP adresu NTP serveru a další dva argumenty pomocí maker $ARG2$ a $ARG3$ slouží pro určení prahových hodnot warning a critical.

```
define command{
        command_name    nt_check_time
        command_line    /…/plugins/check_nrpe -H $HOSTADDRESS$ -c $ARG1$ -a 192.168.10.10 '$ARG2$' '$ARG3$'
        }
```

Výhoda Agenta ***nrpe_nt*** je především v jeho snadné konfiguraci a instalaci. Nevýhodou je, že některé typy pluginů (cpuload_nrpe_nt.exe a memload_nrpe_nt.exe) nefungují na 64bitových verzích OS Windows. Pro tyto případy je vhodnější Agent ***NSClient++***.

4.3.5 Definice kontrol služeb pomocí NSClient++

Agent ***NSClient++*** byl dříve určený pouze pro OS Windows. Poslední verze ale navíc podporuje některé distribuce OS Linux a další platformy. Podrobnější informace lze nalézt na webových stránkách projektu autora ***Michaela Medina*** (Medin, ©2015). Instalace na OS Windows se provádí pomocí instalačního balíčku. Pokud zvolíme „Typical“ instalaci, viz obrázek 27, tak můžeme navíc zvolit instalaci webového rozhraní, pomocí kterého lze všechny potřebné funkce nastavovat.



Obrázek 27 - Instalace NSClient++
Zdroj: **Instalační balíček NSClient++**

Součástí instalace jsou základní moduly, které na straně hostitele provádějí samotné kontroly. Lze používat i externí pluginy a skripty, viz poslední dva skripty v tabulce 6.

Tabulka 6 - Moduly NSClient++

Modul/externí skript	Typ kontroly
check_cpu	vytížení procesoru
check_memory	vytížení operační paměti
check_drivesize	zaplnění disku
check_service	služeb
check_process	procesů
check_eventlog	protokolu událostí
check_files	počtu, velikosti a stáří souborů a složek
check_tasksched	stavu naplánovaných úloh
check_user_count.bat	počtu přihlášených uživatelů (zobrazí i jméno)
check_windows_time.bat	systémového času hostitele

Zdroj: Autor

Doposud jsme prováděli veškerá nastavení pomocí textových konfiguračních souborů, tak i zde máme k dispozici konfigurační soubor ***nsclient.ini.*** Dle subjektivního názoru autora této DP je nastavení pomocí konfiguračního souboru rychlejší a přehlednější než pomocí webového rozhraní. Po instalaci můžeme přímo v konfiguračním souboru některé základní parametry změnit a dokonfigurovat.

```
port =12489                                    # port pro komunikaci s Nagiosem (nebo 5666)
allowed hosts =192.198.10.20, 192.198.10.22,... # povolení přístupu pouze Nagiosu a Workerům
allow arguments = 1                            # povolení možnosti zadávat argumenty na straně Nagiosu
allow nasty characters = 1                     # povolení zadávat speciální znaky v argumentech (např. |`&><'"\[]{})
CheckDisk = 1                                  # povolení kontrol drivesize a files
CheckSystem = 1                                # povolení kontrol cpu, memory, service a process
CheckEventLog = 1                              # povolení kontrol eventlog
CheckTaskSched = 1                             # povolení kontrol tasksched
```

Ve stejném souboru definujeme jednotlivé typy kontrol. A zde trochu nastává problém. K uvedeným modulům v tabulce 6 existují starší verze, jako například k modulu „check_cpu“ je dostupný i starší modul „checkCpu“. Starší moduly jsou zpětně kompatibilní a mají jinou syntaxi. Syntaxe definic kontrol umožňuje širokou škálu modifikací a filtrů. V dokumentaci je uvedeno jen pár základních příkladů. Proto není vždy jednoduché na první pokus správně definovat jednotlivé kontroly. V následující kódu je uveden jednoduchý příklad čtyř způsobů, jak dojít ke stejnému výsledku.

```
alias_check_disk_C_1 = CheckDriveSize MinWarn=10% MinCrit=5% Drive=c:
alias_check_disk_C_2 = CheckDriveSize MaxWarn=90% MaxCrit=95% Drive=c:
alias_check_disk_C_3 = check_drivesize "warn=free<10%" "crit=free<5%" drive=c:
alias_check_disk_C_4 = check_drivesize "warn=used>90%" "crit=used>95%" drive=c:
```

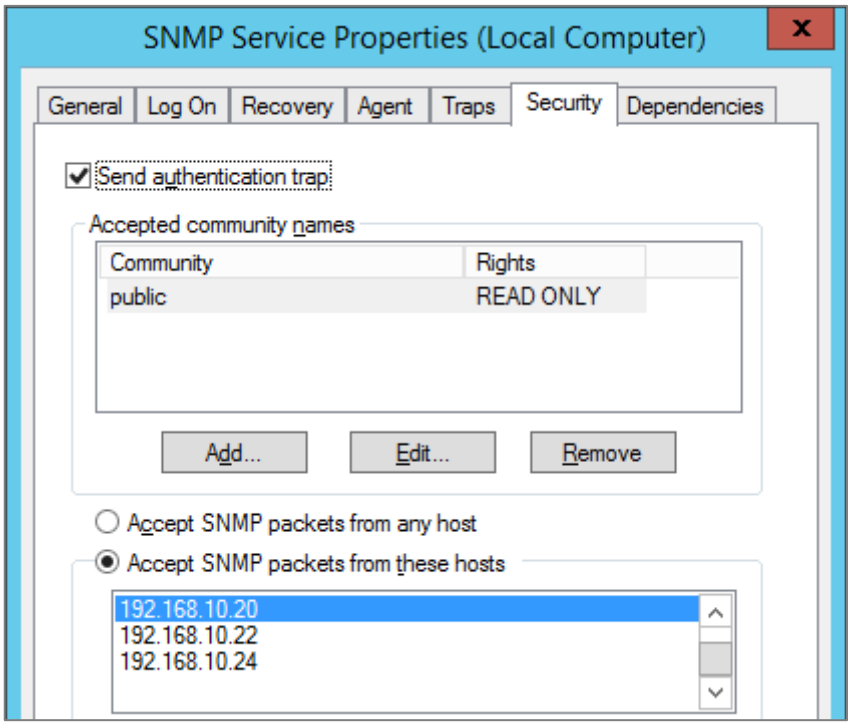
Definice objektů služeb a příkazů na straně Nagiosu je totožná jako v případě Agenta ***nrpe_nt***. Pouze pokud používáme oba Agenty na jednom hostiteli, je potřeba definovat objekty příkazů s nastaveným portem pro Agenta ***NSClient++***.

```
define command {
        command_name    check_nrpe_nscp
        command_line    /.../plugins/check_nrpe -p 12489 -H $HOSTADDRESS$ -c $ARG1$
        }
```

4.3.6 Definice kontrol služeb pomocí SNMP

Poslední typ kontrol je v principu velmi podobný předchozím dvěma typům. Zatímco u předchozích typů je na straně Agenta zpracována převážná část vyhodnocení kontrol, tak v případě SNMP je provedeno vyhodnocení na straně SNMP pluginu v Nagiosu. SNMP Agent pouze vrací potřebná data ke zpracování na základě dotazů pluginu. Proto není snadné nalézt univerzální plugin pro všechny typy kontrol. Nejdříve, než přistoupíme k popisu jednotlivých pluginů a kontrol, ukážeme si základní nastavení SNMP Agentů v loggeru R3 (OS QNX) a serveru REX (OS Windows).

V OS Windows se SNMP Agent nastavuje přes správce služeb. Vzhledem k tomu, že SNMP používáme pouze pro monitorování, nastavíme komunitu pouze pro čtení.



Obrázek 28 - SNMP OS Windows
Zdroj: **OS Windows (výřez)**

Obdobně nastavíme SNMP Agenta v loggeru R3 pomocí konfiguračního souboru ***snmpd.conf***. Zde je možné nastavit i časové parametry protokolu agentX.

```
# rocommunity: a SNMPv1/SNMPv2c read-only access community name
# arguments:  community [default|hostname|network/bits] [oid]
rocommunity  public 192.168.10.20
rocommunity  public 192.168.10.22
rocommunity  public 192.168.10.24

# agentx protocol support
master agentx
agentxTimeout                30
agentxRetries                5
```

Plugin „check_snmp“ ze základního balíčku pluginů je určen pro typy SNMP kontrol, které umožňují vyhodnocovat jednoduché OID proměnné se základními datovými typy.

```
define command{
        command_name    check_snmp_Cowclu
        command_line    /…/plugins/check_snmp -H '$HOSTADDRESS$' -C '$ARG1$' -o '$ARG2$' -w '$ARG3$'
-c '$ARG4$' -l '$ARG5$' -u '$ARG6$'
        }
```

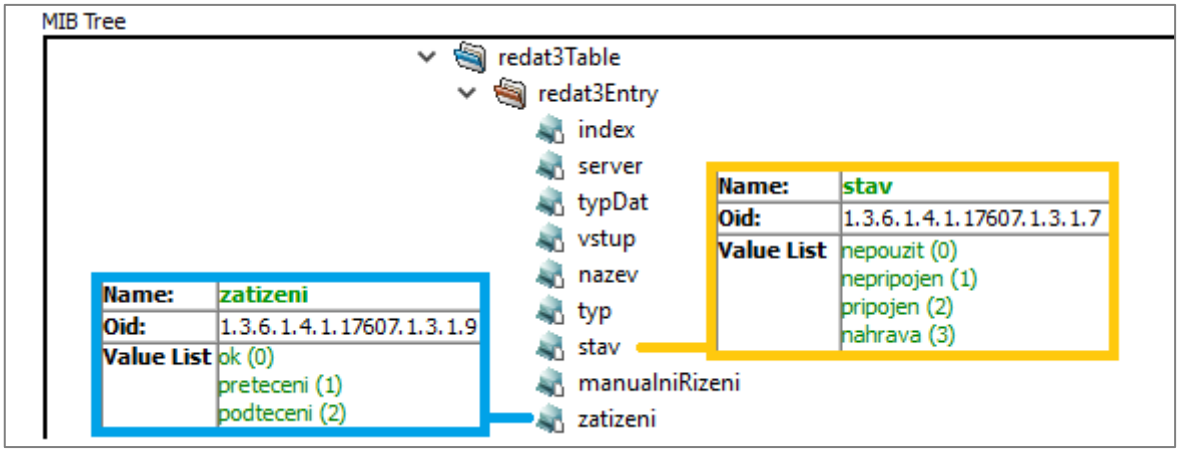
Velmi užitečná je možnost doplnit výsledná data popisem a jednotkami pomocí argumentů přepínačů „-l“ a „-u“.

```
define service{
        use                             pbx_service_OVSD_major
        hosts                           Redat3_HZS
        servicegroups                   PBX_ReDat
        service_description             check_snmp_CPU
        check_command                   check_snmp_Cowclu!public!1.3.6.1.2.1.25.3.3.1.2.1!80!95!CPU load:!%
        }

define service{
        use                             pbx_service_OVSD_major
        hosts                           Redat3_HZS
        servicegroups                   PBX_ReDat
        service_description             check_snmp_Memory
        check_command                   check_snmp_Cowclu!public!1.3.6.1.2.1.25.2.3.1.6.2!1500000!1900000!
Memory load:!bytes allocated from total size of 2 GB
    }
```

Obtížnější bylo najít plugin pro vyčítání dat ze SNMP tabulky, která navíc obsahuje dynamicky se měnící záznamy. Byl nalezen jediný vyhovující plugin „check_snmp-table.pl“ od autora ***Williama Leibzona*** (Leibzon, 2014). Tento plugin umožňuje párovat dva sloupce tabulky a vyhodnocovat data z jednoho sloupce na základě prahových hodnot. V případě

loggeru R3 nás zajímá tabulka „redat3Table“, která obsahuje informace o záznamových kanálech (stav a zatížení) a informace o HW (RAID, teplota, ventilátory a napájení), viz obrázek 29.



Obrázek 29 - SNMP tabulka redat3Table
Zdroj: Výstup aplikace SnmpB (výřez, upraveno)

V definicí příkazů nastavíme makra pro prahové hodnoty a přepínač „-a“, který je určen pro identifikaci názvů záznamových kanálů obsažených v pátém sloupci. OID tohoto sloupce se zadává do argumentu přepínače „-N“. V přepínači „-D“ je pak uvedeno OID sloupce se sledovanými daty. První definice příkazu „check_ReDat3Table_input“ slouží pro kontrolu stavu záznamového kanálu a druhá definice příkazu „check_ReDat3Table_load“ slouží pro kontrolu zatížení záznamového kanálu.

```
define command {
        command_name    check_ReDat3Table_input
        command_line    /.../plugins/check_snmp_table.pl -H $HOSTADDRESS$ -a $ARG1$ -w $ARG2$ -c $ARG3$
-N .1.3.6.1.4.1.17607.1.3.1.5 -D .1.3.6.1.4.1.17607.1.3.1.7 -f -R -C public
        }

define command {
        command_name    check_ReDat3Table_load
        command_line    /.../plugins/check_snmp_table.pl -H $HOSTADDRESS$ -a $ARG1$ -w $ARG2$ -c $ARG3$
-N .1.3.6.1.4.1.17607.1.3.1.5 -D .1.3.6.1.4.1.17607.1.3.1.9 -f -R -C public
        }
```

V následujícím kódu jsou uvedeny příklady definic služeb kontroly stavu záznamového kanálu, kontroly podtečení záznamového kanálu a kontroly přetečení záznamového kanálu.

```
define service{
        use                     pbx_service_OVSD_major
        hosts                   Redat3_T1
        servicegroups           PBX_Recording
        service_description     UDR3_4
        check_command           check_ReDat3Table_input_name! '3302h,4490h,8000h'!'=0,=0,=0'!'=1,=1,=1'
        }

define service{
        use                     pbx_service_OVSD_major
        hosts                   Redat3_T1
        servicegroups           PBX_Recording
        service_description     UDR3_4_load_min
        check_command           check_ReDat3Table_load_name!'3302h,4490h,8000h'!'=^,=^,=^'!'=2,=2,=2'
        }

define service{
        use                     pbx_service_OVSD_minor
        hosts                   Redat3_T1
        servicegroups           PBX_Recording
        service_description     UDR3_4_load_max
        check_command           check_ReDat3Table_load_name!'3302h,4490h,8000h'!'=^,=^,=^'!'=1,=1,=1'
        }
```

Vzhledem k tomu, že plugin „check_snmp“ není vhodný pro vyhodnocování dat obsahujících datum a čas, bylo nutné pro tyto případy vytvořit pluginy vlastní. První plugin check_snmp_lastwritetime.pl je určený pro kontrolu posledního zápisu na disk zařízení ReDat R3 a REX. Plugin je podrobně popsaný v kapitole 4.4.2. V následujících kódech jsou uvedeny příklady definic příkazu a služby pro kontrolu R3.

```
define command {
        command_name    check_lastwrite_ReDat3
        command_line    /…/plugins/check_snmp_lastwritetime.pl  -E R3 -H $HOSTADDRESS$ -C public -z UTC1
-w $ARG1$ -c $ARG2$
        }
```

```
define service{
        use                     pbx_service_OVSD_major
        hosts                   Redat3_T1
        servicegroups           PBX_ReDat
        service_description     check_lastwrite_ReDat3
        check_command           check_lastwrite_ReDat3!60!900
        }
```

Druhý vlastní plugin check_snmp_time.pl vychází z předchozího pluginu a je určený pro kontrolu systémového času hostitele pomocí SNMP dotazu na jednoduchou proměnou OID, obsahující data v časovém formátu. Plugin je popsaný v kapitole 4.4.3.

```
define command {
        command_name    check_snmp_time_Cozl
        command_line    /.../plugins/check_snmp_time.pl -H $HOSTADDRESS$ -C $ARG1$ -o $ARG2$ -z $ARG3$
-l $ARG4$
        }
```

```
define service{
        use                     pbx_service_OVSD_major
        hosts                   Redat3_T1
        servicegroups           PBX_ReDat
        service_description     check_snmp_time
        check_command           check_snmp_time_Cozl!public!1.3.6.1.2.1.25.1.2.0!UTC1!'ReDat3 T1'
        }
```

Na obrázku 30 je zobrazena část služeb, které pro svou kontrolu využívají všechny zmíněné SNMP pluginy.

Service Status Details For Host 'Redat3_T1'

Service	Status	Status Information
check_snmp_time	OK	OK: ReDat3 T1 time is the same as System time.
check_snmp_Memory	OK	SNMP OK - Memory load: 877022 bytes allocated from total size of 2 GB
check_snmp_CPU	OK	SNMP OK - CPU load: 24 %
check_lastwrite_ReDat3	OK	OK: last write to disc: Mon Mar 26 12:07:29 2018
check_error_table	OK	OK: No errors, R3 status is OK.
UDR3_4_load_min	OK	OK - 3302h is 0, 4490h is 0, 8000h is 0
UDR3_4_load_max	OK	OK - 3302h is 0, 4490h is 0, 8000h is 0
UDR3_4	OK	OK - 3302h is 2, 4490h is 2, 8000h is 2
HW_Temperature	OK	OK - HW Case 9:01 is 2
HW_RAID	OK	OK - RAID 3:01 is 2
HW_PW_supl	OK	OK - HW Case 9:04 is 2
HW_NTP	OK	OK - NTP 2:01 is 2
HW_FUNs	OK	OK - HW Case 9:02 is 2
HW_ETH0_Span_HLP	OK	OK - Network 10:03 is 2
HW_ETH0_Servis	OK	OK - Network 10:01 is 2

Obrázek 30 - Nagios service status Redat3_T1
Zdroj: Nagios web rozhraní (výřez, upraveno)

Třetí vlastní plugin „check_snmp_errortable.pl“ je určený pro výpis a kontrolu SNMP tabulky obsahující chyby zařízení ReDat R3 a REX a pro výpis OID Status proměnné celého zařízení. Tento plugin umožňuje výpis čtyř sloupců najednou. Plugin je velmi podrobně popsaný v kapitole 4.4.1.

```
define command {
        command_name    check_ERR_table_ReDat3
        command_line    /usr/local/bin/check_snmp_errortable.pl  -E R3 -H $HOSTADDRESS$ -C public
}
```

```
define service{
        use                     pbx_service_OVSD_major
        hosts                   Redat_eXperience_HZS
        servicegroups           PBX_ReDat
        service_description     check_error_table
        check_command           check_ERR_table_REX
        }
```

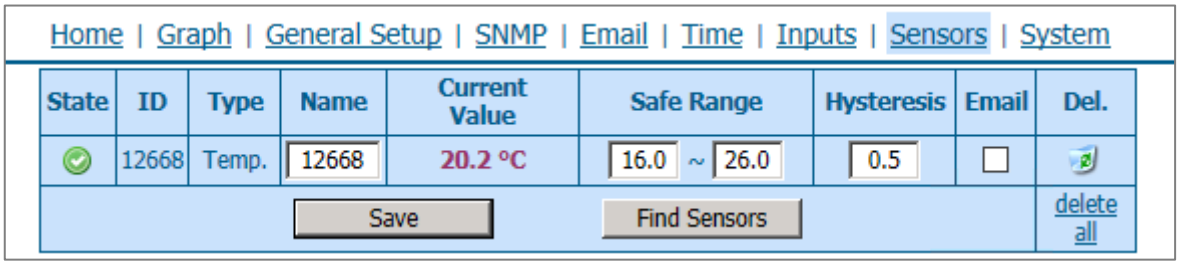
Server REX obsahuje různé typy kontrol pomocí SNMP pluginů a pomocí Agenta NSClient++, viz obrázek 31.

Service Status Details For Host 'Redat_eXperience'

Service	Status	Status Information
CPU	OK	OK: CPU load is ok.
Memory	OK	OK: physical: Total: 8GB - Used: 5.449GB (68%) - Free: 2.551GB (31%)
check_age_backup_DB	OK	FILE OK - 4 files out of 5 to consider
check_disk_usage_C	OK	Used: 43060 MB (84%) Free: 7787 MB (15%)
check_disk_usage_D	OK	Used: 94513 MB (46%) Free: 110156 MB (53%)
check_error_table	OK	OK: No errors, REX status is OK.
check_https	OK	GET / HTTP/1.1
check_https_certificate	OK	OK - Certificate 'wshexperience.cah.cz' will expire on 08/07/2020 11:47.
check_lastwrite_REX	OK	OK: last write to disc 15/03 14:45:16
check_services_Apache	OK	Apache2.4 OK
check_services_NetWorker_res	OK	NetWorker Remote Exec Service OK
check_services_SNMP	OK	SNMP Service OK
check_services_SQLServer	OK	SQL Server (MSSQLSERVER) OK
check_services_SQLWriter	OK	SQL Server VSS Writer OK
check_services_exp_arch	OK	exp_arch OK
check_services_exp_compound	OK	exp_compound OK
check_services_exp_lic	OK	exp_lic OK
check_services_exp_play	OK	exp_play OK
check_services_exp_repl	OK	exp_repl OK
check_services_exp_reporting	OK	exp_reporting OK
check_size_DB_3GB	OK	FILE OK - 0 files out of 1 to consider
check_time	OK	OK: Time is +00.2481843s from 10.50.9.10
check_users	OK	OK - Number of RDP sessions = 0, Users =

Obrázek 31 - Nagios service status Redat_eXperience
Zdroj: Nagios web rozhraní (výřez, upraveno)

Pro teplotní čidla použijeme plugin „check_hwg-ste.pl“, který je dostupný ze stránek výrobce (HW group, 2010) a je určený pro čidla řady Poseidon a STE. Prahové hodnoty se nastavují pomocí webového rozhraní, viz obrázek 32.



Obrázek 32 - HWg-STE Plus
Zdroj: HWg web rozhraní (výřez, upraveno)

Definice objektů služeb a příkazů je pak jednoduchá.

```
define command{
        command_name    check_hwg_input
        command_line    /.../plugins/check_hwg-ste.pl -H '$HOSTADDRESS$' -C '$ARG1$' -I '$ARG2$'
        }
define command{
        command_name    check_hwg_temp
        command_line    /.../plugins/check_hwg-ste.pl -H '$HOSTADDRESS$' -C '$ARG1$' -S '$ARG2$'
        }
```

```
define service{
        use                     pbx_service_OVSD_major
        hosts                   HWg_PBX_T1
        servicegroups           PBX_Inputs
        service_description     Input1_PBX_T1_Proudove_zdroje
        check_command           check_hwg_temp!public!1         #příkaz!komunita!číslo kontaktu
        }
define service{
        use                     pbx_service_OVSD_major
        hosts                   HWg_PBX_T1
        servicegroups           PBX_Temperature
        service_description     Temperature_PBX_T1
        check_command           check_hwg_temp!public!12668     #příkaz!komunita!číslo teplotního čidla
        }
```

4.4 Vlastní pluginy

4.4.1 Plugin pro kontrolu tabulky chyb ReDat

Plugin „check_snmp_errortable.pl“, viz příloha 1, je určen pro výpis a kontrolu SNMP tabulky obsahující chyby zařízení ReDat R3, REX a RVR. K obsahu chybových tabulek má vztah jednoduchá OID proměnná Status. Pokud tabulka chyb obsahuje záznamy, Status proměnná nabyde hodnoty (1 až 3) podle nejvyšší závažnosti ze všech chyb. Když zařízení ReDat neobsahuje chyby, Status proměnná má hodnotu 0 (OK) a tabulka je prázdná. Bohužel

SNMP vyhodnotí prázdnou tabulku jako neexistující. Z tohoto důvodu poslední verze pluginu kontroluje jak chybovou tabulku, tak i Status proměnnou. Příklad výpisu chybové tabulky loggeru R3 se čtyřmi záznamy pomocí programu snmpwalk je na obrázku 33.

```
iso.3.6.1.4.1.17607.1.8.1.1.1 = INTEGER: 1
iso.3.6.1.4.1.17607.1.8.1.1.2 = INTEGER: 2
iso.3.6.1.4.1.17607.1.8.1.2.1 = STRING: "2018/03/22_08:54:04"
iso.3.6.1.4.1.17607.1.8.1.2.2 = STRING: "2018/03/22_08:54:08"
iso.3.6.1.4.1.17607.1.8.1.3.1 = STRING: "UDR"
iso.3.6.1.4.1.17607.1.8.1.3.2 = STRING: "UDR"
iso.3.6.1.4.1.17607.1.8.1.4.1 = INTEGER: 3
iso.3.6.1.4.1.17607.1.8.1.4.2 = INTEGER: 2
iso.3.6.1.4.1.17607.1.8.1.5.1 = STRING: "UDR3 5:04 1022"
iso.3.6.1.4.1.17607.1.8.1.5.2 = STRING: "UDR3 5:04 1022"
iso.3.6.1.4.1.17607.1.8.1.6.1 = INTEGER: 1
iso.3.6.1.4.1.17607.1.8.1.6.2 = INTEGER: 1
iso.3.6.1.4.1.17607.1.8.1.7.1 = INTEGER: 105301
iso.3.6.1.4.1.17607.1.8.1.7.2 = INTEGER: 105302
```

Obrázek 33 - Net-SNMP: Error tabulka ReDat3
Zdroj: Výstup z příkazového řádku Nagios pomocí programu snmpwalk (Net-SNMP)

Z tohoto výpisu není úplně zřejmé, jaké chyby zařízení R3 obsahuje. Tento typ výpisu je zaslán i pluginu při SNMP dotazu na tabulku. Proto je nutné, aby plugin obsahoval číselníky s textovými informacemi nebo některé informace načítal přímo z MIB souboru, jako v případě programu SnmpB, viz obrázek 34. V našem pluginu jsou použity oba způsoby doplnění informací. Plugin musí také obsahovat mechanismus, který převede výstup ze SNMP dotazu (řádky o dvou údajích – OID a hodnota proměnné) do tabulkové formy.

erIndex	erDate	erSource	erSeverity	erText	erFlag	erId
1	2018/03/22_08:54:04	UDR	fatalError(3)	UDR3 5:04 1022	odeslatAPonechat(1)	e105301-UDR-vstup-bez-napeti(105301)
2	2018/03/22_08:54:08	UDR	error(2)	UDR3 5:04 1022	odeslatAPonechat(1)	e105302-UDR-nevytizeny-vstup(105302)

Obrázek 34 - SnmpB: Error tabulka ReDat3
Zdroj: Výstup aplikace SnmpB (Table View, výřez)

Pro naši potřebu jsou v pluginu sledovány čtyři sloupce: Source, Severity, Text a Id. Výsledný plugin je využit v definici služby „check_error_table“. Ve webovém rozhraní Nagiosu v náhledu služby „check_error_table“ se zobrazuje pouze první řádek výstupního textu. Tento řádek obsahuje informace o Status proměnné, počtu chyb a zkrácený výpis tabulky (sloupec Text a Id), viz obrázek 35.

Service	Status	Status Information
check_error_table	WARNING	WARNING: Status is 'FatalError' and the table contains 2 errors: UDR3 5:04 1022 - UDR-vstup-bez-napeti, UDR3 5:04 1022 - UDR-nevytizeny-vstup

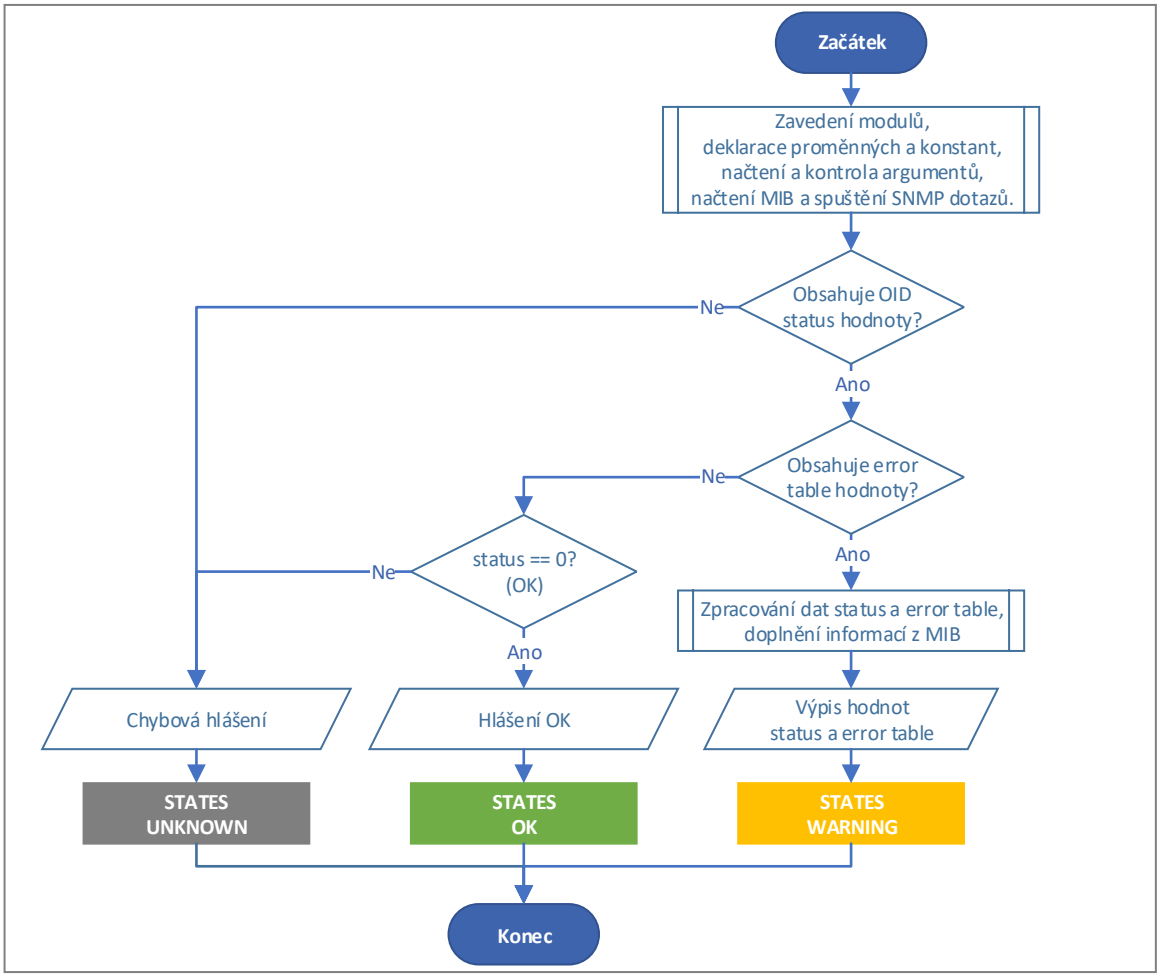
Obrázek 35 - Service status check_error_table
Zdroj: Nagios web rozhraní (výřez, upraveno)

Ve webovém rozhraní Nagiosu v zobrazení „Service Information“ jsou pak navíc zobrazeny všechny požadované údaje z tabulky a Perf data (počet chyb), viz obrázek 36.

Current Status:	**WARNING** (for 0d 0h 29m 57s)
Status Information:	WARNING: Status is 'FatalError' and the table contains 2 errors: UDR3 5:04 1022 - UDR-vstup-bez-napeti, UDR3 5:04 1022 - UDR-nevytizeny-vstup
	105301, FatalError, UDR: UDR3 5:04 1022, Info: UDR-vstup-bez-napeti
	105302, Error, UDR: UDR3 5:04 1022, Info: UDR-nevytizeny-vstup
Performance Data:	Errors=2

Obrázek 36 - Service State Information check_error_table
Zdroj: Nagios web rozhraní (výřez)

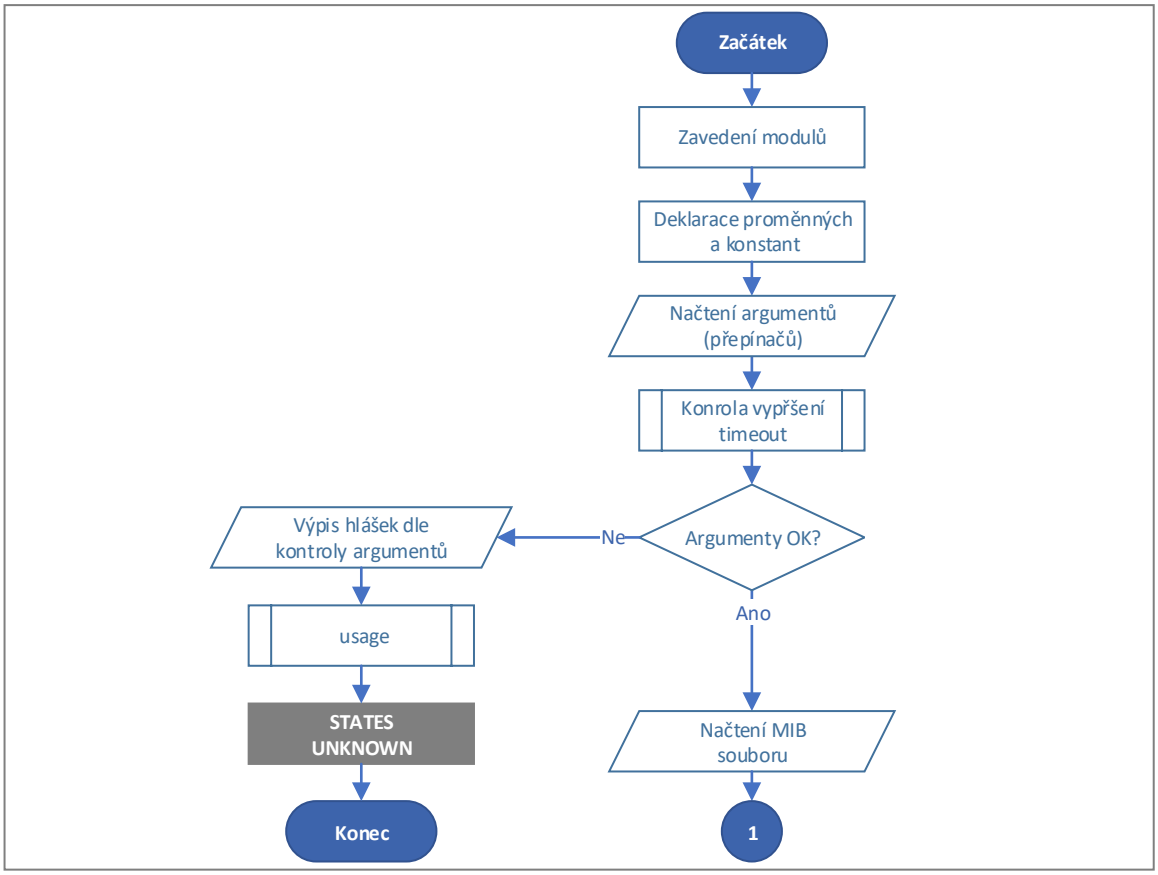
Na obrázku 37 je zjednodušeně znázorněn celý algoritmus pluginu. Jelikož výpis záznamů z tabulky chyb nám bude sloužit jen jako informativní, byl nejvyšší stav návratové hodnoty pluginu zvolen na WARNING. Jednotlivé části pluginu si podrobně rozebereme v následujících kapitolách.



Obrázek 37 - Vývojový diagram algoritmu pluginu check_snmp_errortable
Zdroj: Autor

4.4.1.1 První část pluginu check_snmp_errortable

V první části pluginu jsou zavedeny moduly, jsou deklarovány proměnné a konstanty a jsou očekávány a načteny argumenty pomocí přepínačů. Následovně je spuštěna funkce „vypršení času“ pro neočekávané chyby pluginu. Dále jsou zkontrolovány přípustné hodnoty argumentů a jejich existence. Poté následuje načtení MIB souboru, viz obrázek 38.



Obrázek 38 - Vývojový diagram 1. část pluginu check_snmp_errortable
Zdroj: Autor

Nyní si projdeme jednotlivé řádky kódu v programu Perl. První řádek využívá tzv. konvenci „#!“, kde se zadá cesta interpretu Perl a volitelný parametr, v našem případě „-w“, který nám spustí plugin v režimu „warning“, viz kapitola 3.4.5. Za samotný znak „#“ pak můžeme do kódu psát vlastní komentáře. Na řádku 10 zapínáme striktní režim pro běh pluginu. Následuje zavedení modulu pro možnost spouštění SNMP dotazů a zavedení modulu „Getopt::Long“ pro možnost zadávání argumentů pomocí přepínačů. Přepínače je možné zadávat jak v krátké formě například pro zadání adresy hostitele „-H 192.168.0.100“, tak i v rozšířené formě „--host 192.168.0.100“.

```
1  #!/usr/bin/perl -w
2
3  # Version:      1.4
4  # Date:         Mar 12, 2018
5  # Author:       ivan.stetka@live.com
6  # Summary:      This is a nagios plugin for check SNMP error table of Retia
7  #               recording equipment ReDat3, eXperience and VoIP Recorder
8  #               (RETIA-cz-MIB.mib)
9
10 use strict;
11 use Net::SNMP;
12 use Getopt::Long qw(:config no_ignore_case bundling);
```

Řádkem 14 začínáme deklarovat konstanty (proměnné obsahující po celou dobu běhu pluginu konstantní hodnoty). Pro lepší přehled v kódu jsou konstanty pojmenovány velkými písmeny a ostatní proměnné malými písmeny. První tři konstanty představují čísla sloupců tabulky chyb. Sloupec „Id“ bude z důvodu jiné struktury tabulky R3 a REX (resp. RVR) deklarován jako proměnná, viz níže. Na řádku 18 začínáme deklarovat hash „STATES“, který slouží pro stavovou návratovou hodnotu Nagiosu. Hash „SEVERITYTEXT“ slouží jako číselník pro číselnou hodnotu sloupce Severity chybové tabulky a Status proměnné.

```
14 my $ESOURCE = 3;
15 my $ESEVERITY = 4;
16 my $ETEXT = 5;
17
18 my %STATES = (
19     OK              => 0,
20     WARNING         => 1,
21     CRITICAL        => 2,
22     UNKNOWN         => 3
23 );
24
25 my %SEVERITYTEXT = (
26     0 => 'OK',
27     1 => 'Warning',
28     2 => 'Error',
29     3 => 'FatalError'
30 );
```

Vzhledem k tomu, že je plugin určen pro specifické vyčítání hodnot zařízení ReDat, jsou OID Status proměnné a chybové tabulky doplněny z hashů podle zvoleného typu zařízení, viz řádek 32-42.

```
32 my %ESTATUS = (
33     "R3"   => '1.3.6.1.4.1.17607.1.1.0',
34     "REX"  => '1.3.6.1.4.1.17607.5.2.0',
35     "RVR"  => '1.3.6.1.4.1.17607.3.2.0'
36 );
37
38 my %ETABLE = (
39     "R3"   => '1.3.6.1.4.1.17607.1.8.1',
40     "REX"  => '1.3.6.1.4.1.17607.5.1.1',
41     "RVR"  => '1.3.6.1.4.1.17607.3.4.1'
42 );
```

Následuje deklarace proměnných a přepínačů. Všechny názvy proměnných jsou zvoleny s ohledem na snazší orientaci a čitelnost v kódu.

```
44 my $eId = undef;
45 my $host = undef;
46 my $community = undef;
47 my $equipment = undef;
48 my $protocolV = 1;
49 my $mibFilePath = '/usr/local/bin/RETIA-cz-MIB.mib';
50 my $timeout = 15;
51 my $version = 0;
52 my $help = 0;
53 my $rowCounter = 0;
54 my $commaCounter = 0;
55 my $plural = '';
56 my %eTextValues = ();
57 my %eIdValues = ();
58 my %eSeverityValues = ();
59 my %eSourceValues = ();
60 my %longText = ();
61
62 GetOptions(
63     'H|host:s'          => \$host,
64     'C|community:s'     => \$community,
65     'E|equipment:s'     => \$equipment,
66     'P|protocolV:n'     => \$protocolV,
67     'm|mibFile:s'       => \$mibFilePath,
68     't|timeout:n'       => \$timeout,
69     'V|version'         => \$version,
70     'h|help'            => \$help
71 );
```

V případě, že bude zadán přepínač „-h“, plugin vypíše text podprogramu „help“ (nápovědu) a ukončí se s návratovou hodnotou „0“ (OK). To samé platí i v případě zadání přepínače „-v“ pro výpis verzí pluginu. Podprogramy „help“ a „version“ začínají řádkem 299, viz příloha 1.

```
73 if ($help) {
74     help();
75     exit($STATES{OK});
76 }
77
78 if ($version) {
79     version();
80     exit($STATES{OK});
81 }
```

Pro případ, že by chybně zadané hodnoty argumentů nebo jiné chyby v průběhu běhu celého pluginu způsobily nečinnost pluginu, slouží kontrola vypršení času (timeout). Výchozí nastavení timeout systému Nagios je 10 s. Pokud tedy z nějakého důvodu potřebujeme, aby plugin byl ukončen dříve, než ho ukončí Nagios, můžeme zadat nižší hodnotu proměnné timeout v samotném pluginu pomocí přepínače „-t“.

```
84 $SIG{ALRM} = sub {
85     print "Something is wrong, exceeded timeout parameter.\n";
86     usage();
87     exit($STATES{UNKNOWN});
88 };
89 alarm $timeout;
```

Řádkem 92 začíná kontrola zadaných argumentů v přepínačích. Nejdříve je kontrolováno, zda byly argumenty zadány. Při chybějících hodnotách jsou při všech následujících kontrolách vypsány chybové hlášky, je vypsán text podprogramu „usage“ (řádek 292-296, viz příloha 1) a plugin ukončen návratovou hodnotou 3 (UNKNOWN).

```
91 # input control
92 if (!defined $host || !defined $community || !defined $equipment) {
93     print "Missing required parameter (host, community, equipment).\n";
94     usage();
95     exit($STATES{UNKNOWN});
96 }
```

Pomocí hashe „ETABLE“ je ověřena hodnota proměnné „equipment“.

```
98 if (!defined ($ETABLE{$equipment} )) {
99     print "Bad format of the equipment '$equipment'.\n";
100    usage();
101    exit($STATES{UNKNOWN});
102 }
```

Plugin je v současnosti napsán pouze pro verzi protokolu SNMP 1 a 2 (2c), proto je na řádcích 104-108 provedena kontrola pouze na první dvě verze.

```
104 if ($protocolV != 1 && $protocolV != 2) {
105    print "Protocol version can be only 1 or 2.\n";
106    usage();
107    exit($STATES{UNKNOWN});
108 }
```

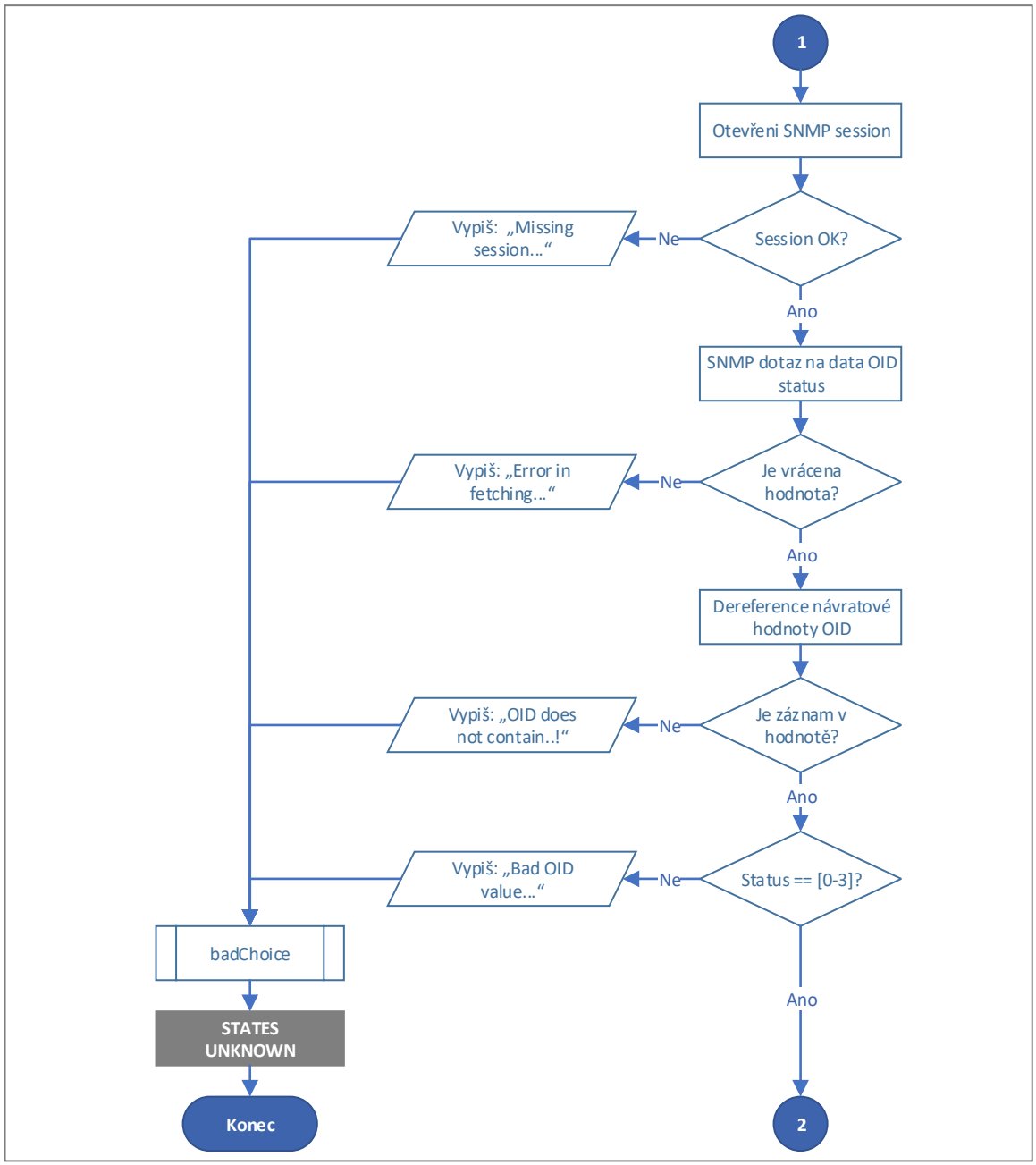
Pomocí funkce open FH (FILEHANDLE) je otevřen a následně načten MIB soubor. Cyklem while se prochází text souboru řádek po řádku a pomocí regulárního výrazu na řádku 115 jsou uloženy do hashe „MIBIDINFO“ pouze hodnoty obsahující číselný kód chybové zprávy (jako klíč hashe) a info text k chybě (jako hodnota hashe). Zde je příklad akceptovaného textu regulárním výrazem:

e105308-UDR-prekrocena-max-delka-zaznamu

```
111 open FH, $mibFilePath or die("Can not open the MIB file.\n");
112 my %MIBIDINFO = ();
113 while (my $line = <FH>) {
114    if ($line =~ /e(\d+)-([a-zA-Z0-9-]+)/) {
115      $MIBIDINFO{$1} = $2;
116    }
117 }
118 close(FH);
```

4.4.1.2 Druhá část pluginu check_snmp_errortable

V druhé části je otevřeno SNMP spojení včetně kontroly spojení. Poté je spuštěn SNMP dotaz na data OID Status proměnné. Následují kontroly návratové hodnoty. Je provedena dereference hodnoty SNMP dotazu. V závěru je provedena kontrola Status hodnoty, viz obrázek 39.



Obrázek 39 - Vývojový diagram 2. část pluginu check_snmp_errortable
Zdroj: Autor

Nejdříve je pomocí modulu Net::SNMP otevřeno spojení SNMP.

```
121 my ($session, $error) = Net::SNMP->session(
122    -hostname     => $host,
123    -community    => $community,
124    -version      => $protocolV
125 );
```

Následuje kontrola spojení SNMP. V případě neúspěchu je vypsána hláška obsahující chybové hlášení SNMP a vypsán text podprogramu „badChoice“ (ř. 262-266, viz příloha 1).

```
128 if (!defined $session) {
129    print "Missing session: $error\n";
130    badChoice();
131    exit($STATES{UNKNOWN});
132 }
```

Na řádku 135 je do pole „oids“ uloženo OID Status proměnné podle zvoleného klíče hashe „STATUS“ a pomocí zvoleného typu zařízení přepínačem „-E“ (equipment). Následuje spuštění SNMP dotazu metodou get request. Pokud je vše v pořádku, návratová hodnota se uloží do proměnné „resultOid“, která obsahuje referenci (odkaz) na hash obsahující záznam zvoleného OID.

```
135 my @oids = ($ESTATUS{$equipment});
136 my $resultOid = $session->get_request(
137    -varbindlist => \@oids
138 );
```

Pokud SNMP dotaz nevrátí žádný výsledek, je vypsána chybová hláška a vypsán text podprogramu „badChoice“.

```
141 if (!defined $resultOid) {
142    print "Error in fetching OID request: ", $session->error(), "\n";
143    badChoice();
144    $session->close();
145    exit($STATES{UNKNOWN});
146 }
```

V případě, že SNMP dotaz vrátí návratovou hodnotu s referencí na hash, je na řádku 149 provedena dereference této reference a záznam OID je uložen do hashe „hashOid“.

```
149 my %hashOid = %{$resultOid};
```

Mohlo by se ale stát, že záznam OID neobsahuje žádnou hodnotu. Proto je provedena další kontrola existence této hodnoty (řádky 152-158).

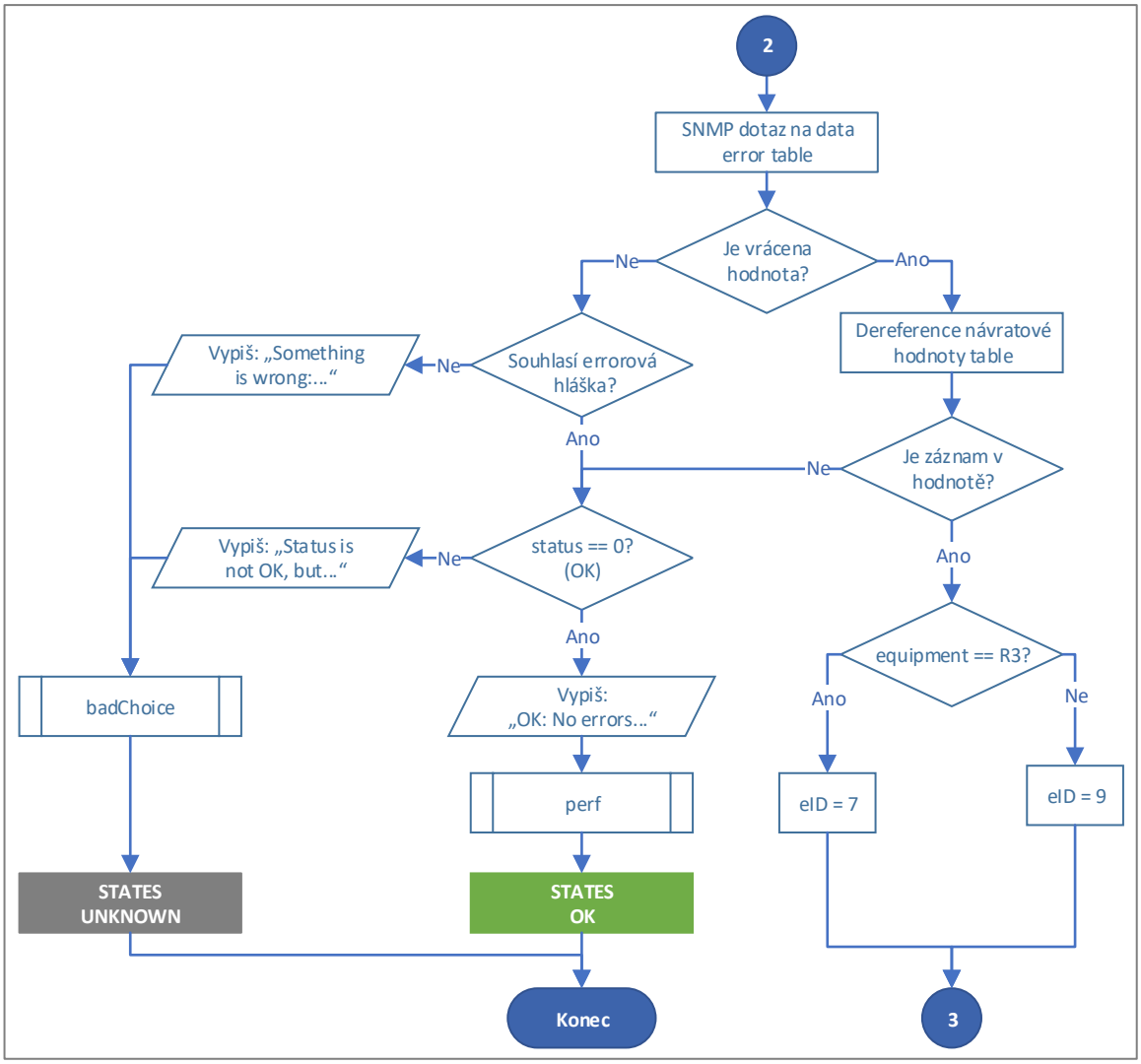
```
152 my $valueOid = $hashOid{$ESTATUS{$equipment}};
153 if (!defined $valueOid) {
154    print "OID '$ESTATUS{$equipment}' does not contain a value!\n";
155    badChoice();
156    $session->close();
157    exit($STATES{UNKNOWN});
158 }
```

Pokud záznam hodnotu obsahuje, je následně provedena kontrola očekávaných hodnot.

```
161 if ($valueOid !~ /^([0-3])$/) {
162    print "Bad OID value '$valueOid'.\n";
163    badChoice();
164    $session->close();
165    exit($STATES{UNKNOWN});
166 }
```

4.4.1.3 Třetí část pluginu check_snmp_errortable

Ve třetí části je spuštěn SNMP dotaz na tabulku chyb. Je provedena dereference SNMP dotazu. Následují víceúrovňové kontroly návratové hodnoty tabulky a při kladném vyhodnocení spolu se Status proměnnou je vrácen stav pluginu OK, viz obrázek 40.



Obrázek 40 - Vývojový diagram 3. část pluginu check_snmp_errortable
Zdroj: Autor

Nejdříve je spuštěn SNMP dotaz metodou get table. Pokud je vše v pořádku, návratová hodnota se uloží do proměnné „result“, která obsahuje referenci (odkaz) na hash obsahující záznam/záznamy zvoleného OID tabulky.

```
169 my $result = $session->get_table(
170    -baseoid => ($ETABLE{$equipment})
171 );
```

Pokud tabulka neobsahuje záznamy o chybách, SNMP vyhodnotí prázdnou tabulku jako neexistující a nevrátí žádnou referenci do proměnné „result“. Proto je na řádku 174 provedena kontrola proměnné „result“. Pokud SNMP vrátí „error“ hlášku začínající textem „The requested table is empty“, viz řádek 175, je spuštěn podprogram „checkOK“.

```
174 if (!defined $result) {
175    if ($session->error() gt 'The requested table is empty') {
176       checkOK();
177       } else {
178          print "Error in fetching table request: ",$session->error(),"\n";
179          badChoice();
180          $session->close();
181          exit($STATES{UNKNOWN});
182       }
183 }
```

Pomocí tohoto podprogramu je následně provedena kontrola hodnoty status proměnné. Tím se ověří celkový stav OK, kdy tabulka chyb neexistuje a současně status zařízení je OK. V ostatních případech vrátí plugin chybová hlášení. Při stavu OK se navíc zavolá podprogram „perf“ (ř. 286-289, viz příloha 1) a je vrácena hodnota performance data s výchozí hodnotou počítadla řádku „0“.

```
269 sub checkOK {
270    if ($valueOid eq '0') {
271       print "OK: No errors, $equipment status is OK.\n",
272       $session->error(), "\n";
273       perf();
274       $session->close();
275       exit($STATES{OK});
276    } else {
277       print "Status is not OK ("
278       . $SEVERITYTEXT{$valueOid}
279       . ") but the error table is empty or does not exist.\n";
280       badChoice();
281       $session->close();
282       exit($STATES{UNKNOWN});
283    }
284 }
```

Pokud je tedy návratová hodnota (reference na hash) SNMP dotazu uložena do proměnné „result“, pokračujeme na řádku 186 dereferencí. Tím se skutečné hodnoty SNMP dotazu uloží do hashe „values“.

```
186 my %values = %{$result};
```

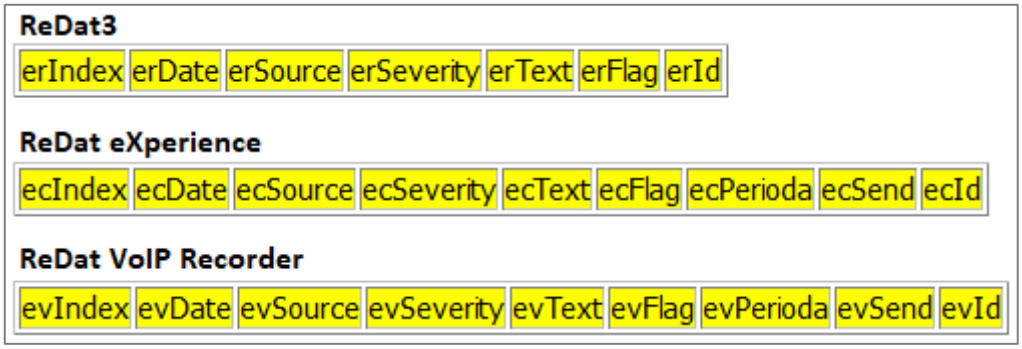
Následně na řádku 189 je spočítán počet prvků v hashi „values“. Pokud hash neobsahuje žádné prvky, je opět provedena kontrola podprogramem „checkOK“. K tomuto stavu by podle chování protokolu SNMP sice nemělo dojít, ale pro jistotu i v tomto případě je provedena kontrola prázdné tabulky.

```
189 my @values = %values;
190 if (scalar @values == 0) {
191    checkOK();
192 }
```

Na řádku 195 je zcela jisté, že tabulka chyb obsahuje záznamy. Proto začneme s přípravou na práci s hodnotami v tabulce.

```
195 if ($equipment eq 'R3') {
196    $eId = 7;
197 } else {
198    $eId = 9;
199 }
```

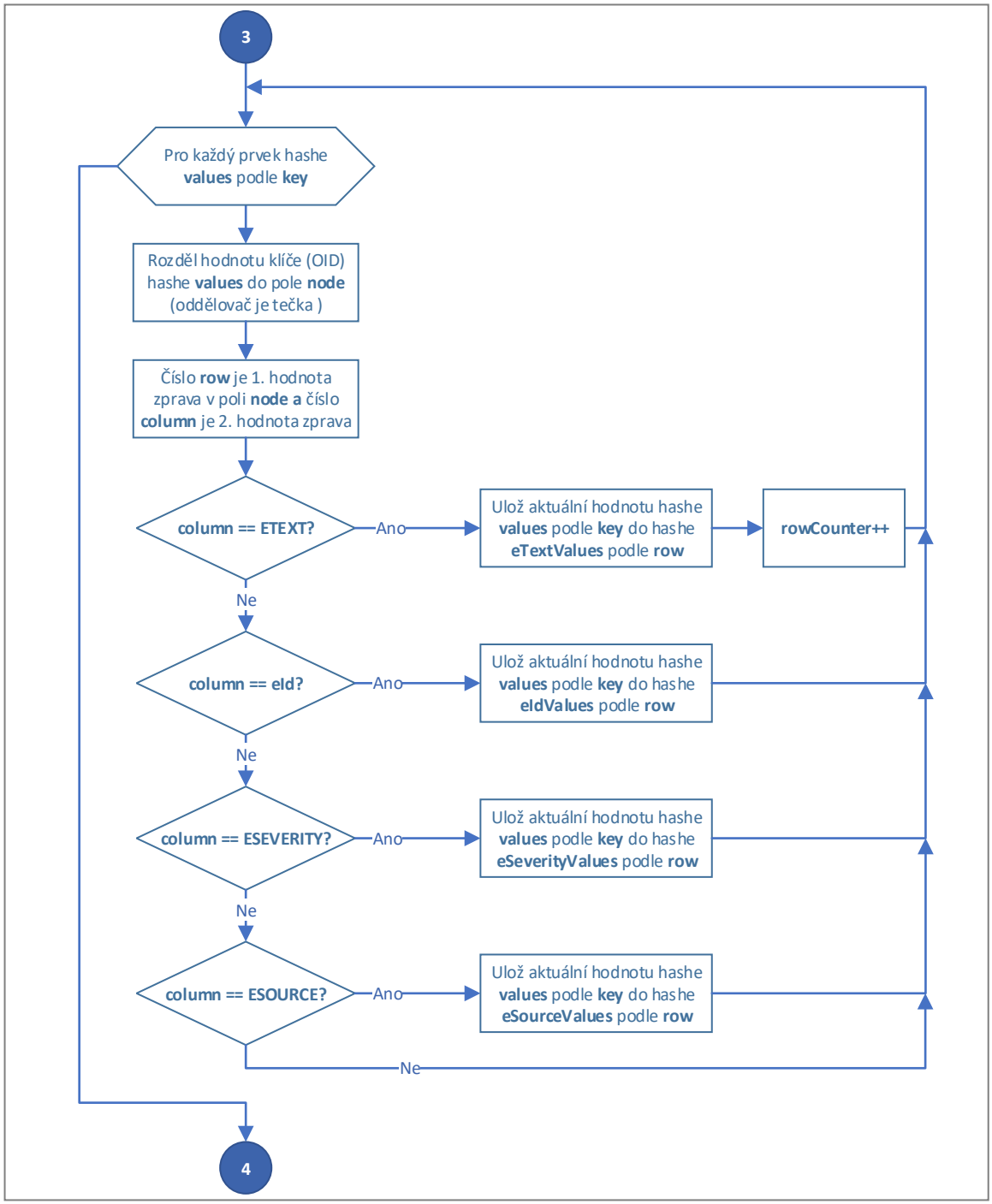
V úvodu uvádíme, že sloupec „Id“ je z důvodu jiné struktury tabulky R3 a REX (resp. RVR) deklarován jako proměnná. Proto na řádku 195 provádíme kontrolu typu zařízení a následně přiřazujeme proměnné „Id“ číslo sloupce. Na obrázku 41 je vidět, že sloupec Id je v zařízení R3 na 7. pozici zleva a v zařízeních REX a RVR je Id na 9. pozici.



Obrázek 41 - Error tabulky ReDat dle zařízení
Zdroj: Výstup aplikace SnmpB (Table View výřez)

4.4.1.4 Čtvrtá část pluginu check_snmp_errortable

Ve čtvrté části je provedeno parsování hodnot z hashe „value“ do jednotlivých hashů představujících sloupce chybové tabulky, viz obrázek 42.



Obrázek 42 - Vývojový diagram 4. část pluginu check_snmp_errortable
Zdroj: Autor

Záznamy z tabulky chyb jsou v hashi „values“ uloženy tak, jak byly získány ze SNMP dotazu, tzn. číselný tvar OID jako klíč hashe a hodnota proměnné OID jako hodnota hashe, například:

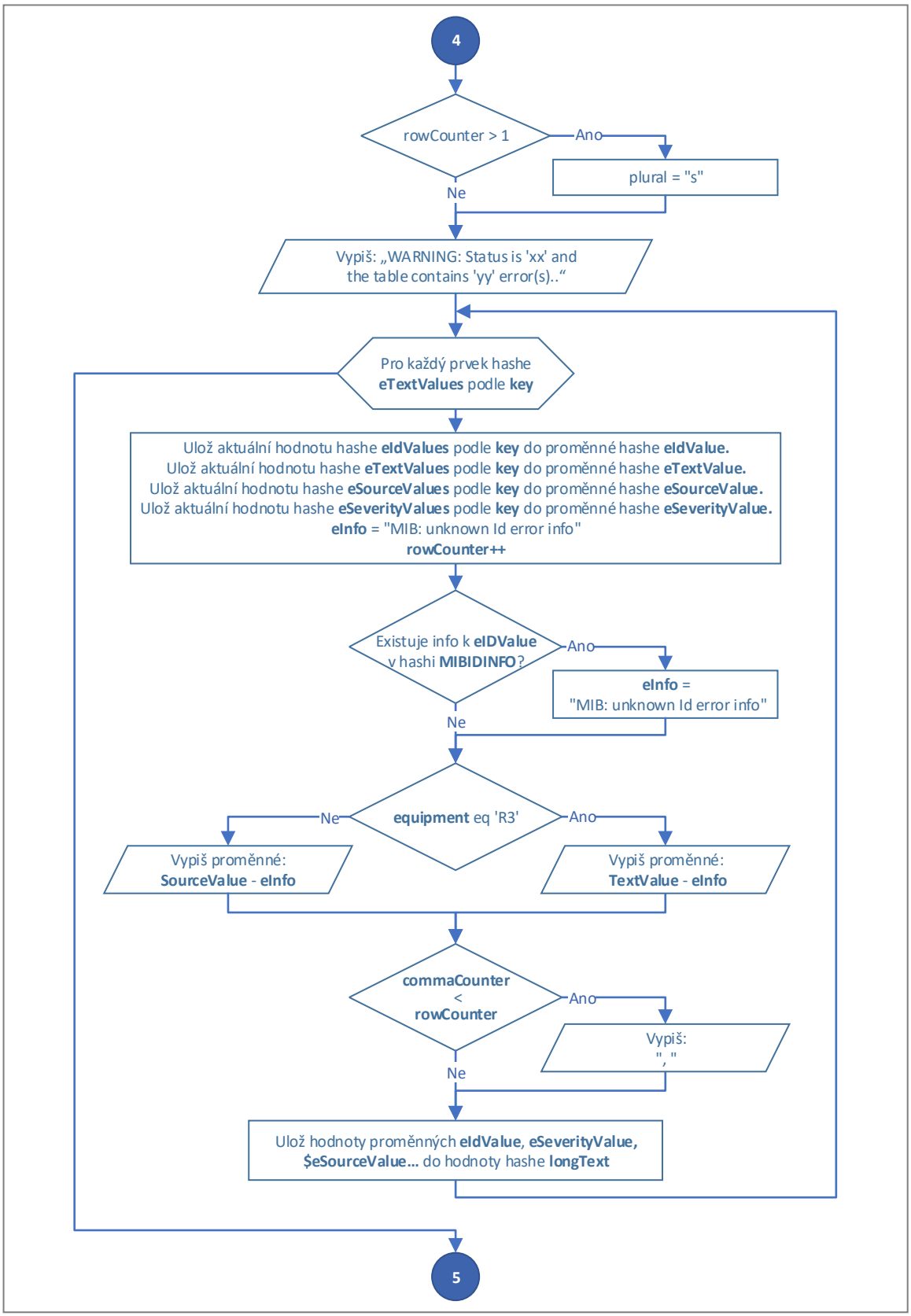
klíč: 1.3.6.1.4.1.17607.1.8.1.3.1, hodnota: UDR

Pro závěrečný výpis záznamů chybové tabulky potřebujeme pomocí následujícího cyklu nalézt a seskupit odpovídající hodnoty podle sloupců tabulky. Na řádku 203 pomocí funkce ***split*** rozdělíme hodnotu klíče do pole „node“ (oddělovač je tečka). Následně do proměnné „row“ uložíme 1. hodnotu zprava z pole „node“, což odpovídá číslu záznamu (řádku) v tabulce. Obdobně do proměnné „column“ uložíme 2. hodnotu zprava z pole „node“, což odpovídá číslu sloupce tabulky. Pomocí kontroly, zda proměnná „column“ odpovídá číslu hledaného sloupce, následně provedeme uložení aktuální hodnoty hashe „values“ do hodnoty hashe představující hledaný sloupec. Výsledkem tohoto cyklu bude uložení jednotlivých hodnot záznamů do čtyř hashů (eTextValues, eIdValues, eSeverityValues a eSourceValues) představujících sloupce tabulky. Dále jsou pomocí proměnné rowCounter při každém průchodu cyklu u sloupce „ETEXT“ počítány řádky tabulky.

```
202 foreach my $key (keys %values ) {
203    my @nodes = split /\./, $key;
204    my $row = $nodes[-1];
205    my $column = $nodes[-2];
206    if ($column == $ETEXT) {
207      $eTextValues{$row} = $values{$key};
208      $rowCounter++;
209    } elsif ($column == $eId) {
210      $eIdValues{$row} = $values{$key};
211    } elsif ($column == $ESEVERITY) {
212      $eSeverityValues{$row} = $values{$key};
213    } elsif ($column == $ESOURCE) {
214      $eSourceValues{$row} = $values{$key};
215    }
216 }
```

4.4.1.5 Pátá část pluginu check_snmp_errortable

V páté části pluginu je nejdříve vypsána první část základního výstupního textu. Následně je pomocí cyklu vypsána druhá část základního textu obsahující dvě základní hodnoty z chybové tabulky pro každý záznam. Dále jsou zde sestaveny jednotlivé záznamy se všemi hodnotami pro dlouhý výstupní text, viz obrázek 43.



Obrázek 43 - Vývojový diagram 5. část pluginu check_snmp_errortable
Zdroj: Autor

V případě více než jednoho záznamu v tabulce chyb je pak ve výstupním textu doplněno písmeno „s“ pro množné číslo slova „error“ pomocí proměnné „plural“.

Na řádku 222 začíná výpis první části základního výstupního textu obsahující aktuální textový popis stavu zařízení a počet chyb zařízení.

```
219 if ($rowCounter > 1) {
220    $plural = "s";
221 }
```

Cyklus začínající na řádku 224 postupně prochází jednotlivé sloupcové hashe a do proměnných jsou ukládány hodnoty z těchto hashů. Tzn., že při každém průchodu cyklu se načtou hodnoty z jednoho řádku. Zároveň se tyto hodnoty, odpovídající jednomu záznamu, vždy uloží jako jedna hodnota do hashe „longText“, viz řádek 242 a 243.

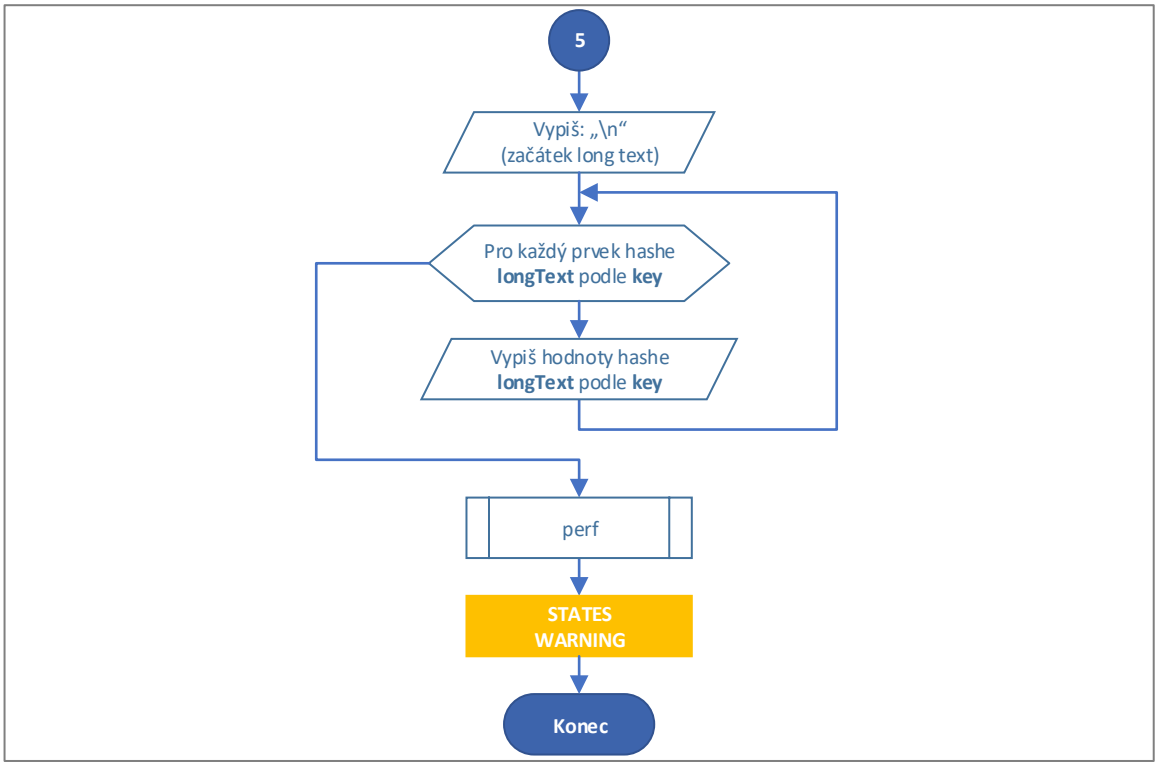
```
222 print "WARNING: Status is '$SEVERITYTEXT{$valueOid}'"
223 . " and the table contains $rowCounter error$plural: ";
224 foreach my $key (sort keys %eTextValues) {
225    my $eIdValue = $eIdValues{$key};
226    my $eTextValue = $eTextValues{$key};
227    my $eSourceValue = $eSourceValues{$key};
228    my $eSeverityValue = $eSeverityValues{$key};
229    my $eInfo = "MIB: unknown Id error info";
230    $commaCounter++;
231    if (defined $MIBIDINFO{$eIdValue}) {
232      $eInfo = $MIBIDINFO{$eIdValue};
233    }
234    if ($equipment eq 'R3') {
235      print "$eTextValue - $eInfo";
236    } else {
237      print "$eSourceValue - $eInfo";
238    }
239    if ($commaCounter < $rowCounter) {
240      print ", ";
241    }
242    $longText{$key} = ("$eIdValue, $SEVERITYTEXT{$eSeverityValue}, "
243    . "$eSourceValue: $eTextValue, Info: $eInfo");
244 }
```

Dále je při každém průchodu cyklu vypsána druhá část základního textu obsahující dvě základní hodnoty z chybové tabulky pro každý záznam, viz ř. 234-238. Pomocí čítače na ř. 230 a podmínky na ř. 239-241 je mezi jednotlivé záznamy vložena čárka.

Na řádku 231-232 je také doplněn info text ke kódu chyby z hashe „MIBIDINFO“.

4.4.1.6 Šestá část pluginu check_snmp_errortable

Šestá závěrečná část pluginu zajišťuje výpis dlouhého výstupního textu, výpis performance data a vrácení stavu WARNING Nagiosu.



Obrázek 44 - Vývojový diagram 6. část pluginu check_snmp_errortable
Zdroj: Autor

Na řádku 247 je výpis zalomení řádku, který odděluje základní výstupní text pluginu od dlouhého výstupního textu. Cyklus na řádku 248-250 zajistí výpis jednotlivých záznamů dlouhého textu.

```
247 print "\n";
248 foreach my $key (sort keys %longText){
249     print "$longText{$key}\n";
250 }
```

Následuje výpis performance dat pomocí podprogramu „perf“.

```
253 perf();
```

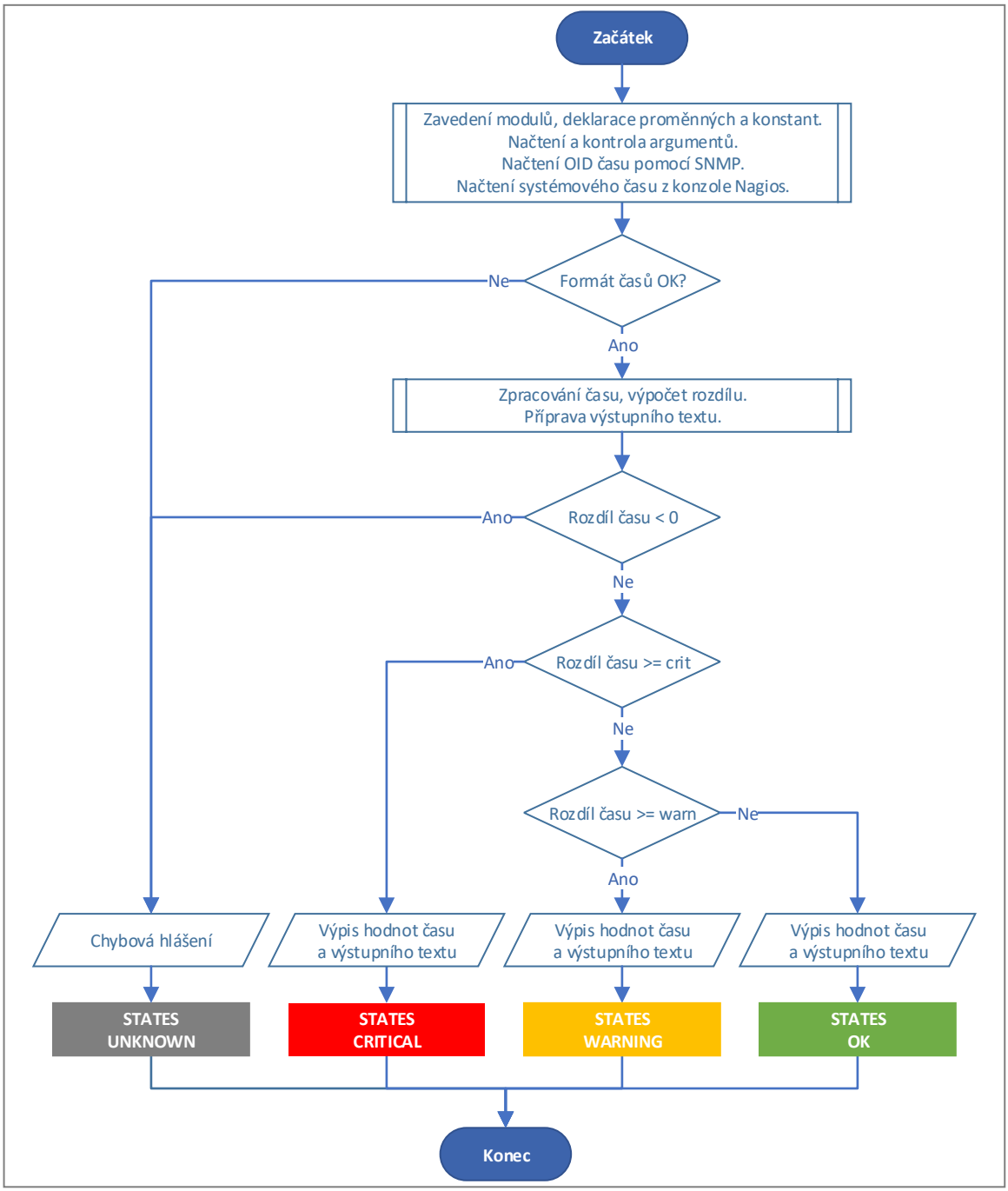
```
287 sub perf {
288     print "|Errors=$rowCounter\n";
289 }
```

Na závěr je Nagiosu vrácen stav WARNING a ukončen plugin.

```
256 $session->close();
257 exit($STATES{WARNING});
```

4.4.2 Plugin pro kontrolu zápisu na disk ReDat

Plugin „check_snmp_lastwrite.pl“ je určen pro kontrolu posledního zápisu na disk zařízení ReDat R3 a REX, viz příloha 2. Na obrázku 45 je zjednodušeně znázorněn celý algoritmus pluginu.



Obrázek 45 - Vývojový diagram algoritmu pluginu check_snmp_lastwrite
Zdroj: Autor

Některé části kódu jsou obdobné jako u předchozího pluginu, konkrétně deklarace proměnných a konstant, zavedení modulů, načtení argumentů, kontrola argumentů, otevření SNMP spojení a spuštění SNMP dotazu. Proto si v následujících odstavcích popíšeme pouze části kódu, které se týkají zpracování času. Pro práci s časem použijeme následující modul.

```
15 use Time::Piece;
```

V hashi „TIMEOID“ jsou předdefinovány OID jednotlivých zařízení, odkud se bude získávat hodnota času posledního zápisu na disk. Jedná se o oblast disku, kam jsou ukládány záznamy, resp. archiv záznamů.

Hash „TIMEZONES“ slouží pro modifikaci příkazu „date“, kterým se získává systémový čas z příkazového řádku Nagiosu. Nagios se řídí časem z NTP serveru, který se automaticky přepíná na letní a zimní čas. V hashi „TIMEZONES“ tento typ času označujeme „CZ“. Zařízení REX využívá tento typ času, ale R3 využívá „UTC 1“ (zimní čas).

```
24 my %TIMEOID = (
25 "R3"       => '1.3.6.1.4.1.17607.1.4.1.6.1',
26 "REX"      => '1.3.6.1.4.1.17607.5.4.1.4.4'
27 );
28
29 my %TIMEZONES = (
30 "UTC"      => "date -u ",
31 "UTC1"     => "date -u -d '+1 hour' ",
32 "UTC2"     => "date -u -d '+2 hour' ",
33 "CZ"       => "date "
34 );
```

Na řádku 146 začíná kontrola času získaného z OID hodnoty pomocí regulárního výrazu. OID čas je v podobě textového řetězce. Proto následuje konverze a parsování řetězce na požadovaný formát času pomocí funkce ***strptime***, viz řádek 147, resp. 149.

```
146 if ($valueOid =~ /^(\d{4})\/(\d{2})\/(\d{2})_(\d{2}):(\d{2}):(\d{2})$/) {
147    $oidTime = Time::Piece->strptime($valueOid, "%Y/%m/%d_%H:%M:%S");
148 } elsif ($valueOid =~ /^(\d{4})-(\d{2})-(\d{2}) (\d{2}):(\d{2}):(\d{2})$/) {
149    $oidTime = Time::Piece->strptime($valueOid, "%Y-%m-%d %H:%M:%S");
150 } else {
151    print "Wrong OID time format: '$valueOid'.\n";
152    badChoice();
153    $session->close();
154    exit($STATES{UNKNOWN});
155 }
```

Pokud je formát OID času v pořádku, provedeme na ř. 158 načtení systémového času z příkazového řádku Nagiosu pomocí funkce ***open***. Návratová hodnota se systémovým časem je pak na ř. 161 očištěna od všech zalomení řádků.

```
158 open CH, "$TIMEZONES{$timeZone} '+%F %T' |";
159 my $consoleTime = <CH>;
160 close CH;
161 $consoleTime =~ s/\n//;
```

Následuje kontrola a parsování řetězce se systémovým časem jako v případě OID hodnoty.

```
164 if ($consoleTime =~ /^(\d{4})-(\d{2})-(\d{2}) (\d{2}):(\d{2}):(\d{2})/) {
165    $systemTime = Time::Piece->strptime($consoleTime,"%Y-%m-%d %H:%M:%S");
166 } else {
167    print "Wrong system time format: '$consoleTime'.\n";
168    $session->close();
169    exit($STATES{UNKNOWN});
170 }
```

Na řádku 173 je proveden výpočet rozdílu mezi časy v sekundách. Následuje přepočet rozdílu v minutách a hodinách. Dále je proveden výpočet zbytku minut a sekund po odečtení celých hodin a minut. Nepředpokládá se, že by plugin vyhodnocoval delší rozdíl, než je několik hodin, proto se dále nepracuje s přepočtem rozdílu na dny, měsíce atd.

```
173 my $diffInS = $systemTime - $oidTime;
174 my $diffInM = int($diffInS / 60);
175 my $diffInH = int($diffInM / 60);
176 my $restInM = $diffInM - ($diffInH * 60);
177 my $restInS = $diffInS - ($diffInM * 60);
```

Dalším předpokladem je, že při kontrole posledního zápisu na disku by nemělo docházet ke stavu, kdy je systémový čas opožděn oproti OID času posledního zápisu. Proto je na ř. 180-185 vyhodnocen záporný rozdíl jako chybový stav.

```
180 if ($diffInM < 0) {
181    print "System time error or wrong time zone " . $timeZone . "\n"
182    . "(system time '$systemTime' is delayed over OID time '$oidTime')\n";
183    $session->close();
184    exit($STATES{UNKNOWN});
185 }
```

Následuje příprava výstupního textu pomocí proměnné „outputText“. Řádek 188 obsahuje krátký výstupní text. Řádky 189-192 obsahují dlouhý text a řádky 193-194 performace data.

```
188 my $outputText = "last write to disc: " . $oidTime
189 . "\nWrite before: "
190 . sprintf("%02d:%02d:%02d", $diffInH, $restInM, $restInS)
191 . "\nSystem time: " . $systemTime
192 . "\nTime zone: " . $timeZone
193 . "|LastWrite=$diffInS" . "seconds;"
194 . ($warn * 60) . ";" . ($crit * 60) . "\n";
```

V závěru jsou vyhodnoceny stavy pluginu na základě prahových hodnot „crit“ a „warn“. Společně s návratovou hodnotou stavu pluginu je Nagiosu vrácen i kompletní výstupní text.

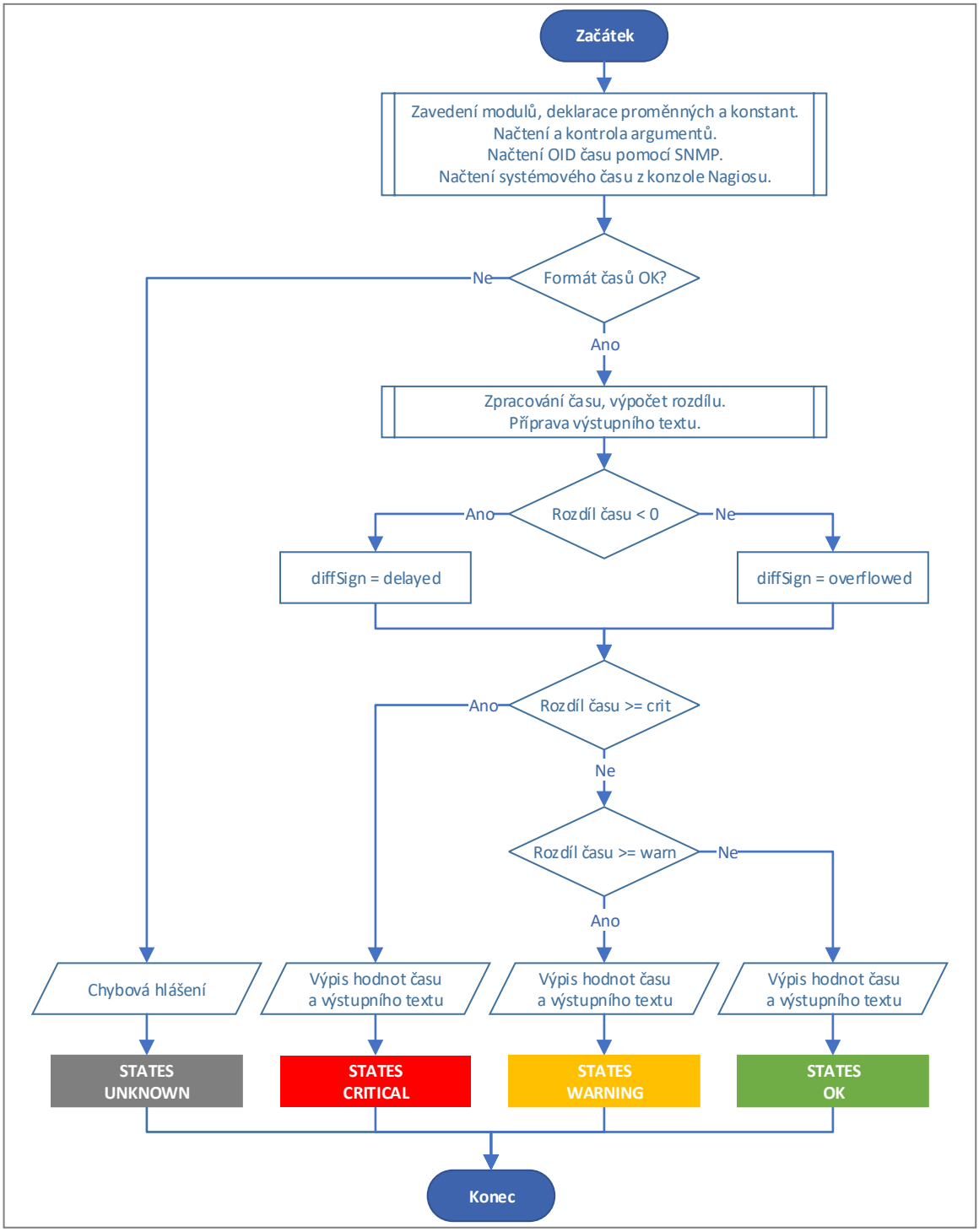
```
197 if ($diffInM >= $crit) {
198    print "CRITICAL: " . $outputText;
199    $session->close();
200    exit($STATES{CRITICAL});
201 } elsif ($diffInM >= $warn) {
202    print "WARNING: " . $outputText;
203    $session->close();
204    exit($STATES{WARNING});
205 } else {
206    print "OK: " . $outputText;
207    $session->close();
208    exit($STATES{OK});
209 }
```

4.4.3 Plugin pro kontrolu času

Plugin „check_snmp_time.pl“, viz příloha 3, je určený pro kontrolu systémového času hostitele pomocí SNMP dotazu na jednoduchou proměnou OID. Tento plugin vychází z předchozího pluginu a obsahuje navíc tyto funkce:

- OID není přednastavené, ale zadává se jako argument přepínače „-o“,
- akceptace 4 formátů času (navíc formát Unix Epoch a OctetString-HexaString),
- vyhodnocení rozdílu času opoždění/předcházení,
- kontrola rozdílu času přesně o 1 a 2 hodiny (zřejmě chybně zvolená časová zóna),
- vyhodnocení prahových hodnot v minutách nebo sekundách,
- vložení libovolného popisku zařízení.

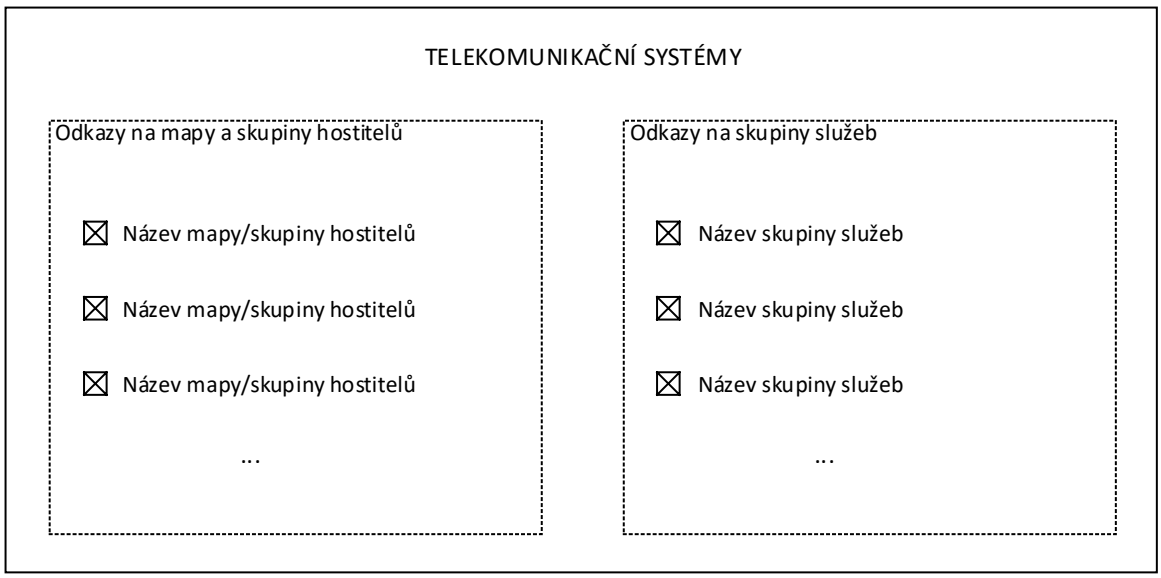
Na obrázku 46 je zjednodušeně znázorněn celý algoritmus pluginu.



Obrázek 46 - Vývojový diagram algoritmu pluginu check_snmp_time
Zdroj: Autor

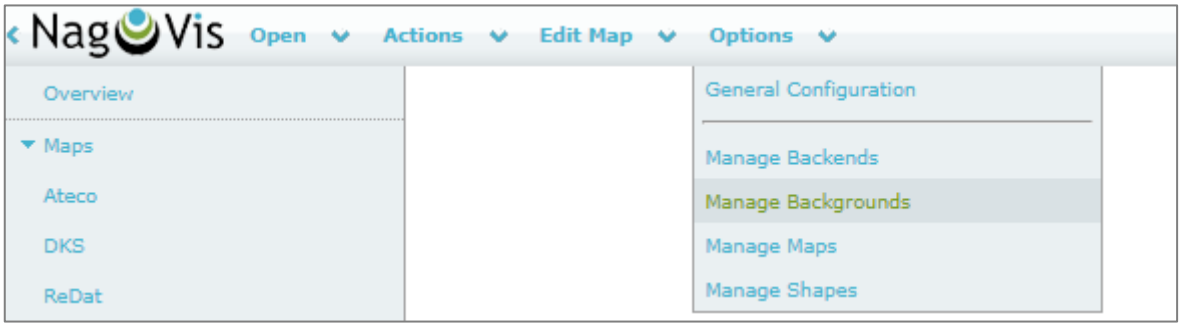
4.5 Konfigurace NagVisu

Při návrhu základní mapy pro zobrazení dat z Nagiosu pomocí nadstavby NagVis byl zvolen jednoduchý design, viz obrázek 47. Levá část obsahuje odkazy na další mapy nebo skupiny hostitelů včetně jejich služeb. Pravá část pak obsahuje vybrané skupiny služeb. Seskupení hostitelů a služeb bylo zvoleno s ohledem na logické členění systémů a zařízení.



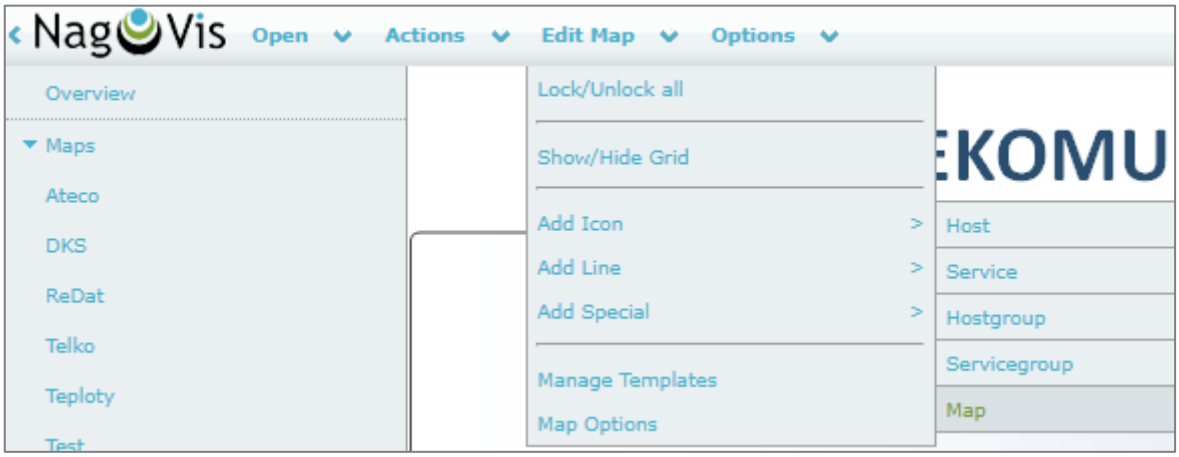
Obrázek 47 - Wireframe NagVis mapy Telekomunikační systémy
Zdroj: Autor

Převážná část konfigurace map byla tvořena pomocí velmi intuitivního webového rozhraní NagVisu. Před samotnou konfigurací je potřeba připravit grafické pozadí pro jednotlivé mapy. Všechna grafická pozadí byla vytvořena v aplikaci MS Visio. Grafická pozadí se vkládají a nové mapy vytvářejí pomocí menu ***Options***, viz obrázek 48.



Obrázek 48 - NagVis menu Options
Zdroj: NagVis web rozhraní (výřez)

Pomocí menu ***Edit Map*** se pak mapy editují a na mapy vkládají objekty hostitelů, služeb a jejich skupin, viz obrázek 49.

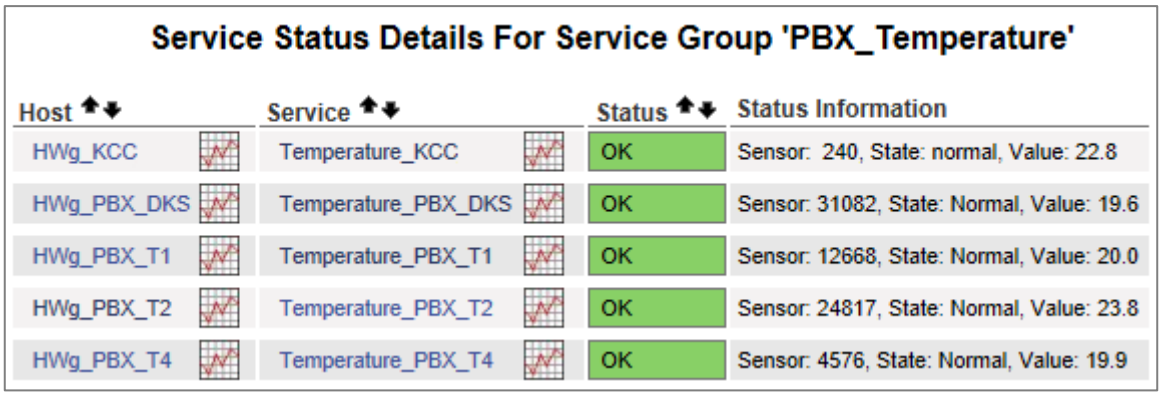


Obrázek 49 - NagVis menu Edit Map
Zdroj: NagVis web rozhraní (výřez)

Na obrázku 50 je zobrazena základní mapa PBX systému, která je přístupná pomocí nástěnných LCD monitorů pracovišti administrátorů STS a centrálnímu ICT Helpdesku.

Obrázek 50 - NagVis mapa Telekomunikační systémy
Zdroj: Autor

První tři objekty na pravé straně jsou odkazy na další mapy. Tyto mapy obsahují nejdůležitější skupiny hostitelů a jejich služeb v PBX systému. Mapy jsou součástí příloh 4 až 6 této DP. Ostatní objekty představují odkazy na skupiny hostitelů nebo služeb na příslušné ***cgi*** soubory v Nagiosu, viz příklad skupiny „PBX_Temperature na obrázku 51.

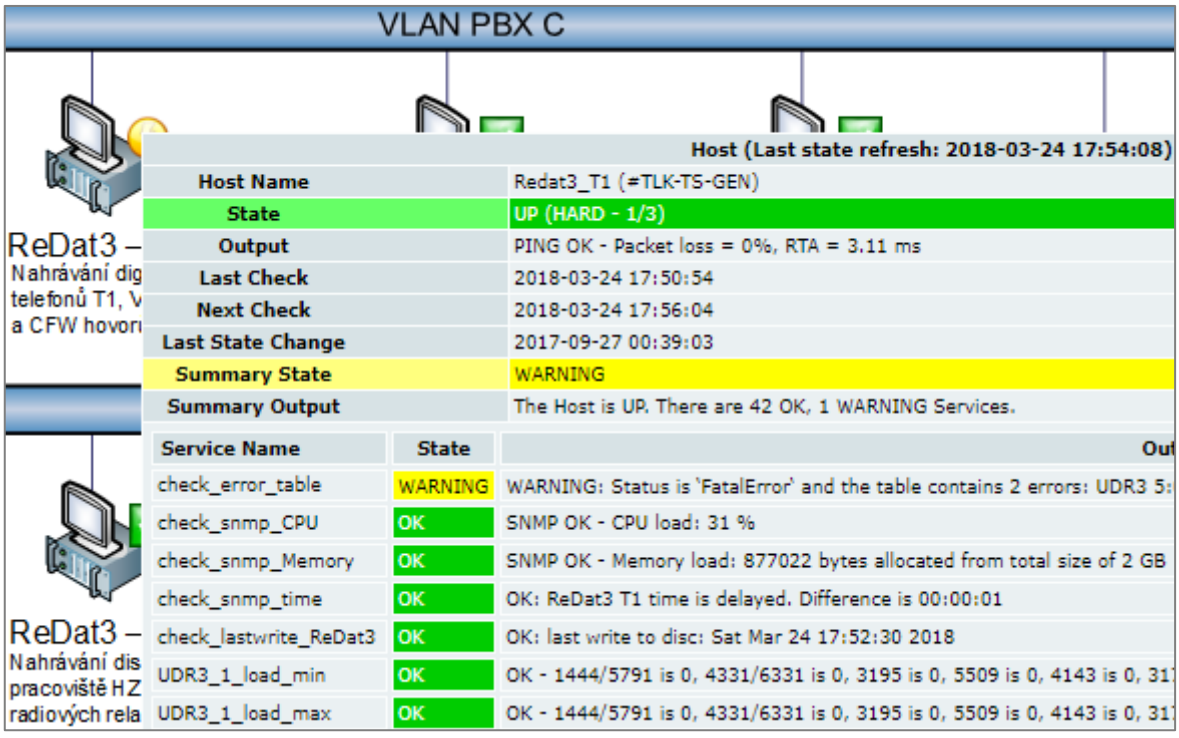


Service Status Details For Service Group 'PBX_Temperature'

Host	Service	Status	Status Information
HWg_KCC	Temperature_KCC	OK	Sensor: 240, State: normal, Value: 22.8
HWg_PBX_DKS	Temperature_PBX_DKS	OK	Sensor: 31082, State: Normal, Value: 19.6
HWg_PBX_T1	Temperature_PBX_T1	OK	Sensor: 12668, State: Normal, Value: 20.0
HWg_PBX_T2	Temperature_PBX_T2	OK	Sensor: 24817, State: Normal, Value: 23.8
HWg_PBX_T4	Temperature_PBX_T4	OK	Sensor: 4576, State: Normal, Value: 19.9

Obrázek 51 - Nagios service status group PBX_Temperature
Zdroj: Nagios web rozhraní (výřez)

Po skončení konfigurace mapy je potřeba všechny objekty zamknout pomocí menu ***Edit Map*** funkce ***Lock***. Tím je umožněno, po pouhém najetí kurzoru na objekt, zobrazení podrobného náhledu stavů konkrétního objektu, viz příklad na obrázku 52.



Obrázek 52 - NagVis mapa ReDat, status Warning ReDat3 T1
Zdroj: Nagios web rozhraní (výřez)

4.6 Testování a řešené problémy

Testování jednotlivých pluginů a kontrol probíhalo jak na testovacích, tak i na produkčních systémech a zařízeních. Testování bylo prováděno simulací výpadku hostitelů a služeb, snižováním prahových hodnot jednotlivých kontrol tak, aby byla ověřena funkčnost detekce stavů OK, WARNING a CRITICAL. Při těchto testech a v produkčním prostředí bylo řešeno několik problémů, viz níže.

Původní kontrola SNMP tabulky chyb a Status proměnné hostitelů ReDat byla prováděna odděleně. Status proměnná byla kontrolována pluginem „check_snmp“ a tabulka chyb byla kontrolována vlastním pluginem „check_snmp_errortable“. Při testování vlastního pluginu se ukázalo, že stav OK při neexistenci tabulky není dostačujícím kontrolním mechanismem. Při chybném zadání OID tabulky nebo IP adresy hostitele plugin vyhodnotil stav OK bez dalšího upozornění. Z tohoto důvodu poslední verze pluginu kontroluje jak chybovou tabulku, tak i Status proměnnou současně a OID tabulky je zvoleno výběrem typu zařízení ReDat.

Plugin od výrobce HWg pro kontrolu čidel HWg-STE Plus obsahoval chybně zadané předdefinované OID signalizačních kontaktů.

Kontrola běžících procesů v OS Windows XP pomocí NRPE Agenta a programu „check_winprocess.exe“ nebyla funkční. Program využívá ke kontrole program „task.exe“, který je součástí OS. Bylo zjištěno, že pokud je OS v české jazykové mutaci, tak u samotného programu „task.exe“ nefungují některé funkce. Po nahrazení programu „task.exe“ anglickou verzí byla kontrola běžících procesů v plném rozsahu zprovozněna.

Aplikace SmartRecord v KC Aastra, která zajišťuje záznamy hovorů SW telefonů, přestane občas zaznamenávat hovory, přestože služba programu je aktivní. Až restart služby vyvolaný administrátorem STS zajistí opětovnou funkčnost aplikace. Po důkladné analýze pomocí prohlížeče událostí v OS byla nalezena událost, která je spojena s tímto problémem. Pomocí Agenta NSClient++ a modulu „Check_EventLog“ je prováděna kontrola této události. Administrátoři STS jsou díky tomu ihned informování o výpadcích záznamů a mohou včas provádět restart služby do doby, než tento problém vyřeší výrobce aplikace.

5 Zhodnocení výsledků

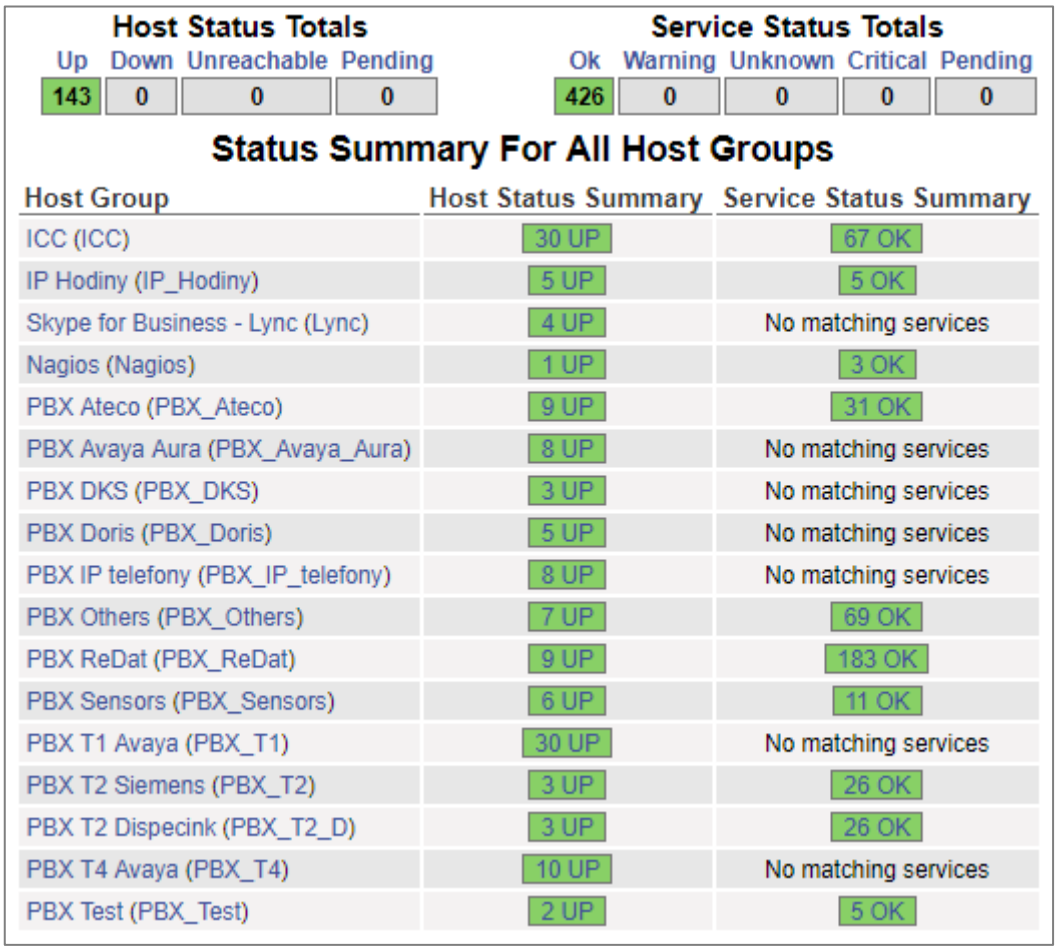
Pro monitorování PBX systému Zadavatele byl z důvodu neustálého snižování nákladů Zadavatele zvolen stávající systém Nagios, který využívají i jiná ICT oddělení. Systém Nagios je pro svou modulárnost a flexibilitu velmi vhodným řešením pro monitorování takto heterogenního PBX systému. Pro zajištění kompletního monitorování PBX systému by bylo nutné z důvodu jeho zastaralé technologie provést upgrade jednotlivých systémů. V některých případech to již není z důvodu ukončení podpory výrobcem možné. Plánovaná kompletní obnova PBX systému byla z důvodu očekávaných vysokých finančních nákladů několikrát posunuta. Proto nebylo možné zajistit přímé monitorování všech systémů. Jedná se především o monitorování služeb samotných telefonních ústředen. Vzhledem k tomu, že se ale jedná o velmi spolehlivá zařízení, tak absence monitorování neovlivnila provoz PBX systému. Další systém, kde nebylo možné nasadit přímé monitorování, je dispečerský systém Trading Siemens. Díky tomu, že jsou všechna koncová dispečerská zařízení nahrávána záznamovým systémem ReDat, je zajištěno alespoň nepřímé monitorování těchto zařízení pomocí kontrol jednotlivých záznamových kanálů. Ovšem největší problémy vždy byly s ostatními nadstavbami a doplňujícími systémy, kde bylo monitorování hostitelů a služeb pomocí systému Nagios zcela zajištěno.

Nejčastějším problémem před nasazením monitorování Nagiosem byla především funkčnost dohledové aplikace ReDat Explorer a s tím související problémy s dohledem a funkčností systému ReDat. Častým problémem byla také částečná nebo úplná nefunkčnost některých nadstavbových aplikací a systémů při dosažení plné kapacity lokálního pevného disku nebo operační paměti konkrétního zařízení. Dalším problémem byly časté výpadky jedné z klimatizačních jednotek v technologické místnosti. Občas došlo i k chybě způsobené samotnými administrátory STS, kteří zapomněli zapnout klimatizační jednotku při odchodu z technologické místnosti. Nefunkčnost některých zařízení byla zjištěna až při fyzické kontrole nebo díky nahlášení problému uživatelem.

Specifikace požadavků na monitorování hostitelů a služeb PBX systému byla vytvořena na základě zkušeností autora této diplomové práce na pozici administrátora PBX systému Zadavatele a také analýzy předešlých problémů a poruch PBX systému. Tato

specifikace požadavků byla zdrojem pro analýzu a výběr nejvhodnějších typů Nagios pluginů a typů Agenta na straně hostitelů. Vzhledem k tomu, že v rámci tvorby této diplomové práce bylo provedeno pilotní monitorování systému ReDat systémem Nagios pomocí protokolu SNMP, bylo nutné vytvořit pro určité typy kontrol vlastní pluginy pomocí programovacího jazyka Perl. Dále bylo zjištěno, že není mnoho dostupných pluginů pro kontrolu časových údajů v různých formátech. Proto byl vytvořen univerzální plugin pro kontrolu systémového času zařízení pomocí SNMP protokolu.

Za poslední měsíce provozu monitorování systému PBX nebyly zaznamenány nahlášené poruchy monitorovaných systémů koncovým uživatelem. Při nedostupnosti systému nebo služby je vždy předem informováno pracoviště SDS a centrální ICT Helpdesk prostřednictvím Nagiosu. PBX systém je v současné době monitorován systémem Nagios pomocí cca 140 hostitelů a 420 služeb, viz obrázek 53.



Host Status Totals

Up	Down	Unreachable	Pending
143	0	0	0

Service Status Totals

Ok	Warning	Unknown	Critical	Pending
426	0	0	0	0

Status Summary For All Host Groups

Host Group	Host Status Summary	Service Status Summary
ICC (ICC)	30 UP	67 OK
IP Hodiny (IP_Hodiny)	5 UP	5 OK
Skype for Business - Lync (Lync)	4 UP	No matching services
Nagios (Nagios)	1 UP	3 OK
PBX Ateco (PBX_Ateco)	9 UP	31 OK
PBX Avaya Aura (PBX_Avaya_Aura)	8 UP	No matching services
PBX DKS (PBX_DKS)	3 UP	No matching services
PBX Doris (PBX_Doris)	5 UP	No matching services
PBX IP telefony (PBX_IP_telefony)	8 UP	No matching services
PBX Others (PBX_Others)	7 UP	69 OK
PBX ReDat (PBX_ReDat)	9 UP	183 OK
PBX Sensors (PBX_Sensors)	6 UP	11 OK
PBX T1 Avaya (PBX_T1)	30 UP	No matching services
PBX T2 Siemens (PBX_T2)	3 UP	26 OK
PBX T2 Dispecink (PBX_T2_D)	3 UP	26 OK
PBX T4 Avaya (PBX_T4)	10 UP	No matching services
PBX Test (PBX_Test)	2 UP	5 OK

Obrázek 53 - PBX systém Nagios status summary
Zdroj: Autor, webové rozhraní Nagios (výřez)

5.1 Ekonomické zhodnocení

Pro zohlednění úspor mzdových nákladů při redukci počtu administrátorů STS ze šesti na čtyři vycházíme z regionální statistiky ceny práce uvedené na portálu Ministerstva práce a sociálních věcí (MPSV, ©2018). Medián hrubé mzdy techniků v oblasti ICT pro Prahu činí 38 748 Kč měsíčně. To přináší Zadavateli celkovou měsíční úsporu 103 845 Kč (superhrubá mzda za dva administrátory STS).

V souvislosti se snížením počtu administrátorů došlo ke zrušení nočních služeb, služeb o víkendech a svátcích. Nově bylo zavedeno držení pohotovostí mimo pracovní dobu. Při porovnání původních nákladů na dřívější služby proti nově vzniklým nákladům na držení pohotovostí a nutné občasné výjezdy během nich nedošlo k výrazné změně. Rozdíl těchto nákladů se pohybuje měsíčně cca ± 2 000 Kč na jednoho administrátora STS.

Upgrade loggerů ReDat a náhrada Aplikačního serveru ReDat za ReDat eXperience byl plánovaný a byl by proveden bez ohledu na zvolený monitorovací systém, proto negeneruje vícenáklady.

Byl použit stávající systém Nagios (Nagios Core) Zadavatele, v jehož provozu a údržbě nedošlo k zásadní změně. Tento systém je open source, vydávaný pod GPL licencí a je přístupný zdarma. Konfigurace systému Nagios byla prováděna v rámci běžných pracovních činností autora této diplomové práce. Při využití a konfiguraci systému Nagios nevznikly tedy žádné nové náklady a budoucí aktualizace a upgrade systému Nagios by neměly ani další generovat.

Jediným nákladem, který vznikl při nasazení monitorování PBX systému, byl nákup pěti čidel HWg-STE Plus ve výši 28 500 Kč. Životnost těchto zařízení se běžně počítá na 2 až 5 let. Při rozpočtení na minimálních 24 měsíců životnosti činí tento náklad 1 188 Kč měsíčně.

Celková úspora na základě výše zmíněných nákladů činí 102 657 Kč měsíčně.

6 Závěr

Hlavním cílem DP bylo zajištění monitorování PBX systému pomocí open source systému Nagios pro společnost zabývající se poskytováním IT a telekomunikačních služeb. Mezi dílčí cíle patřila pilotní implementace monitorování záznamového systému ReDat do Nagiosu pomocí SNMP protokolu a tvorba vizualizace výstupních dat z Nagiosu pomocí aplikace NagVis.

Podnět pro potřebu zajištění monitorování PBX systému vznikl v době, kdy došlo v rámci snižování nákladů k redukci počtu administrátorů PBX systému ze šesti na čtyři, a tím došlo ke zrušení nočních služeb, služeb o víkendech a svátcích. Administrátoři dříve prováděli kontroly a dohled systému manuálně v rámci služeb H24, což již dále nebylo možné. Dalším podnětem bylo ukončení podpory a rozvoje monitorovací aplikace ReDat Explorer výrobcem Retia. V rámci snižování nákladů Zadavatele byla snaha co nejvíce využít stávající zdroje, technologie a systémy, popřípadě doplnit tyto systémy o svobodný software a software s otevřeným kódem. Proto byl pro zajištění monitorování PBX systému zvolen jeden ze stávajících monitorovacích systémů Nagios.

Všechny typy hostitelů (systémy, aplikace a zařízení) dostupné z počítačové sítě Zadavatele, patřící do PBX systému, byly začleněny do monitorování pomocí definování nových konfiguračních souborů v Nagiosu. Podle typu hostitele byly vybrány nejvhodnější typy Nagios pluginů a typy Agentů na straně hostitele. Bylo provedeno pilotní monitorování systému ReDat jak dostupnými SNMP pluginy, tak především vlastními pluginy vytvořenými v programovacím jazyce Perl. Ve vizualizační nadstavbě NagVis byly vytvořeny nové mapy PBX systému, které jsou přístupné jak pracovišti administrátorů STS, tak i ICT Helpdesku.

Přínos praktické části spočívá především v implementaci a tvorbě vlastních pluginů pro systém ReDat a tvorbě univerzálního pluginu pro kontrolu systémového času hostitelů pomocí SNMP protokolu. Pluginy jsou v praktické části velmi podrobně popsány a díky tomu mohou být inspirací a návodem pro jiné autory vlastních pluginů.

Zkušenosti autora této DP s konfigurací Nagiosu a tvorbou vlastních pluginů budou dále využity po plánované obnově stávajícího PBX systému při implementaci jeho monitorování.

Podařilo se zajistit částečné monitorování telefonních ústředen a kompletní monitorování nadstavbových systémů a aplikací bez dalších investic (s výjimkou investice do nákupu teplotních čidel v zanedbatelné výši). Naopak vznikly úspory na mzdových nákladech díky snížení počtu administrátorů PBX systému. Monitorování systému PBX zefektivnilo práci administrátorů tohoto systému. Dle zkušeností za poslední dva měsíce produkčního provozu monitorování se daří odhalovat problémy dříve, než je zaznamenají uživatelé.

7 Seznam odborné literatury a zdrojů

- BARTH, Wolfgang, 2008. *Nagios: system and network monitoring. 2nd ed.* Munich: Open Source Press. ISBN 1593271794.
- BAZALA, David, 2006 *Telekomunikace a VOIP telefonie*. Praha: BEN - technická literatura. ISBN 80-7300-201-9.
- BOUŠKA, Petr, 2010. *Rozumíme počítačovým sítím.* Connect! Brno: CPress Media. 2. 4/2010. s. 8-10. ISSN 1211-3085.
- BOLDIŠ, Ján, 1985. *Telefonie.* Praha: Nakladatelství dopravy a spojů. US-31-008-85.
- CLARK, Martin P, 2003 *Data networks, IP, and the Internet: protocols, design, and operation*. Hoboken, NJ: J. Wiley. ISBN 0-470-84856-1.
- DANIELE, Mike et al., 2000. *RFC 2741 Agent Extensibility (AgentX) Protocol.* [online]. [cit. 2018-01-08]. Dostupné z: https://tools.ietf.org/html/rfc2741
- FIALA, Jan, 2017. *Textový editor PSPad.* [online]. [cit. 2018-03-09]. Dostupné z: http://www.pspad.com/cz/
- FOROUZAN, Behrouz A, 2010. *TCP/IP protocol suite*. 4th ed. Boston: McGraw-Hill Higher Education. ISBN 978-0-07-337604-2.
- FREE SOFTWARE FOUNDATION, INC., ©2014-2016. *Licenses.* [online]. [cit. 2017-12-31]. Dostupné z: https://www.gnu.org/licenses/
- *H. Res. 269*, 2002. [online]. In the House of Representatives, U.S. [cit. 2018-01-09]. Dostupné z: https://www.congress.gov/107/bills/hres269/BILLS-107hres269eh.pdf
- HO, Don, ©2018. *Notepad++ free source code editor*. [online]. [cit. 2018-02-08]. Dostupné z: http://notepad-plus-plus.org
- HW GROUP, 2010. *Nagios plug-ins for HWg devices.* [online]. [cit. 2018-03-23]. Dostupné z: https://www.hw-group.com/software/Nagios/Nagios_plugins_en.html
- *IETF Documents*, 2018. [online]. IETF Tools. [cit. 2017-09-14]. Dostupné z: https://tools.ietf.org/html/
- ISO, ©1994. *OSI Basic reference model* [online]. [cit. 2017-10-16]. Dostupné z: http://standards.iso.org/ittf/PubliclyAvailableStandards/index.html
- ITU, ©2018. *Global ICT developments.* Statistics. [online]. [cit. 2017-09-12]. Dostupné z: https://www.itu.int/en/ITU-D/Statistics/Pages/stat/default.aspx
- KABELOVÁ, Alena a Libor DOSTÁLEK, 2012. *Velký průvodce protokoly TCP/IP a systémem DNS*. 5., aktualiz. vyd. Brno: Computer Press. ISBN 978-80-251-2236-5.
- KAPOUN, Vladimír, 2000. *Digitální ústředny*. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a informatiky. ISBN 80-214-1731-5.
- KOCJAN, W, 2008. *Learning Nagios 3.0*. Birmingham: Packt Publishing. ISBN: 978-1-84719-518-0.
- KOZIEROK, Charles M, 2005. *The TCP/IP guide: a comprehensive, illustrated Internet protocols reference*. San Francisco: No Starch Press. ISBN 978-1-59327-047-6.

- KRETCHMAR, James M., DOSTÁLEK, Libor, 2004. *Administrace a diagnostika sítí: pomocí OpenSource utilit a nástrojů.* Brno: Computer Press. ISBN 80-251-0345-5.
- LEIBZON, William, 2014. *WL-NagiosPlugins.* [online]. [cit. 2018-03-25]. Dostupné z: https://github.com/willixix/WL-NagiosPlugins
- LINGE, Joerg, 2017. *PNP4Nagios Documentation.* [online]. GitHub. [cit. 2018-01-23]. Dostupné z: https://docs.pnp4nagios.org
- MATOUŠEK, Petr, 2014. *Síťové aplikace a jejich architektura.* Brno: VUTIUM. ISBN 978-80-214-3766-1.
- MAURO, Douglas R. a Kevin J. SCHMIDT, 2005. *Essential SNMP*. 2nd ed. Sebastopol, CA: O'Reilly. ISBN 978-0-596-00840-6.
- MEDIN, Michael, ©2015. *NSClient++*. [online]. [cit. 2018-01-05]. Dostupné z: https://nsclient.org/
- MPSV, ©2018. *Regionální statistika ceny práce.* [online]. [cit. 2018-03-23]. Dostupné z: http://portal.mpsv.cz/sz/stat/vydelky/download/2016/pra_164_mzs.pdf
- NAGIOS ENTERPRISES, ©2009-2018a. *Nagios Open Source.* [online]. [cit. 2018-01-05]. Dostupné z: https://www.nagios.org/
- NAGIOS ENTERPRISES, ©2009-2018b. *Nagios Documentation.* [online]. [cit. 2018-01-05]. Dostupné z: https://www.nagios.org/documentation
- NAGIOS ENTERPRISES, ©2009-2018c. *Nagios Exchange.* [online]. [cit. 2018-02-12]. Dostupné z: https://exchange.nagios.org
- NAGIOS ENTERPRISES, ©2009-2018d. *Nagios Plugins.* [online]. [cit. 2018-02-09]. Dostupné z: https://www.nagios.org/downloads/nagios-plugins/
- NAGVIS PROJECT, ©2008-2013. *NagVis Documentation.* [online]. [cit. 2018-02-10]. Dostupné z: http://www.nagvis.org/doc
- *Net-SNMP*, 2013. [online]. [cit. 2018-01-08]. http://www.net-snmp.org/
- ORANGE SA, ©2018. *Object Identifier Repozitory*. [online]. [cit. 2017-12-31]. Dostupné z: http://www.oid-info.com
- *Perl Programming Documentation*, 2017. [online]. [cit. 2018-03-09]. Dostupné z: http://www.perlmonks.org
- *PerlMonks*, 2018. [online]. [cit. 2018-03-08]. Dostupné z: http://www.perlmonks.org/
- PERL.ORG, ©2005-2018. *Comprehensive Perl Archive Network.* [online]. [cit. 2018-01-08]. Dostupné z: https://www.perl.org/cpan.html
- PROKEŠ, Josef, 2000. *Komunikace od kouřových signálů k Internetu aneb mám tě přečteného jako e-mail*. Zpravodaj ÚVT MU, roč. XI, č. 2, s. 15-16. ISSN 1212-0901.
- PŠENČÍKOVÁ, Jana, 2009. *Algoritmizace*. Vyd. 2. Kralice na Hané: Computer Media. ISBN 978-80-7402-034-6.
- RETIA, A.S., ©1993-2018. *ReDat záznamový systém*. [online]. [cit. 2018-02-15]. Dostupné z: https://www.redat.cz/cs/

- SANDERS, Chris, 2012. *Analýza sítí a řešení problémů v programu Wireshark*. Brno: Computer Press. ISBN 978-80-251-3718-5.
- SKINNER, Grant, nedatováno. *RegExr"*. [online]. [cit. 2018-03-09]. Dostupné z: https://regexr.com
- SLASHDOT MEDIA, ©2018. *SnmpB. SourceForge.* [online]. [cit. 2017-02-01]. Dostupné z: https://sourceforge.net/projects/snmpb/
- SOSINSKY, Barrie, 2010. *Mistrovství – počítačové sítě.* Brno: Computer Press. ISBN 978-80-251-3363-7.
- SPI, ©1997-2017. *Společenská smlouva s komunitou svobodného softwaru*. [online]. [cit. 2017-12-31]. Dostupné z: https://www.debian.org/social_contract
- STRACH, Jiří, 2009. *Media a komunikace.* [online]. [cit. 2018-01-10]. Dostupné z: http://boss.ped.muni.cz/vyuka/material/puvodni/Telecomunication.pps
- *Strawberry Perl*, 2017. [online]. The Perl for MS Windows [cit. 2018-02-09]. Dostupné z: http://strawberryperl.com
- SZABO, Gabor et al., 2016. *Padre the Perl IDE*. [online]. [cit. 2018-03-02]. Dostupné z: http://padre.perlide.org/
- VOZŇÁK, Miroslav, 2009. *Spojovací systémy.* Ostrava: VŠB – Technická univerzita Ostrava. ISBN 978-80-248-1961-7.

8 Přílohy

Příloha 1 - Plugin check_snmp_errortable

Příloha 2 - Plugin check_snmp_lastwritetime

Příloha 3 - Plugin check_snmp_time

Příloha 4 - NagVis mapa Ateco

Příloha 5 - NagVis mapa Ústředny

Příloha 6 - NagVis mapa ReDat

Plugin check_snmp_errortable

```
#!/usr/bin/perl -w

# Version:      1.4
# Date:         Mar 12, 2018
# Author:       ivan.stetka@live.com
# Summary:      This is a nagios plugin for check SNMP error table of Retia
#               recording equipment ReDat3, eXperience and VoIP Recorder
#               (RETIA-cz-MIB.mib)

use strict;
use Net::SNMP;
use Getopt::Long qw(:config no_ignore_case bundling);

my $ESOURCE = 3;
my $ESEVERITY = 4;
my $ETEXT = 5;

my %STATES = (
    OK              => 0,
    WARNING         => 1,
    CRITICAL        => 2,
    UNKNOWN         => 3
);

my %SEVERITYTEXT = (
    0       => 'OK',
    1       => 'Warning',
    2       => 'Error',
    3       => 'FatalError'
);

my %ESTATUS = (
    "R3"    => '1.3.6.1.4.1.17607.1.1.0',
    "REX"   => '1.3.6.1.4.1.17607.5.2.0',
    "RVR"   => '1.3.6.1.4.1.17607.3.2.0'
);

my %ETABLE = (
    "R3"    => '1.3.6.1.4.1.17607.1.8.1',
    "REX"   => '1.3.6.1.4.1.17607.5.1.1',
    "RVR"   => '1.3.6.1.4.1.17607.3.4.1'
);

my $eId = undef;
my $host = undef;
my $community = undef;
my $equipment = undef;
my $protocolV = 1;
my $mibFilePath = '/mibs/RETIA-cz-MIB.mib';
my $timeout = 15;
my $version = 0;
my $help = 0;
my $rowCounter = 0;
my $commaCounter = 0;
my $plural = '';
my %eTextValues = ();
my %eIdValues = ();
my %eSeverityValues = ();
my %eSourceValues = ();
my %longText = ();
```

```
61
62 GetOptions(
63     'H|host:s'          => \$host,
64     'C|community:s'     => \$community,
65     'E|equipment:s'     => \$equipment,
66     'P|protocolV:n'     => \$protocolV,
67     'm|mibFile:s'       => \$mibFilePath,
68     't|timeout:n'       => \$timeout,
69     'V|version'         => \$version,
70     'h|help'            => \$help
71 );
72
73 if ($help) {
74     help();
75     exit($STATES{OK});
76 }
77
78 if ($version) {
79     version();
80     exit($STATES{OK});
81 }
82
83 # timeout control
84 $SIG{ALRM} = sub {
85     print "Something is wrong, exceeded timeout parameter.\n";
86     usage();
87     exit($STATES{UNKNOWN});
88 };
89 alarm $timeout;
90
91 # input control
92 if (!defined $host || !defined $community || !defined $equipment) {
93     print "Missing required parameter (host, community, equipment).\n";
94     usage();
95     exit($STATES{UNKNOWN});
96 }
97
98 if (!defined ($ETABLE{$equipment} )) {
99     print "Bad format of the equipment '$equipment'.\n";
100    usage();
101    exit($STATES{UNKNOWN});
102 }
103
104 if ($protocolV != 1 && $protocolV != 2) {
105    print "Protocol version can be only 1 or 2.\n";
106    usage();
107    exit($STATES{UNKNOWN});
108 }
109
110 # load the MIB file
111 open FH, $mibFilePath or die("Can not open the MIB file.\n");
112 my %MIBIDINFO = ();
113 while (my $line = <FH>) {
114    if ($line =~ /e(\d+)-([a-zA-Z0-9-]+)/) {
115      $MIBIDINFO{$1} = $2;
116    }
117 }
118 close(FH);
119
120 # opening of the SNMP session
121 my ($session, $error) = Net::SNMP->session(
122    -hostname     => $host,
123    -community    => $community,
124    -version      => $protocolV
125 );
126
```

```
127 # check of the session
128 if (!defined $session) {
129    print "Missing session: $error\n";
130    badChoice();
131    exit($STATES{UNKNOWN});
132 }
133
134 # getting of the OID data (status) from the remote agent (reference to hash)
135 my @oids = ($ESTATUS{$equipment});
136 my $resultOid = $session->get_request(
137    -varbindlist => \@oids
138 );
139
140 # checking of the result (existence of the OID status)
141 if (!defined $resultOid) {
142    print "Error in fetching OID request: ", $session->error(), "\n";
143    badChoice();
144    $session->close();
145    exit($STATES{UNKNOWN});
146 }
147
148 # dereference of resultOid
149 my %hashOid = %{$resultOid};
150
151 # checking of the data existence in the return value
152 my $valueOid = $hashOid{$ESTATUS{$equipment}};
153 if (!defined $valueOid) {
154    print "OID '$ESTATUS{$equipment}' does not contain a value!\n";
155    badChoice();
156    $session->close();
157    exit($STATES{UNKNOWN});
158 }
159
160 # checking acceptable OID values
161 if ($valueOid !~ /^([0-3])$/) {
162    print "Bad OID value '$valueOid'.\n";
163    badChoice();
164    $session->close();
165    exit($STATES{UNKNOWN});
166 }
167
168 # retrieving the table error data from the remote agent (reference to hash)
169 my $result = $session->get_table(
170    -baseoid => ($ETABLE{$equipment})
171 );
172
173 # checking of the result (existence of the error table)
174 if (!defined $result) {
175    if ($session->error() gt 'The requested table is empty') {
176      checkOK();
177      } else {
178        print "Error in fetching table request: ", $session->error(), "\n";
179        badChoice();
180        $session->close();
181        exit($STATES{UNKNOWN});
182      }
183 }
184
185 # dereference of the result (error table)
186 my %values = %{$result};
187
```

```
188 # check if the table is empty
189 my @values = %values;
190 if (scalar @values == 0) {
191    checkOK();
192 }
193
194 # assign a column number eID by equipment
195 if ($equipment eq 'R3') {
196    $eId = 7;
197 } else {
198    $eId = 9;
199 }
200
201 # identification of rows and columns, putting into hashes
202 foreach my $key (keys %values ) {
203    my @nodes = split /\./, $key;
204    my $row = $nodes[-1];
205    my $column = $nodes[-2];
206    if ($column == $ETEXT) {
207      $eTextValues{$row} = $values{$key};
208      $rowCounter++;
209    } elsif ($column == $eId) {
210      $eIdValues{$row} = $values{$key};
211    } elsif ($column == $ESEVERITY) {
212      $eSeverityValues{$row} = $values{$key};
213    } elsif ($column == $ESOURCE) {
214      $eSourceValues{$row} = $values{$key};
215    }
216 }
217
218 # text output
219 if ($rowCounter > 1) {
220    $plural = "s";
221 }
222 print "WARNING: Status is '$SEVERITYTEXT{$valueOid}'"
223 . " and the table contains $rowCounter error$plural: ";
224 foreach my $key (sort keys %eTextValues) {
225    my $eIdValue = $eIdValues{$key};
226    my $eTextValue = $eTextValues{$key};
227    my $eSourceValue = $eSourceValues{$key};
228    my $eSeverityValue = $eSeverityValues{$key};
229    my $eInfo = "MIB: unknown Id error info";
230    $commaCounter++;
231    if (defined $MIBIDINFO{$eIdValue}) {
232      $eInfo = $MIBIDINFO{$eIdValue};
233    }
234    if ($equipment eq 'R3') {
235      print "$eTextValue - $eInfo";
236    } else {
237      print "$eSourceValue - $eInfo";
238    }
239    if ($commaCounter < $rowCounter) {
240      print ", ";
241    }
242    $longText{$key} = ("$eIdValue, $SEVERITYTEXT{$eSeverityValue}, "
243    . "$eSourceValue: $eTextValue, Info: $eInfo");
244 }
245
246 # long text output
247 print "\n";
248 foreach my $key (sort keys %longText){
249    print "$longText{$key}\n";
250 }
251
```

```
252 # perf data output (call subroutine perf)
253 perf();
254
255 # closing the session and returning Warning state
256 $session->close();
257 exit($STATES{WARNING});
258
259 # # # SUBROUTINES # # #
260
261 # Bad choice text
262 sub badChoice {
263    print "Is your choice of the equipment: '$equipment',"
264    . " SNMP protocol version: '$protocolV'"
265    . " or the host IP address: '$host' correct?\n";
266 }
267
268 # Check state OK
269 sub checkOK {
270    if ($valueOid eq '0') {
271       print "OK: No errors, $equipment status is OK.\n",
272       $session->error(), "\n";
273       perf();
274       $session->close();
275       exit($STATES{OK});
276    } else {
277       print "Status is not OK ("
278       . $SEVERITYTEXT{$valueOid}
279       . ") but the error table is empty or does not exist.\n";
280       badChoice();
281       $session->close();
282       exit($STATES{UNKNOWN});
283    }
284 }
285
286 # Perf data
287 sub perf {
288    print "|Errors=$rowCounter\n";
289 }
290
291 # Usage
292 sub usage {
293    print "Usage: $0 -H <host> -C <community> -E <equipment>"
294    . " [-P <protocol version>] [-m <MIB path>] [-t <timeout>]"
295    . "\nFor more information, use the -h (help).\n"
296 }
297
298 # Help
299 sub help {
300    print <<HELP;
301
302 This is a nagios plugin for check SNMP error table of Retia
303 recording equipment ReDat3, eXperience and VoIP Recorder.
304
305 Example ReDat3:
306 $0 -H 192.168.0.30 -C public -E R3 -P 1 -m /mibs/RETIA-cz-MIB.mib -t 5
307
308 Example REX:
309 $0 -H 192.168.0.40 -C public -E REX -P 1 -m /mibs/RETIA-cz-MIB.mib -t 5
310
311 -H, --host      # Hostname or IP - required value
312 -C, --community # SNMP community for read - required value
313 -E, --equipment # ReDat equipment - required value
314                  R3 - ReDat3
315                  REX - ReDat eXperience
316                  RVR - ReDat VoIP Recorder
```

```
317 -P, --protocolV # SNMP protocol version [default: 1]
318 -m, --mibFile   # Path to MIB [default: /mibs/RETIA-cz-MIB.mib]
319 -t, --timeout   # Timeout in seconds [default: 15]
320 -V, --version   # Print plugin version history
321 -h, --help      # Print this help
322 HELP
323 }
324
325 # Version history
326 sub version {
327    print "\n$0\n";
328    print <<VERSION;
329
330 version 1.4 (12.03.2018)
331 - check the value of the equipment variable changed
332 version 1.3 (09.03.2018)
333 - code cleanup
334 version 1.2 (27.02.2018)
335 - simplified long text printing
336 version 1.1 (23.02.2018)
337 - added checking Status equipment from OID, sub checkOK
338 version 1.0 (21.02.2018)
339 - changed arguments and checking the result
340 version 0.9 (20.02.2018)
341 - changed output text, long text and perf data
342 version 0.8 (18.09.2017)
343 - added sub usage, changed and added help text
344 version 0.7 (14.09.2017)
345 - changed help text, changed Version history, code cleanup
346 version 0.6 (23.08.2017)
347 - added variable erSource (optimalization for REX - ReDat eXperience)
348 version 0.5 (09.08.2017)
349 - changed name of variables equal retia error table, changed help text,
350 added Version history and ABOUT
351 version 0.4 (08.08.2017)
352 - added severity text
353 version 0.3 (20.07.2017)
354 - added variables severity and errorCounter, changed print text
355 version 0.2 (19.07.2017)
356 - fixed FH and values
357 version 0.1 (19.07.2017)
358 - new plugin for check SNMP error table of Retia logger ReDat3
359 VERSION
360 }
```

Plugin check_snmp_lastwritetime

```
1  #!/usr/bin/perl -w
2
3  # Version:      1.5
4  # Date:         Mar 13, 2018
5  # Author:       ivan.stetka@live.com
6  # Summary:      This is a nagios plugin for check SNMP OID time (last write to disk)
7  #               of Retia recording equipment ReDat3 and eXperience
8  #               (RETIA-cz-MIB.mib)
9  #               for time format:   "2000/01/31_23:59:59" - ReDat3
10 #                                  "2000-01-31 23:59:59" - eXperience
11
12 use strict;
13 use Net::SNMP;
14 use Getopt::Long qw(:config no_ignore_case bundling);
15 use Time::Piece;
16
17 my %STATES = (
18     OK             => 0,
19     WARNING        => 1,
20     CRITICAL       => 2,
21     UNKNOWN        => 3
22 );
23
24 my %TIMEOID = (
25     "R3"           => '1.3.6.1.4.1.17607.1.4.1.6.1',
26     "REX"          => '1.3.6.1.4.1.17607.5.4.1.4.4'
27 );
28
29 my %TIMEZONES = (
30     "UTC"          => "date -u ",
31     "UTC1"         => "date -u -d '+1 hour' ",
32     "UTC2"         => "date -u -d '+2 hour' ",
33     "CZ"           => "date "
34 );
35
36 my $host = undef;
37 my $community = undef;
38 my $equipment = undef;
39 my $protocolV = 1;
40 my $crit = 20;
41 my $warn = 5;
42 my $timeZone = 'UTC2';
43 my $timeout = 15;
44 my $version = 0;
45 my $help = 0;
46 my $oidTime = undef;
47 my $systemTime = undef;
48
49 GetOptions(
50     'H|host:s'          => \$host,
51     'C|community:s'     => \$community,
52     'E|equipment:s'     => \$equipment,
53     'P|protocolV:n'     => \$protocolV,
54     'w|warning:n'       => \$warn,
55     'c|critical:n'      => \$crit,
56     'z|timeZone:s'      => \$timeZone,
57     't|timeout:n'       => \$timeout,
58     'V|version'         => \$version,
59     'h|help'            => \$help
60 );
61
```

```
62 if ($help) {
63     help();
64     exit($STATES{OK});
65 }
66
67 if ($version) {
68     version();
69     exit($STATES{OK});
70 }
71
72 # timeout control
73 $SIG{ALRM} = sub {
74     print "Something is wrong, exceeded timeout parameter.\n";
75     usage();
76     exit($STATES{UNKNOWN});
77 };
78 alarm $timeout;
79
80 # input control
81 if (!defined $host || !defined $community || !defined $equipment) {
82     print "Missing required parameter (host, community, equipment).\n";
83     usage();
84     exit($STATES{UNKNOWN});
85 }
86
87 if (!defined ($TIMEOID{$equipment} )) {
88     print "Bad format of the equipment '$equipment'.\n";
89     usage();
90     exit($STATES{UNKNOWN});
91 }
92
93 if (!defined ($TIMEZONES{$timeZone} )) {
94     print "Bad format of time zone '$timeZone'.\n";
95     usage();
96     exit($STATES{UNKNOWN});
97 }
98
99 if ($protocolV != 1 && $protocolV != 2) {
100     print "Protocol version can be only 1 or 2.\n";
101     usage();
102     exit($STATES{UNKNOWN});
103 }
104
105 # opening of the SNMP session
106 my ($session, $error) = Net::SNMP->session(
107     -hostname     => $host,
108     -community    => $community,
109     -version      => $protocolV
110 );
111
112 # check of the session
113 if (!defined $session) {
114     print "Missing session: $error\n";
115     badChoice();
116     exit($STATES{UNKNOWN});
117 }
118
119 # getting of the OID data (time) from the remote agent (reference to hash)
120 my @oids = ($TIMEOID{$equipment});
121 my $resultOid = $session->get_request(
122     -varbindlist => \@oids
123 );
124
```

```
125 # checking of the result (existence of the OID status)
126 if (!defined $resultOid) {
127    print "Error in fetching OID request: ", $session->error(), "\n";
128    badChoice();
129    $session->close();
130    exit($STATES{UNKNOWN});
131 }
132
133 # dereference of resultOid
134 my %hashOid = %{$resultOid};
135
136 # checking of the data existence in the return value
137 my $valueOid = $hashOid{$TIMEOID{$equipment}};
138 if (!defined $valueOid) {
139    print "OID '$TIMEOID{$equipment}' does not contain a value!\n";
140    badChoice();
141    $session->close();
142    exit($STATES{UNKNOWN});
143 }
144
145 # parsing time from OID value
146 if ($valueOid =~ /^(\d{4})\/(\d{2})\/(\d{2})_(\d{2}):(\d{2}):(\d{2})$/) {
147    $oidTime = Time::Piece->strptime($valueOid, "%Y/%m/%d_%H:%M:%S");
148 } elsif ($valueOid =~ /^(\d{4})-(\d{2})-(\d{2}) (\d{2}):(\d{2}):(\d{2})$/) {
149    $oidTime = Time::Piece->strptime($valueOid, "%Y-%m-%d %H:%M:%S");
150 } else {
151    print "Wrong OID time format: '$valueOid'.\n";
152    badChoice();
153    $session->close();
154    exit($STATES{UNKNOWN});
155 }
156
157 # retrieving time from the console (Nagios-Linux)
158 open CH, "$TIMEZONES{$timeZone} '+%F %T' |";
159 my $consoleTime = <CH>;
160 close CH;
161 $consoleTime =~ s/\n//;
162
163 # parsing time from the console
164 if ($consoleTime =~ /^(\d{4})-(\d{2})-(\d{2}) (\d{2}):(\d{2}):(\d{2})/) {
165    $systemTime = Time::Piece->strptime($consoleTime,"%Y-%m-%d %H:%M:%S");
166 } else {
167    print "Wrong system time format: '$consoleTime'.\n";
168    $session->close();
169    exit($STATES{UNKNOWN});
170 }
171
172 # calculation of time differences
173 my $diffInS = $systemTime - $oidTime;
174 my $diffInM = int($diffInS / 60);
175 my $diffInH = int($diffInM / 60);
176 my $restInM = $diffInM - ($diffInH * 60);
177 my $restInS = $diffInS - ($diffInM * 60);
178
179 # check the system time delay
180 if ($diffInM < 0) {
181    print "System time error or wrong time zone " . $timeZone . "\n"
182    . "(system time '$systemTime' is delayed over OID time '$oidTime')\n";
183    $session->close();
184    exit($STATES{UNKNOWN});
185 }
186
```

```
187 # text output, long text and perf data
188 my $outputText = "last write to disc: " . $oidTime
189 . "\nWrite before: "
190 . sprintf("%02d:%02d:%02d", $diffInH, $restInM, $restInS)
191 . "\nSystem time: " . $systemTime
192 . "\nTime zone: " . $timeZone
193 . "|LastWrite=$diffInS" . "seconds;"
194 . ($warn * 60) . ";" . ($crit * 60) . "\n";
195
196 # evaluating states according to thresholds
197 if ($diffInM >= $crit) {
198    print "CRITICAL: " . $outputText;
199    $session->close();
200    exit($STATES{CRITICAL});
201 } elsif ($diffInM >= $warn) {
202    print "WARNING: " . $outputText;
203    $session->close();
204    exit($STATES{WARNING});
205 } else {
206    print "OK: " . $outputText;
207    $session->close();
208    exit($STATES{OK});
209 }
210
211 $session->close();
212 exit($STATES{UNKNOWN});
213
214 # # # SUBROUTINES # # #
215
216 # Bad choice text
217 sub badChoice {
218 print "Is your choice of the equipment: '$equipment',"
219 . " SNMP protocol version: '$protocolV'"
220 . " or the host IP address: '$host' correct?\n";
221 }
222
223 # Usage
224 sub usage {
225 print "Usage: $0 -H <host> -C <community> -E <equipment>"
226 . " [-P <protocol version>] [-w <warning>] -c [<critical>] [-z <timeZone>]"
227 . " [-t <timeout>] \n For more information, use the -h (help).\n"
228 }
229
230 # Help
231 sub help {
232    print <<HELP;
233
234 This is a nagios plugin for check SNMP OID time (last write to disk)
235 of Retia recording equipment ReDat3 and eXperience.
236
237 Example ReDat3:
238 $0 -H 192.168.0.30 -C public -E R3 -P 1 -w 20 -c 60 -z UTC1 -t 20
239
240 Example REX:
241 $0 -H 192.168.0.40 -C public -E REX -P 1 -w 20 -c 60 -z UTC1 -t 20
242
243 -H, --host      # Hostname or IP - required value
244 -C, --community # SNMP community for read - required value
245 -E, --equipment # ReDat equipment - required value
246                  R3 - ReDat3
247                  REX - ReDat eXperience
248 -P, --protocolV # SNMP protocol version - required value
249 -w, --warning   # warning threshold in minutes [default 5]
250 -c, --critical  # critical threshold in minutes [default 20]]
```

```
-z, --timeZone  # time zone [default UTC1]
                 UTC - Coordinated Universal Time
                 UTC1 - UTC +1h, CZ winter time, Central European Time
                 UTC2 - UTC +2h, CZ summer time, Central European Summer Time
                 CZ - respect CZ summer/winter time CET/CEST
-t, --timeout   # Timeout in seconds [default: 15]
-V, --version   # Print plugin version history
-h, --help      # Print this help
HELP
}

# Version history
sub version {
   print "\n$0\n";
   print <<VERSION;

version 1.5 (13.03.2018)
- check the value of the equipment and the timeZone variable changed
version 1.4 (09.03.2018)
- code cleanup
version 1.3 (2.3.2018)
- changed the way of inserting the OID, code cleanup
version 1.2 (23.01.2018)
- changed output text and PERFDATA
version 1.1 (19.09.2017)
- added sub usage, changed and added help text
version 1.0 (13.09.2017)
- added Archive-Version history and ABOUT, changed help and print text
- added input control timeZone, code cleanup
version 0.9 (30.08.2017)
- added timeZone, OID time format for ReDat REX add hash TIMEZONE
version 0.8 (30.08.2017)
- testing OID time formats
version 0.7 (30.08.2017)
- builds on the v 0.4 (no functions)
- added arguments warning, critical and case sensitive in Getopt::Long
version 0.6 (19.07.2017)
- only test v05 - value consoleTime (directly entered value)
version 0.5 (19.07.2017)
- added FUNCTIONS
version 0.4 (19.07.2017)
- module DateTime replaced Time::Piece
version 0.3 (19.07.2017)
- added options, module Time::Piece replaced DateTime
version 0.2 (19.07.2017)
- test system time from Nagios console, DateTime replaced Time::Piece
version 0.1 (18.07.2017)
- new plugin to check SNMP OID time for Retia logger ReDat3
- test system time from DOS, module DateTime
VERSION
}
```

Plugin check_snmp_time

```
1  #!/usr/bin/perl -w
2
3  # Version:      0.4
4  # Date:         Mar 9, 2018
5  # Author:       ivan.stetka@live.com
6  # Summary:      This is a nagios plugin to check SNMP OID time generally
7  #               (based on plugin check_snmp_lastwritetime.pl)
8  #               for time format:    "2000/01/31_23:59:59"
9  #                                   "2000-01-31 23:59:59"
10 #                                   "949363199" UNIX Epoch time format
11 #                                   "07d0011f173b3b00" Hex time format
12 #                                   (SNMP MIB HOST-RESOURCES-MIB DateAndTime format)
13
14 use strict;
15 use Net::SNMP;
16 use Getopt::Long qw(:config no_ignore_case bundling);
17 use Time::Piece;
18 use POSIX qw(strftime);
19
20 my %STATES = (
21     OK             => 0,
22     WARNING        => 1,
23     CRITICAL       => 2,
24     UNKNOWN        => 3
25 );
26
27 my %TIMEZONES = (
28     "UTC"  => "date -u ",
29     "UTC1" => "date -u -d '+1 hour' ",
30     "UTC2" => "date -u -d '+2 hour' ",
31     "CZ"   => "date ",
32 );
33
34 my $host = undef;
35 my $community = undef;
36 my $oid = undef;
37 my $protocolV = 1;
38 my $crit = 5;
39 my $warn = 2;
40 my $timeZone = 'UTC1';
41 my $timeout = 15;
42 my $labelEquipment = 'Equipment';
43 my $unit = 's';
44 my $version = 0;
45 my $archive = 0;
46 my $help = 0;
47 my $success = 0;
48 my $oidTime;
49 my $systemTime;
50 my $zoneText = "";
51 my $diffSign = undef;
52 my $diffText = undef;
53 my $diffInUnit = undef;
54
```

```
55 GetOptions(
56     'H|host:s'          => \$host,
57     'C|community:s'     => \$community,
58     'o|oid:s'           => \$oid,
59     'P|protocolV:n'     => \$protocolV,
60     'w|warning:n'       => \$warn,
61     'c|critical:n'      => \$crit,
62     'z|timeZone:s'      => \$timeZone,
63     't|timeout:n'       => \$timeout,
64     'l|labelE:s'        => \$labelEquipment,
65     'U|unit:s'          => \$unit,
66     'V|version'         => \$version,
67     'a|archive'         => \$archive,
68     'h|help'            => \$help
69 );
70
71 if ($help) {
72     help();
73     exit($STATES{OK});
74 }
75
76 if ($version) {
77     version();
78     exit($STATES{OK});
79 }
80
81 # input control
82 if (!defined ($TIMEZONES{$timeZone} )) {
83     print "Bad format of time zone '$timeZone'.\n";
84     usage();
85     exit($STATES{UNKNOWN});
86 }
87
88 if (!defined $host || !defined $community || !defined $oid) {
89     print "Missing required parameter (host, community, oid).\n";
90     usage();
91     print "Check -h (help) for more information.\n";
92     exit($STATES{UNKNOWN});
93 }
94
95 chomp($oid);
96 if ($oid !~ /^\d+(\.\d+)*$/) {
97     print "Bad format of OID number '$oid'.\n";
98     usage();
99     exit($STATES{UNKNOWN});
100 }
101
102 if (defined $protocolV) {
103    if ($protocolV != 1 && $protocolV != 2) {
104       print "SNMP protocol version can be only 1 or 2.\n";
105       exit($STATES{UNKNOWN});
106    }
107 }
108
109 # timeout control
110 $SIG{ALRM} = sub {
111    print "Something is wrong, exceeded timeout parameter.\n";
112    usage();
113    exit($STATES{UNKNOWN});
114 };
115 alarm $timeout;
116
```

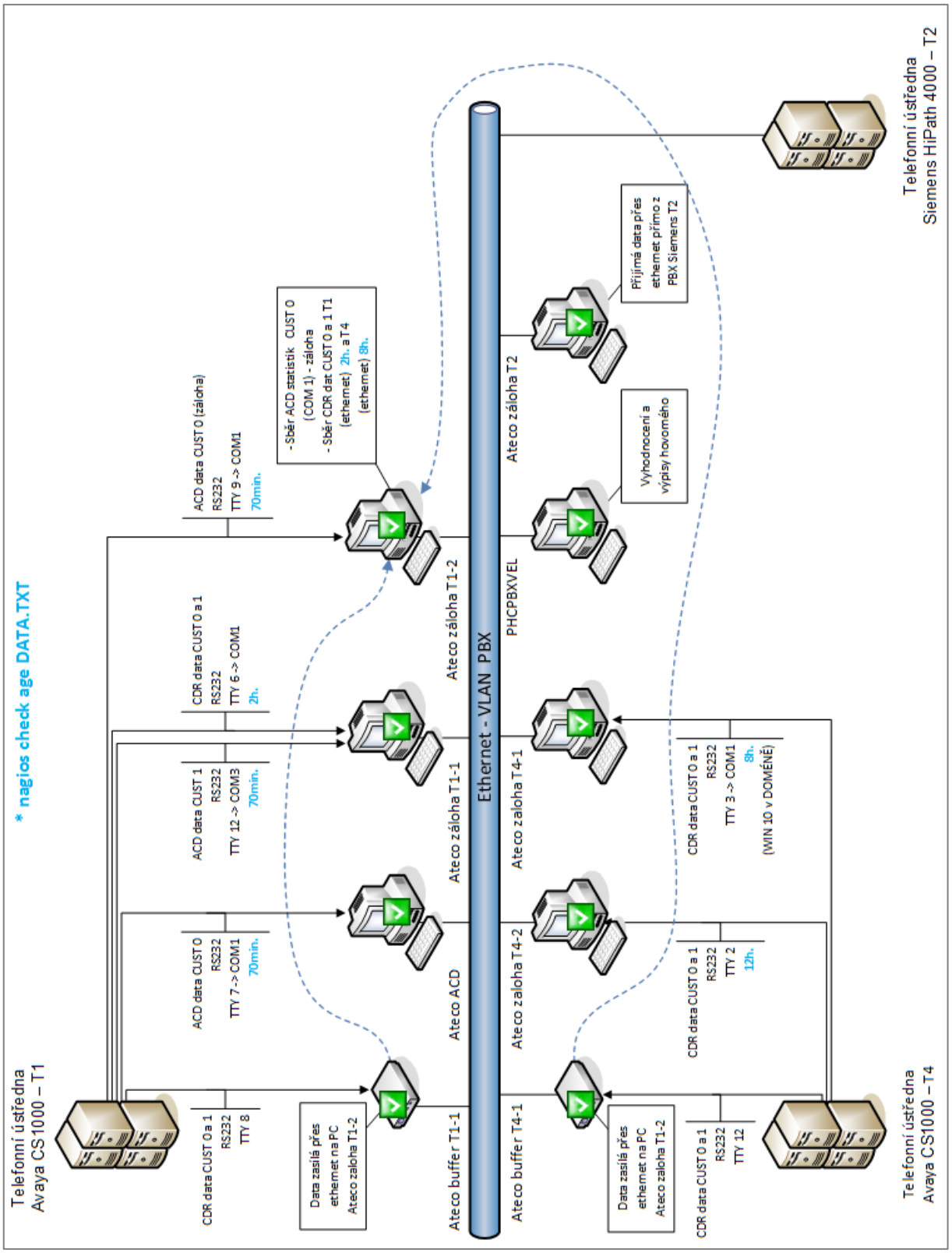
```
117 # opening of the SNMP session
118 my ($session, $error) = Net::SNMP->session(
119    -hostname     => $host,
120    -community    => $community,
121    -version      => $protocolV
122 );
123
124 # check of the session
125 if (!defined $session) {
126    print "Missing session: $error\n";
127    exit($STATES{UNKNOWN});
128 }
129
130 # getting of the OID data (status) from the remote agent (reference to hash)
131 my @oids = ($oid);
132 my $result = $session->get_request(
133    -varbindlist => \@oids
134 );
135
136 # checking of the result (existence of the OID status)
137 if (!defined $result) {
138    print "Error in fetching OID request: ", $session->error(), "\n";
139    badChoice();
140    exit($STATES{UNKNOWN});
141 }
142
143 # dereference of result
144 my %hash = %{$result};
145
146 # checking of the data existence in the return value
147 if (!exists $hash{$oid}) {
148    print "OID '$oid' does not contain a value!\n";
149    badChoice();
150    $session->close();
151    exit($STATES{CRITICAL});
152 }
153 my $valueOid = $hash{$oid};
154
155 # parsing time from OID value
156 if ($valueOid =~ /^(\d{4})\/(\d{2})\/(\d{2})_(\d{2}):(\d{2}):(\d{2})$/) {
157    $oidTime = Time::Piece->strptime($valueOid, "%Y/%m/%d_%H:%M:%S");
158 } elsif ($valueOid =~ /^(\d{4})-(\d{2})-(\d{2}) (\d{2}):(\d{2}):(\d{2})$/) {
159    $oidTime = Time::Piece->strptime($valueOid, "%Y-%m-%d %H:%M:%S");
160 } elsif ($valueOid =~ /^(\d{9,10})$/) {
161    my $oidEpochTime = strftime("%Y-%m-%d %H:%M:%S", localtime($valueOid));
162    $oidTime = Time::Piece->strptime($oidEpochTime, "%Y-%m-%d %H:%M:%S");
163 } elsif ($valueOid =~ /^0x((([a-f]|[0-9]){2}){8})$/) {
164    $valueOid =~ s/^.{2}//s;
165    my @hexTime = unpack 'n C6 a C2',(pack "H*", $valueOid);
166    my $oidHexTime = (sprintf "%04d-%02d-%02d %02d:%02d:%02d", @hexTime);
167    $oidTime = Time::Piece->strptime($oidHexTime, "%Y-%m-%d %H:%M:%S");
168 } else {
169    print "Wrong OID time format: '$valueOid'.\n";
170    $session->close();
171    exit($STATES{UNKNOWN});
172 }
173
174 # retrieving time from the console (Nagios-Linux)
175 open CH, "$TIMEZONES{$timeZone} '+%F %T' |";
176 my $consoleTime = <CH>;
177 close CH;
178 $consoleTime =~ s/\n//;
179
```

```
180 # parsing time from the console
181 if ($consoleTime =~ /^(\d{4})-(\d{2})-(\d{2}) (\d{2}):(\d{2}):(\d{2})/) {
182    $systemTime = Time::Piece->strptime($consoleTime, "%Y-%m-%d %H:%M:%S");
183 } else {
184    print "Wrong System time format: '$consoleTime'.\n";
185    $session->close();
186    exit($STATES{UNKNOWN});
187 }
188
189 # calculation of time differences
190 my $diffInS = $oidTime - $systemTime;
191 my $diffInM = (int($diffInS / 60));
192 my $diffInAbsM = abs($diffInM);
193 my $diffInAbsH = (int($diffInAbsM / 60));
194 my $restInAbsM = $diffInAbsM - ($diffInAbsH * 60);
195 my $restInAbsS = abs($diffInS) - ($diffInAbsM * 60);
196
197 # checking whether the time difference is exactly 1 or 2 hours
198 if ($diffInAbsM == 60 || $diffInAbsM == 120) {
199    $zoneText = ", maybe wrong time zone " . $timeZone;
200 }
201
202 # evaluation of the time difference type
203 if ($diffInS < 0 ) {
204    $diffSign = " delayed";
205 } elsif ($diffInS > 0 ) {
206    $diffSign = " overflowed";
207 } else {
208    $diffSign = " the same";
209 }
210
211 # preparation diff output text
212 if ($diffInS == 0) {
213    $diffText = " as System time.";
214 } else {
215    $diffText = ". Difference is "
216    . sprintf("%02d:%02d:%02d", $diffInAbsH, $restInAbsM, $restInAbsS);
217 }
218
219 # check the unit of threshold and the performance data
220 if ($unit eq 'm') {
221    $diffInUnit = $diffInM;
222 } elsif ($unit eq 's') {
223    $diffInUnit = $diffInS;
224 } else {
225    print "Wrong threshold unit\n";
226    usage();
227    print "Check -h (help) for more information.\n";
228    $session->close();
229    exit($STATES{UNKNOWN});
230 }
231
232 # text output, long text and perf data
233 my $outputText = $labelEquipment
234 . " time is" . $diffSign . $zoneText . $diffText
235 . "\n$labelEquipment time: $oidTime"
236 . "\nSystem time: $systemTime"
237 . "\nTime zone: $timeZone"
238 . "|Difference=$diffInUnit" . $unit . ";" . $warn . ";" . $crit . "\n";
239
```

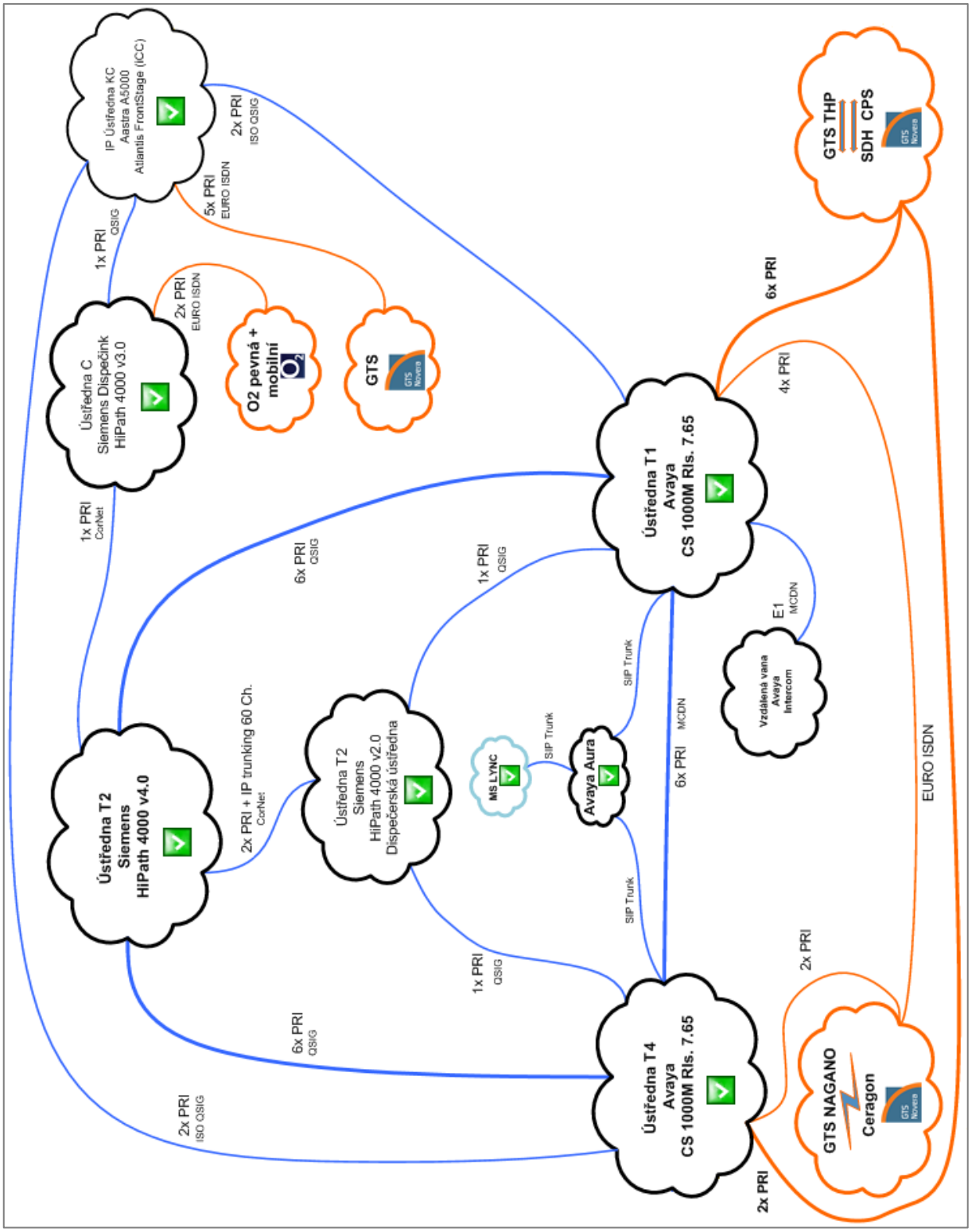
```
240 # evaluating states according to thresholds
241 if (abs($diffInUnit) >= $crit) {
242    print "CRITICAL: " . $outputText;
243    $session->close();
244    exit($STATES{CRITICAL});
245 } elsif (abs($diffInUnit) >= $warn) {
246    print "WARNING: " . $outputText;
247    $session->close();
248    exit($STATES{WARNING});
249 } else {
250    print "OK: " . $outputText;
251    $session->close();
252    exit($STATES{OK});
253 }
254
255 $session->close();
256 exit($STATES{UNKNOWN});
257
258 # # # SUBROUTINES # # #
259
260 # Bad choice text
261 sub badChoice {
262    print "Is your choice of the OID: '$oid',"
263    . " SNMP protocol version: '$protocolV'"
264    . " or the host IP address: '$host' correct?\n";
265 }
266
267 # Usage
268 sub usage {
269    print "Usage: $0 -H <host> -C <community> -o <oid>"
270    . " [-P <protocol version>] [-w <warning>] [-c <critical>]"
271    . " [-z <timeZone>] [-t <timeout>] [-l <label>] [-U <unit>]"
272    . "\nFor more information, use the -h (help).\n";
273 }
274
275 # Help
276 sub help {
277    print <<HELP;
278
279 This is a nagios plugin to check SNMP OID time generally
280 based on plugin check_snmp_lastwritetime.pl)
281 for time format:      "2000/01/31_23:59:59"
282                       "2000-01-31_23:59:59"
283                       "949363199" UNIX Epoch time format
284                       "07d0011f173b3b00" Hex time format
285                       (SNMP MIB HOST-RESOURCES-MIB DateAndTime format)
286
287 Example Mobatime
288 (mobatime.mobaNetClocks.mobaNetClocksV1.mbnscTime.mbnscTimeDeviceTime.0):
289 perl $0 -H 192.168.0.50 -C public -P 2 -o 1.3.6.1.4.1.13842.6.1.2.7.0
290 -w 20 -c 60 -z UTC1 -t 15 -l "Mobatime 1" -U s
291
292 -H, --host      # Hostname or IP - required value
293 -C, --community # SNMP community for read - required value
294 -P, --protocolV # SNMP protocol version - required value
295 -o, --oid       # OID - required value
296 -w, --warning   # warning threshold in unit -U [default 2]
297 -c, --critical  # critical threshold in unit -U [default 5]
```

```
298 -z, --timeZone  # time zone [default UTC1]
299                  UTC - Coordinated Universal Time
300                  UTC1 - UTC +1h, CZ winter time, Central European Time
301                  UTC2 - UTC +2h, CZ summer time, Central European Summer Time
302                  CZ - respect CZ summer/winter time CET/CEST
303 -t, --timeout   # Timeout in seconds [default 15]
304 -l, --labelE    # Equipment label [Equipment]
305 -U, --unit      # Threshold and Performance data unit [default s]
306                  s - seconds
307                  m - minutes
308 -V, --version   # Print plugin version history
309 -h, --help      # Print this help
310 HELP
311 }
312
313 # Version history
314 sub version {
315    print "\n$0\n";
316    print <<VERSION;
317
318 Plugin to check SNMP OID time, for time format:
319 "2000/01/31_23:59:59", "2000-01-31 23:59:59",
320 Epoch "949363199" and Hex "07d0011f173b3b00"
321
322 version 0.4 (09.03.2018)
323 - code cleanup
324 version 0.3 (14.02.2018)
325 - added Hex time format
326 version 0.2 (05.02.2018)
327 - fixed diff variables, added ability to change Threshold and Perfdata units
328 version 0.1 (24.01.2018)
329 - new plugin to check SNMP OID time based on plugin check_snmp_lastwritetime.pl
330 - added epoch time format, equipment label and delayed/overflowed check
331 VERSION
332 }
```

Nagvis mapa Ateco



Nagvis mapa Ústředny



Nagvis mapa ReDat