Service Tags Functional Specification

1 Project Description

Sun Service Tags is a simple tagging scheme, and associated network application(s), designed to commonly identify Sun products and make available to network querying applications the stored common identifiers. The "service tag" identifiers are used to register Sun products back to Sun and by the customer to create and maintain an inventory of their Sun assets.

1.1 Definition

See section 2.

1.2 Motivation, Goals, and Requirements

The foundation problem is that Sun cannot clearly measure its actual install base of products. We can measure downloads, and we can measure things like "paid for subscriptions" or "purchases support contracts", but when attempting to represent those numbers as a "penetration" measurement we lack the base numbers of actual installs to work from. Service Tags attempts to provide the common representation of an installed product and the means to "harvest" that information to ship back to Sun.

The ability to feed business intelligence reporting is being driven by Sun's increasing track service monetization and penetration of Sun's (free) software. Sun needs to have product registration and tracking to ensure the model is working. With the product information provided by Service Tags, Sun will be able to our "free" install base more cleanly. Service Tags will generate indirect revenue benefits. Improved product business intelligence enables Sun to make more informed product investment decisions as well as increases the ability to market to users installing and running Sun product.

1.3 Changes From the Previous Release

N/A. This is a new product.

1.4 Program Plan Overview

1.5 Related Projects

1.5.1 Dependencies on Other Sun Projects

None

1.5.2 Dependencies on Non-Sun Projects

None

1.5.3 Sun Projects Depending on this Project

Registry Client project – ARC Case # 699.

Anza Project in SysNet. This project is being incepted at this time. The scope of Anza will include the Registry Client mentioned above as well as the Sun back end Web Service and Reporting and Portal Services.

1.5.4 Projects Rendered Obsolete by this Project

None

1.5.5 Related Active Projects [Describe the relationship.]

None

1.5.6 Suggested Projects to Enhance this Program

1.6 Competitive Analysis

Network discovery of assets is a well established capability in the Asset Management software market. What differentiates Service Tags from the Asset Management network discovery capabilities is that Sun is embedding Network Discovery in the the OS and Software Stack in addition to Systems and Storage. Also, by making Service Tags a tool that the customer will want to use, rather than a tool they are mandated to use to achieve license compliance (a la Microsoft), Sun will be able to provide a value add to customers while gathering the benefit of the product information contained in Service Tags.

2 Technical Description

2.1 Architecture

Service Tags is composed of both client and server side components that persist, discover and transport service tag records from customer systems to Sun. The whole ST architecture is shown in Fig. 1. A client-side component interaction diagram is

shown in Fig. 2.

The Service Tags product has the ability to utilize alternate service discovery protocols depending on the customer's preferences and environment. One such discovery protocol is the Service Location Protocol (SLP). Service Location Protocol is an IETF standards track protocol that provides a framework to allow networking applications to discover the existence, location, and configuration of networked services in enterprise networks.

Another service discovery protocol that can be used with Service Tags is the DNS-SD (Bonjour) technology being integrated into Solaris Nevada. By advertising the existence of Service Tag Listeners via the link-ThankThalocal DNS-SD infrastructure, service tag information can quickly be located and collected without the need for a centralized service directory.

The client side components of Service Tags (ST) is shown in the diagram below and are as follows:

- ST Registry
- ST Helper
- ST Listener
- ST Discoverer

2.1.1 ST Registry

The registry is an XML-based local repository containing a list of the product instances installed in the system. Service tags are created by Sun product teams when their products are installed in the system. Each service tag has a unique instance identifier which is generated or supplied by the product team when the record is added to the registry. This identifier is used as the key when updating the record.

The registry is created when the Service Tag software is installed in the system. It is owned by the user noaccess with the group set to noaccess. This permission on the file is set to 664 (rw-rw-r--) and is persisted in /var/sadm/servicetag/registry/servicetag.xml.

2.1.2 ST Helper

The ST Helper provides the programmatic interface to the registry. Using the Helper, service tag records can be added, changed, deleted and extracted from the registry. There are 2 interfaces that the Helper provides: Command Line Interface (CLI) and the C API.

The ST Helper CLI, stclient, is a public interface and intended to be used by internal Sun product teams. The CLI binary is owned by the user noaccess and runs with an owner setuid. It was written using getopt instead of getopt_long to parse the options to ensure compatibility with older O/S such as Solaris 8.

The C API is a project private interface to be used by the Service Tag components. To interface with the XML-based registry, the open software library, libxml, was used. This library was chosen because it is distributed in different Solaris releases starting from Solaris 8. It is also available in other platforms such as Linux and Windows which makes the porting of the Service Tags into these platforms feasible.

2.1.3 ST Listener

The ST Listener makes the contents of a system's ST Registry visible on a local network. It listens on a configurable port for ST messages. The ST Listener communicates via REST-style HTTP messaging. The messages and protocol are further detailed in stlisten_protocol.pdf.

2.1.4 ST Discoverer

The ST Discoverer supports discovery of the ST Listener network application. It is a lightweight built-in UDP-based discovery protocol which can be supplanted by various other discovery protocols such as Service Location Protocol (SLP), and DNS Service Discovery (DNS-SD). The messages and protocol are further detailed in stdiscover_protocol.pdf.

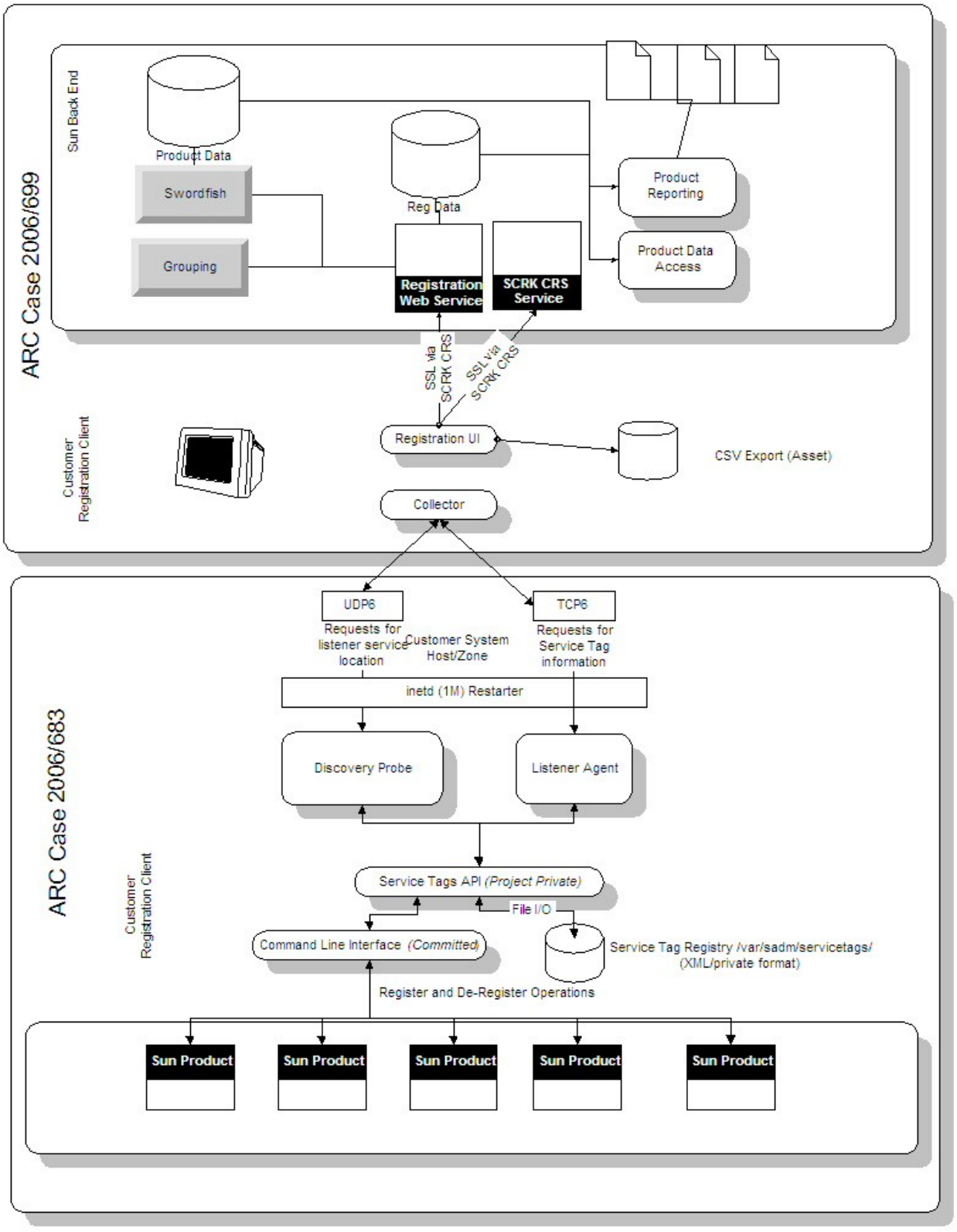

Fig. 1 - Service Tags Architecture

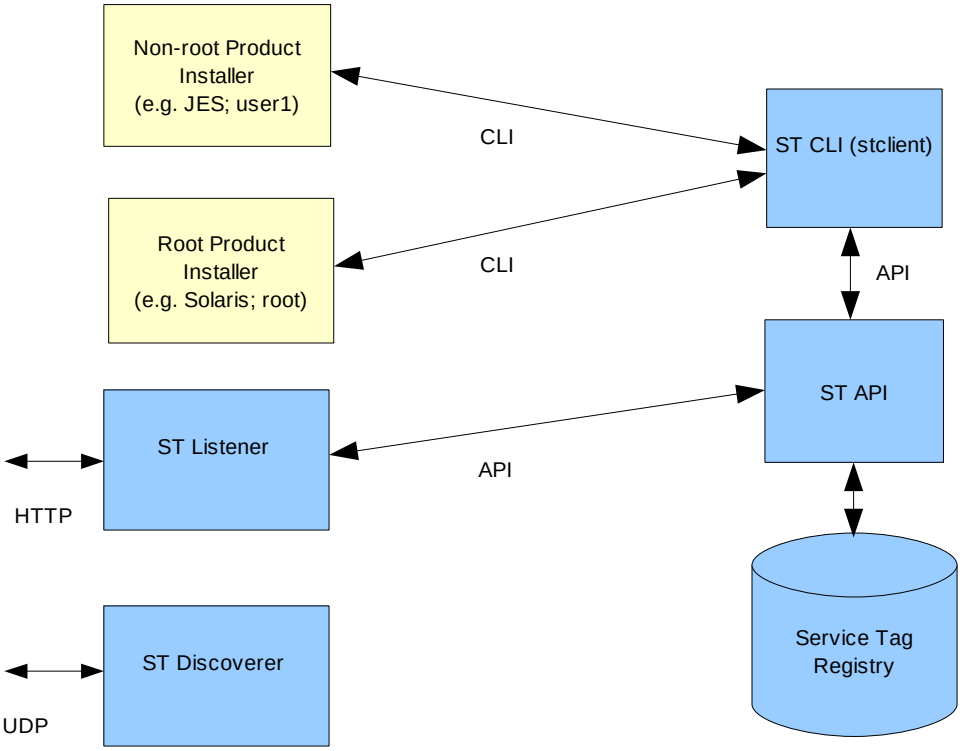

Fig. 2 - ST Client-Side Component Interactions

2.2 Interfaces

2.2.1 Exported Interface

All these components are included in the SUNWstr package.

Interface Name	Proposed Stability Classification	Specified in What Document?	Former Stability Classification or Other Comments
ST CLI	Committed	ServiceTag_API_CLI_v06.pdf	Volatile

ST C API	Project Private	ServiceTag_API_CLI_v06.pdf	Project Private
ST Registry	Committed	ServiceTag_API_CLI_v06.pdf	Volatile
ST Discover Protocol	Project Private	stdiscover_protocol.pdf in.stdiscover.manpage.pdf	Project Private
ST Listen Protocol	Project Private	stlisten_protocol.pdf in.stlisten.manpage.pdf	Project Private

2.2.2 Imported Interface

Interface Name	Proposed Stability Classification	Specified in What Document?	Former Stability Classification or Other Comments
PSARC/2001/175/libxml	External	http://www.xmlsoft.org/html/index.html	External
swoRDFish	Evolving	http://sac.sfbay/SARC/2003/179/	External
SLP – Service Location Protocol	Stable	http://sac.sfbay/PSARC/1997/307/	
DNS-SD – Multicast DNS and Service Discovery	Committed	http://sac.sfbay.sun.com/Archives/CaseLog/arc/PSARC/2005/562/	

2.3 User Interface

Service Tags client-side component supports one Command Line Interface : ST Helper CLI. The ST Helper CLI is used to add, change, delete or extract service tag records from the registry. It is intended to be used only by Sun product teams to add/remove service tags from the registry during the installation and/or deinstallation of their product.

2.4 Compatibility and Interoperability

- Service Tags are being tested against Solaris 8, 9, 10GA, 10U3 and S11 B55.
- In future releases, Service Tags will be interoperable with Windows, Linux and HP-UX.

2.4.1 Standards Conformance

- No, we are not implementing any significant standards.
- No, we are not deviating from or extending any standards.
- There are no relevant standards to which we do not conform.
- ST supports IPv6.

2.4.2 Operating System and Platform Compatibility

For this release, Solaris 8 and higher will be supported. There are no hardware dependencies.

The software dependencies are libxml library (2.4.7 and higher).

2.4.3 Interoperability with Sun Projects/Products

- Solaris 10 Update 4 integration will be first interoperability focus, Eventually, all Sun products (Software, Systems and Storage) will leverage and inter-operate with Service Tags for discovery and registration

2.4.4 Interoperability with External Products

- No.

2.4.5 Coexistence with Similar Functionality

- No other overlapping functionality exists at Sun.

2.4.6 Support for Multiple Concurrent Instances

- Yes

2.4.7 Compatibility with Earlier and Future Releases

N/A. This is a new release.

2.5 Performance and Scalability

2.5.1 Performance Goals

No operation which is local to the current machine (such as querying or updating the ST Registry via CLI or API) should take more than 1 second. No operation which is local to the current LAN (such as, in most cases, the ST Discover and communication with ST Listener) should take more than 30 seconds. No operation which required WAN communication should require more than 2 minutes.

Note that all of the above figures assume well-performing infrastructure (such as a LAN with reasonably-sized broadcast domains and relatively low latency). In cases where underlying infrastructure problems exist, the performance of various operations related to service tags will undoubtably suffer.

2.5.2 Performance Measurement

2.5.3 Scalability Limits and Potential Bottleneck

Registry is bounded by libxml.

2.5.4 Static System Behavior

- No large files or database connectivity required.
- Solaris 10 - SPARC
  144K total filesize(s) on disk
  4696K in memory size

  Solaris 10 - X86
  147K total filesize(s) on disk
  3564K in memory size

2.5.5 Dynamic System Behavior

- The system does not wake up periodically. When running under inetd, both the Listener and Discoverer wake up when inetd wakes them up. When running in daemon mode, these processes are always running. Registry leverages libxml which manages working set and multi-threaded

2.6 Failure and Recovery

2.6.1 Resource Exhaustion

The Helper CLI and API will gracefully exit and generate the appropriate message when a resource exhaustion, e.g. virtual memory, is encountered.

The Listener will return appropriate HTTP error codes (such as a “503: Service Unavailable”) if it encounters a resource exhaustion which prevents the handling of a request. If it encounters a resource exhaustion at startup time which prevents it from initializing and running, it will exist with a process-level error code and message.

2.6.2 Software Failures

The Helper CLI and API will fail if the registry is corrupted. This corruption is possible if the registry was manually edited by an authorized user. In this situation, the registry must be deleted and recreated.

ST Listener and Discovery can be impacted by Registry crashing.

- User interupts are handled gracefully. Testing against Process Kills and Reboots have failed to corrupt registry in testing.

2.6.3 Network Failures

None,

2.6.4 Data Integrity

The Helper CLI and API checks the registry against a DTD to ensure that the integrity of the registry before any read/write operation is done.

2.6.5 State and Checkpointing

None.

2.6.6 Fault Detection

The Helper CLI returns a non-zero return code in the case of a failure. This return code can be checked by the calling script. In addition to this, an error message is also displayed. The Helper API also returns 0 when the call to the function was successfully otherwise it returns a non-zero code. Using the st_error function, the error code can be translated to a human readable message.

2.6.7 Fault Recovery (or Cleanup after Failure)

For the Helper CLI and API, fault recovery depends on the failing component. In the case of a corrupted registry, the recovery option would be to delete and recreate the registry. Components would have to be added back to the registry. In the case of environment failures such as insufficient memory, the Helper CLI and API can be restarted after the problem is corrected. For failures caused by the users of the CLI and API such as in the case of invalid parameters, the Helper CLI and API can also be restarted after the correct parameters are provided.

• *2.7 Security*

The ST Registry is secured by UNIX's file protection scheme. It is owned by the user noaccess with a file permission set to 664 (rw-rw-r--) and group set to noaccess. The registry can only be publicly updated using the ST CLI. The CLI executable is owned by the user noaccess with the setuid set to owner. This enables non-root users to access the registry. The ST interfaces, CLI and API, provide record level security. Only root and the creator of the service tag can update or delete a record.

The ST Listener and Discoverer both allow for the specification of a UNIX user as whom it should run. In the default (and recommended) case, it will run as user noaccess. This aligns with the permissions required for updating the ST Registry. If an administrator wishes to run these processes with other permissions, it is his responsibility to ensure appropriate access to the ST Registry.

Note that in the default usage, there is little or no authentication or authorization support protecting the communications (and resulting ST Registry reads and writes) between the ST Listener and future ST clients. This is because a core goal of the Service Tag system is to support registrations with essentially no up-front bootstrapping work. Any likely scheme for authentication and authorization at this level would require such bootstrapping. Future work should implement an alternative configuration such that administrations who wish to take on such a bootstrapping effort to improve local security may do so.

After examining the default installation combined with the fact that we are running "default on", we realized that there IS a potential risk with the configuration in a situation where the customer network is not actively blocking TCP ports on their internal network and/or at the perimeter (The "always-off" concept of router deployment). The ST Listener process could be hit from across multiple network segment, including even into the Internet, if the routers allowed it. This was counter to our default deployment design where we presumed the existence of controlling routers (a bad assumption). We modified the ST Listener process to support command line (and default installed as such) operation in "local network only" traffic mode. TCP packets will not leave the immediately attached network by default.
The customer has the option to manually modify the configuration to allow network

routed behavior - but that requires them to make a conscious choice to do so. They also can completely disable all (and each individually) service tag processes/steps if they so choose.

PSARC raised a question about why we are not encrypting communication between the collector/harvester and the ST Listener daemon/process and the security risks that poses. Customer input requesting/requiring all information gathering to behave in the clear (as they are used to SNMPv1 behaving) was what drove that design. We recognize that not all customers necessarily take that position, but a significant number did. We provide a number of mechanisms for the customer to control the Service Tag operation, up to and including completely disabling it. The customer is also not required to run the collector - it is an optional registration tool.

2.8 Software Engineering and Usability

2.8.1 Namespace Management

2.8.2 Dependencies on non-Standard System Interfaces

- None.

2.8.3 Year 2000 Compliance

The date/time stamp stored in the registry uses the 4 digit year format, specifically the format used is YYYY-MM-DD HH24:MI:SS GMT.

2.8.4 Internationalization (I18N)

- i18n is not required for Solaris integration as there is no customer interface. This was determined by Solaris Localization team (Young J. Sun)

I18n Level	*Meaning*
"Level 0"	Are images trivially replaceable without source code? Cultural differences demand it.
Level 1	"8-bit clean" to support ISO Latin-1 (European) codesets. This includes data paths and messages.
Level 2	Formatting (date, time, currency, numbers) are locale-sensitive.
Level 3	User-visible text trivially translated in message catalogs. Localization

	(L10n) should be possible without algorithmic source code.
Level 4	Asian language support--via Extended Unix Coding (EUC)? Are data paths capable of handling wide Asian characters? Code Set Independence (CSI) enabled? (See Solaris 2.6's `attributes(5)` manpage.)

2.8.5 64-bit Issues

64-bit clean

2.8.6 Porting to other Platforms

Porting to other platforms is anticipated. For common codes such as the Helper. it would involved recompiling and testing using the platform specific version of the libxml library. For platform specific codes, the code will be developed for the target platform. In the case of the Listener, some effort will be required to determine each platform's appropriate mechanism[s] for obtaining dynamic information (such as the “sysinfo” mechanism on Solaris).

2.8.7 Accessibility

Yes, see included service_tag_508.pdf.

3 Release Information

3.1 Product Packaging

This product is delivered in a bundled package.

3.1.1 Package Overview

Name: SUNWstr

Default Installation Root: /usr

Required.

3.1.2 (Default) Installation Locations

Configuration Files:

None

Libraries:

$ROOT/lib

Temporary files:

None

DLLs:

None

Java Packages:

None

3.1.3 Effect on External Environment

3.2 Installation

3.2.1 Installation procedure

Follows standard Solaris packaging: pkgadd SUNWstr .

3.2.2 Effects on System Files

Updates the /etc/services file to include the entry:
servicetag 4950/udp # Servicetag daemon

On Solaris 8 & 9, updates /etc/inetd.conf to include an entry for the service tag Listener and Discoverer. No other changes are made to existing system files.

3.2.3 Boot-Time Requirements

For Solaris 10 and above, start the Listener and Discoverer via SMF.

For Solaris 8 & 9, inserts an entry into inetd.

3.2.4 Licensing

3.2.5 Upgrade

This is not application since this is the first release.

3.2.6 Software Removal

Follows standard Solaris packaging: pkgrm SUNWstr .

3.3 System Administration

- The only administration allowed is for configuration of Discovery protocol (UDP, SLP, etc.) This is accomplished via command line switch flags.

4 Component Name Architecture

4.1 Description

See section 2.

4.2 Interfaces

4.2.1 User-visible

- No

4.2.2 Internal (optional for ARC review)

4.3 Operation

Appendix A: Standards Supported

References

R.1 Related Projects

Registration Client – ARC Case # 2006/699

R.2 Background Information for this Project or its Product

R.3 Interface Specifications

1. Service Tags Design Specifications -
    1. Service Tags CLI and API Design Specification
        1. ServiceTag_API_CLI_v06.pdf
    2. Service Tags Listener (Man page and protocol specification):
        1. in.stlisten.manpage.pdf
        2. stlisten_protocol.pdf
    3. Service Tags Discoverer (Man page and protocol specification):
        1. in.stdiscover.manpage.pdf
        2. stdiscover_protocol.pdf