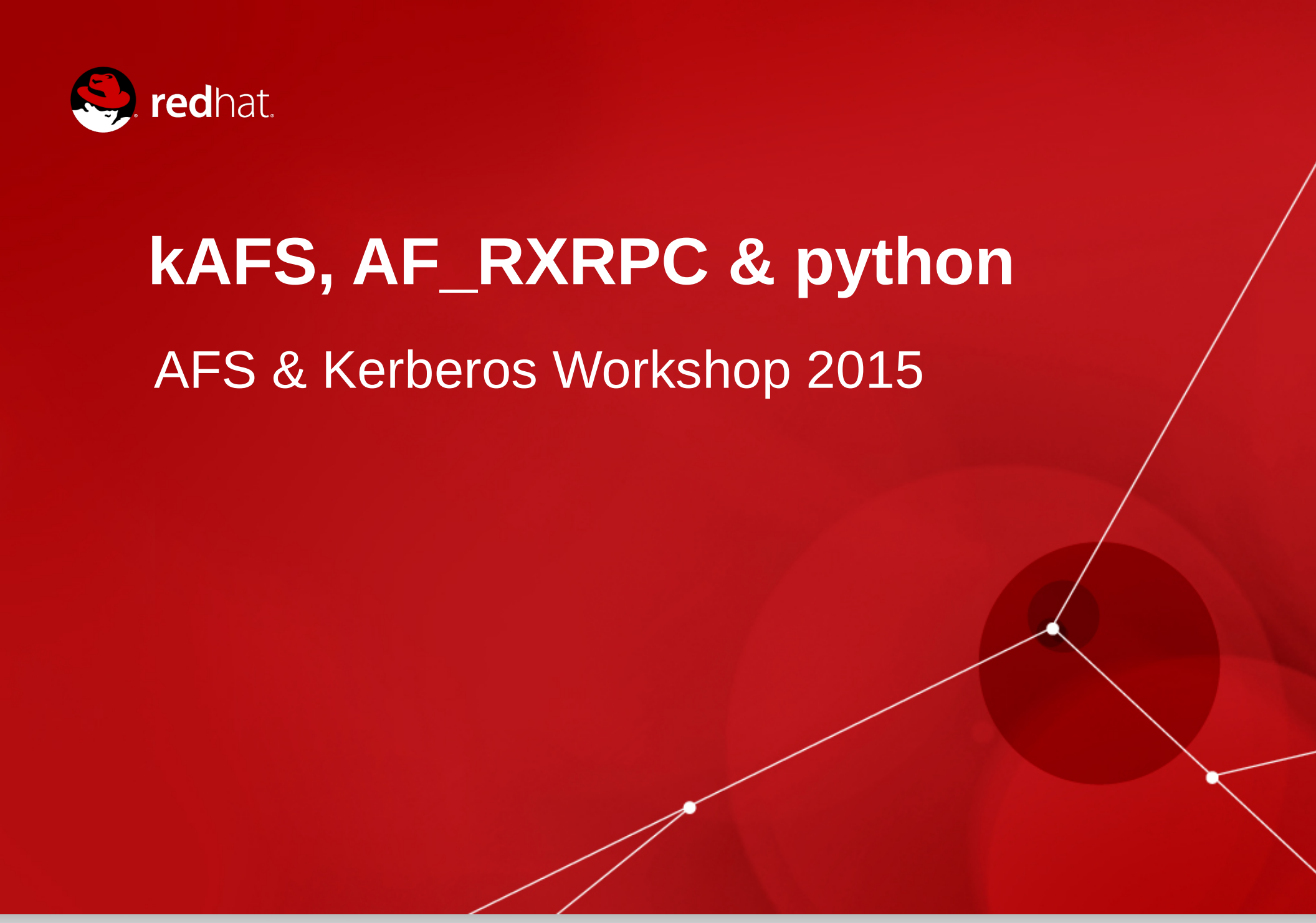


kAFS, AF_RXRPC & python

AFS & Kerberos Workshop 2015

Who?

- David Howells <dhowells@redhat.com>

Goals of kAFS

- Write a complete kernel filesystem that can drop-in replace OpenAFS's kernel module
- Able to use OpenAFS userspace tools
  - Not so easy as it appears!

The ultimate goal

- Fedora/RHEL integration
- Install by default
- Set up at system installation

kAFS Architecture

- kAFS comprises a number of components within the kernel:
  - Authentication management – keys
  - Transport – AF_RXRPC
  - Local cache – FS-Cache
  - Filesystem client – kAFS
- All within the kernel: no userspace component

Kernel Keys & Keyrings

Keys

- Data required by kernel services for secure operation:
  - Identities
  - Authorisation tokens
  - Encryption keys
- May be used by userspace too
- Not limited to use by AFS

Keys: Keyrings

- Keyrings are special keys that hold other keys
- Contents can be added and deleted
- Processes have special keyrings that can be inherited and shared
  - *Session keyring*
    - Set up by PAM upon login
    - Inherited across *fork()*
  - *Persistent per-user keyring*
    - Used by Kerberos

Keys: Use

- Allow kernel filesystems and drivers to look up and cache keys
- Allow user programs to propose and look up keys
- *add_key()*, *request_key()* and *keyctl()* system calls
- *pam_keyinit* module
- *keyctl* program

Keys: After login

- After logging in, I see:

```
[dhowells@andromeda ~]$ cat /proc/keys
17517315 I--Q--     1 perm 1f3f0000  4043    -1 keyring   _uid_ses.4043: 1/4
1bac056e I--Q--     5 perm 1f3f0000  4043  4043 keyring   _ses: 1/4
2ea802a8 I--Q--     3 perm 1f3f0000  4043    -1 keyring   _uid.4043: empty


[dhowells@andromeda ~]$ keyctl show
Session Keyring
       -3 --alswrv   4043  4043  keyring: _ses
782762664 --alswrv   4043    -1   \_ keyring: _uid.4043
```

AF_RXRPC

AF_RXRPC

- Linux network protocol
- The transport for kAFS
- Usable from userspace
- Kernel hides underlying UDP sockets
- UDP sockets shared between multiple processes
- Multiplex over a single fd
- Client and server

AF_RXRPC

- *socket()* interface family:

```
fd = socket(AF_RXRPC, SOCK_DGRAM, PF_INET);
```

- Can be used from the kernel or from userspace
- Does RxRPC remote procedure call transport in the kernel
  - Uses *recvmsg()* and *sendmsg()*
  - Uses ancillary data for parameters
- Can be a client or a server
  - *connect()*, *bind()* and *listen()* are available

AF_RXRPC

- RxRPC connections are shared between callers with same:
  - Source, destination, direction, service ID and key
- New connections are created automatically to expand call slots
- UDP port belongs to AF_RXRPC and so can be shared
  - Userspace and kernel can share

AF_RXRPC

- AF_RXRPC state can be viewed through /proc:

```
[root@andromeda ~]# cat /proc/net/rxrpc_calls
Proto Local        Remote            SvID ConnID   CallID   End Use State    Abort    UserID
UDP   0.0.0.0:7001 90.155.74.22:7000    1 00000008 00000002 Clt   1 Complete 00000000 ffff880002c31058
UDP   0.0.0.0:7001 90.155.74.22:7000    1 00000008 00000003 Clt   1 Complete 00000000 ffff880002c31058
UDP   0.0.0.0:7001 90.155.74.22:7000    1 00000008 00000004 Clt   1 Complete 00000000 ffff880002c31058
UDP   0.0.0.0:7001 90.155.74.22:7000    1 00000008 00000005 Clt   1 Complete 00000000 ffff880002c31058
UDP   0.0.0.0:7001 90.155.74.22:7000    1 00000008 00000006 Clt   1 Complete 00000000 ffff880002c31058
UDP   0.0.0.0:7001 90.155.74.22:7000    1 00000008 00000007 Clt   1 Complete 00000000 ffff880002c31058


[root@andromeda ~]# cat /proc/net/rxrpc_conns
Proto Local        Remote            SvID ConnID   Calls    End Use State    Key      Serial   ISerial
UDP   0.0.0.0:7001 90.155.74.22:7003   34 00000008 00000001 Clt   0 Client   2bed31d2 00000003 00000002
UDP   0.0.0.0:7001 90.155.74.22:7000    1 00000008 00000007 Clt   6 Client   2bed31d2 00000011 00000009
UDP   0.0.0.0:7001 90.155.74.22:7000    1 284bd9a0 00000000 Svc   0 SvUnsec  00000000 00000001 00000007
```

AF_RXRPC: Security

- Transparent to user
- Uses keys to pass security tokens
- Can do authenticated and encrypted transfer
- Can do *kauth, Kerberos 4* and *Kerberos 5*
- Uses *setsockopt()* to set parameters
- Uses kernel crypto layer

AF_RXRPC: Security

- My test klog adds a key for AF_RXRPC to the session keyring:

```
[dhowells@andromeda ~]$ klog
[dhowells@andromeda ~]$ cat /proc/keys
17517315 I--Q--     1 perm 1f3f0000  4043    -1 keyring   _uid_ses.4043: 1/4
19755aa7 I--Q--     1   1d 39390000  4043  4043 rxrpc     afs@CAMBRIDGE.REDHAT.COM
1bac056e I--Q--     5 perm 1f3f0000  4043  4043 keyring   _ses: 2/4
2ea802a8 I--Q--     3 perm 1f3f0000  4043    -1 keyring   _uid.4043: empty

[dhowells@andromeda ~]$ keyctl show
Session Keyring
       -3 --alswrv   4043  4043  keyring: _ses
782762664 --alswrv   4043    -1   \_ keyring: _uid.4043
427121319 --als--v   4043  4043   \_ rxrpc: afs@CAMBRIDGE.REDHAT.COM
```

- AF_RXRPC and kAFS pick these keys up by name

AF_RXRPC: Missing Features

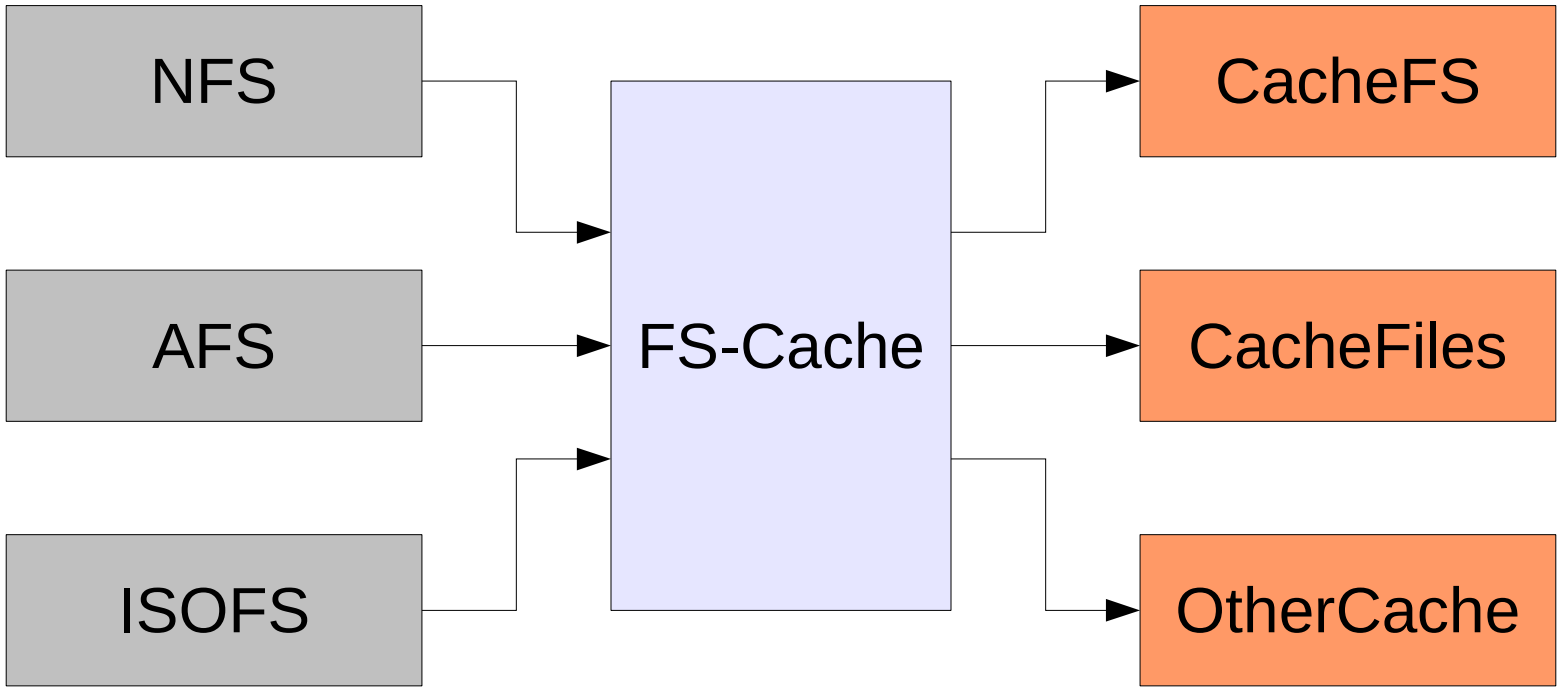
- IPv6
- GSS
- Debug & stat packets

FS-Cache

FS-Cache

- Local disk cache for network filesystems
- Persistent across reboots
- Designed for AFS, but also used by NFS, CIFS, ...
- Transparent to user

FS-Cache




redhat.
kAFS

kAFS

- Supports:
  - Reading and writing
  - Advisory file locking
  - Security
  - Automounting of mount points
    - Superblock per volume
  - Local caching
  - Failover
  - AFSDB DNS records

kAFS

- Does not yet support (or in progress):
  - *pioctl()*
    - AFS userspace tools
  - *ioctl()*
  - *inotify(), dnotify()*
  - DNS SRV records
  - Tuning
  - Disconnected operation

kAFS

- Write performance is good
  - Batching of writes
- Read performance is slow
  - Does single-page reads over the network
  - Should properly implement *->readpages()*

kAFS: Missing Features

- *pioctl()*
- PAGs
- IPv6
- TCP or SCTP
- SRV records
- GSS
- Should handle VMOVED and VBUSY
- 64-bit timestamps

kAFS: PAGs

- kAFS does not support PAGs
  - Keys go in the session keyring

kAFS: pioctl()

- Linus hates the idea
- *pioctl()* is abused
- Pathless *pioctl()* calls should not exit
- Need to substitute xattr calls, keyrings, ...
- Necessary to support OpenAFS userspace

OpenAFS and kAFS coexistence

- Would like for OpenAFS and kAFS to be able to coexist on a machine
- Ought to be trivial:
  - Give them separate local Cache Manager ports
  - Give them different mountpoints
  - Shared keys
- However...
  - Pathless *pioctl()* calls are a pain


redhat.
Userspace Utility Suite in Python

kafs-utils

- Userspace utilities written in Python 3
- C library to do RxRPC transport
- Uses AF_RXRPC

http://git.infradead.org/users/dhowells/kafs-utils.git

Kafs-utils: User Interface

- Character-for-character compatibility with OpenAFS?
  - Needed to support existing scripts
- New interface
  - Could be easier to use
- Bash completion makes the OpenAFS syntax easier

Kafs-utils: Internals

- The build process processes xg files into a python module
- RPC calls appear as methods
- Command scripts use these methods directly

Kafs-utils: Internals

- Each command is a python module
- Command lists, command exclusions, limits descriptions are all per-module global array variables

Kafs-utils: bos setcellname (1/2)

```
command_arguments = [
    [ "server",      get_bosserver,    "rs",     "<machine name>" ],
    [ "name",        get_cell,         "rs",     "<cell name>" ],
    [ "cell",        get_cell,         "os",     "<cell name>" ],
    [ "noauth",      get_auth,         "fn" ],
    [ "localauth",   get_auth,         "fn" ],
    [ "verbose",     get_verbose,      "fn" ],
    [ "encrypt",     get_dummy,        "fn" ],
]

cant_combine_arguments = [
    ( "cell",           "localauth" ),
    ( "noauth",         "localauth" ),
]

argument_size_limits = {
    "name"              : kafs.BOZO_BSSIZE,
}
```

Kafs-utils: Internals

- Each command module provides a main function as the entry point
- Dispatcher passes the set of parameters obtained
  - *None* indicates no parameter
  - *get_xxx()* functions perform translations
- The generated C-library protocol methods know how to convert python types to expected protocol types
  - Integers, structs
  - Lists, pointers, strings

Kafs-utils: bos setcellname (2/2)

```
description = r"""
Set the BOS server's current cell name
"""

def main(params):
    cell = params["cell"]
    bos_conn = cell.open_bos_server(params["server"], params)

    ret = kafs.BOZO_SetCellName(bos_conn, str(params["name"]))
```

Kafs-utils: Internals

- Xg constants appear as protocol module constants
- Protocol functions return complex python type with named elements
- Protocol types converted to Python types

Kafs-utils: pts listentries

```
def main(params):
    cell = params["cell"]

    if "users" in params and "groups" in params:
        flags = kafs.PRWANTUSERS | kafs.PRWANTGROUPS
    elif "groups" in params:
        flags = kafs.PRWANTGROUPS
    else:
        flags = kafs.PRWANTUSERS

    ret = cell.call_pt_server(params,  kafs.PR_ListEntries, flags, 0)

    output("Name                             ID   Owner Creator\n")
    for i in ret.entries:
        outputf("{:24s} {:7d} {:7d} {:7d}\n", i.name, i.id, i.owner, i.creator)
```

Kafs-utils: Internals

- Aborts are thrown as exceptions
- Caught by command dispatched if command doesn't handle them

Kafs-utils: pts adduser excerpt

```
try:
     ret = cell.call_pt_server(params, kafs.PR_AddToGroup, uid, gid)
     prcache.evict_groups()
except kafs.AbortPRIDEXIST:
     error("Entry for id already exists ; unable to add user ",
           user, " to group ", group, ignored, "\n")
```

Kafs-utils: Future development

- Not all the commands are done
- Some poorly documented RPC calls
- Middle layer?
- Transport substitution – librx?
- Support Python 2 also?
- Better UI for humans?
- Better UI for machines?
- Direct python scripting? – vos each