ПРОФЕССИОНАЛИТЕТ, 2025, № 3 (007)

***УДК** 004.451:004.7*

Э. В. Тищенко[1], В. Ю. Корякин[1]

[1]ФГАОУ ВО «РГУ нефти и газа (НИУ) имени И.М. Губкина», г. Москва, Российская Федерация

ИССЛЕДОВАНИЕ ФУНКЦИОНАЛА ПО УПРАВЛЕНИЮ СЕТЬЮ ПОСРЕДСТВОМ ПОДСИСТЕМЫ D-BUS В ОС АЛЬТ

Аннотация. В статье исследуются возможности управления сетевыми подключениями в операционной системе Альт с использованием универсальной системной шины D-Bus и сервиса NetworkManager. Актуальность работы обусловлена необходимостью построения гибких средств автоматизации сетевой подсистемы, которые способны не только выполнять заранее заданные команды, но и реагировать на изменения состояния интерфейсов в режиме реального времени. Рассмотрены теоретические основы межпроцессного взаимодействия D-Bus, принципы адресации объектов, использование сервисов, методов и сигналов, а также архитектура NetworkManager как основного механизма управления проводными и беспроводными подключениями в современных Linux-дистрибутивах. Отдельное внимание уделено различию между управлением через статические конфигурационные файлы и программным обращением к API системного демона. Практическая часть включает два эксперимента: сравнение опросного Polling-подхода с событийно-ориентированным переключением на резервный канал через D-Bus API и разработку Python-агента, изменяющего правила firewall в зависимости от активного сетевого подключения. Полученные результаты показывают, что событийная модель D-Bus обеспечивает более быструю реакцию на отказ основного интерфейса, снижает необходимость постоянного опроса системы и позволяет создавать автоматизированные средства управления безопасностью. Показано, что такой подход может применяться в учебных стендах, при администрировании рабочих станций и при разработке сервисов мониторинга. Сделан вывод о применимости D-Bus API NetworkManager для построения надежных и расширяемых решений, интегрированных в инфраструктуру ОС Альт.

Ключевые слова: D-Bus, ОС Альт, сетевые интерфейсы, сигналы, NetworkManager, API, автоматизация.

Введение и теоретические основы

Сетевая подсистема является фундаментальным компонентом любой современной операционной системы. Хотя методы управления посредством вызова утилит командной строки надежны, они не позволяют достичь нужной гибкости и оперативности для построения сложных систем автоматизации. В операционных системах семейства Linux, включая ОС Альт, де-факто стандартом для межпроцессного взаимодействия (IPC) является шина сообщений D-Bus. Этот механизм обеспечивает универсальную и масштабируемую связь между системными службами и приложениями [1, с. 325].

В современных дистрибутивах Linux, включая BaseALT, управление сетью осуществляется преимущественно с помощью демона NetworkManager. Он был выбран для исследования, поскольку стал де-факто стандартом и полностью основан на взаимодействии с шиной D-Bus. NetworkManager обеспечивает автоматическое обнаружение сетевых устройств, унифицированное управление проводными и беспроводными подключениями, а его интеграция с D-Bus делает его идеальным инструментом для изучения программного управления сетью [2]. Выбирать etcnet, к примеру, нельзя, так как etcnet является способом управления сетью через статические файлы, а D-Bus используется для динамического управления интерфейсами и событиями через API, а не через прямое редактирование конфигурационных файлов [4]. Далее в эксперименте № 1 будет проведен сравнительный анализ производительности двух автоматизированных методов для подтверждения тезиса. Изучение принципов работы D-Bus API NetworkManager даст понимание того, как реализовывать программное управление сетью, способное реагировать на изменения в системе, а не просто выполнять предопределенные команды [6].

Цель данной статьи – исследование и демонстрация на практике возможности управления сетью в ОС Альт через D-Bus API сервиса NetworkManager.

Подсистема межпроцессного взаимодействия D-Bus

D-Bus (Desktop Bus) — это система межпроцессного взаимодействия (IPC), являющаяся системной шиной для обмена сообщениями между приложениями. Работа шины зависит от следующих принципов [7]:

– шина: центральный демон, который отвечает за пересылку сообщений между процессами. Существует 2 шины – системная, которая отвечает за коммуникацию системных серверов, и шина сеанса, с помощью которой связываются между собой пользовательские приложения;
– сервисы: сервисами называют приложения, которые регистрируют свое имя при подключении к шине для обеспечения доступа других приложений к ним [5, с. 203];
– объекты и пути: сервисы предоставляют свой функционал и доступ к своим данных по средствам объектов, адресация к которым осуществляется благодаря иерархическим путям;
– интерфейсы: интерфейсы логически группируют методы и сигналы, формируются объектами;
– методы: методами являются функции, которые вызываются клиентами для выполнения какого-либо действия;
– сигналы: сообщения, которые сервис рассылает всем подписчикам для уведомления о событиях [3].

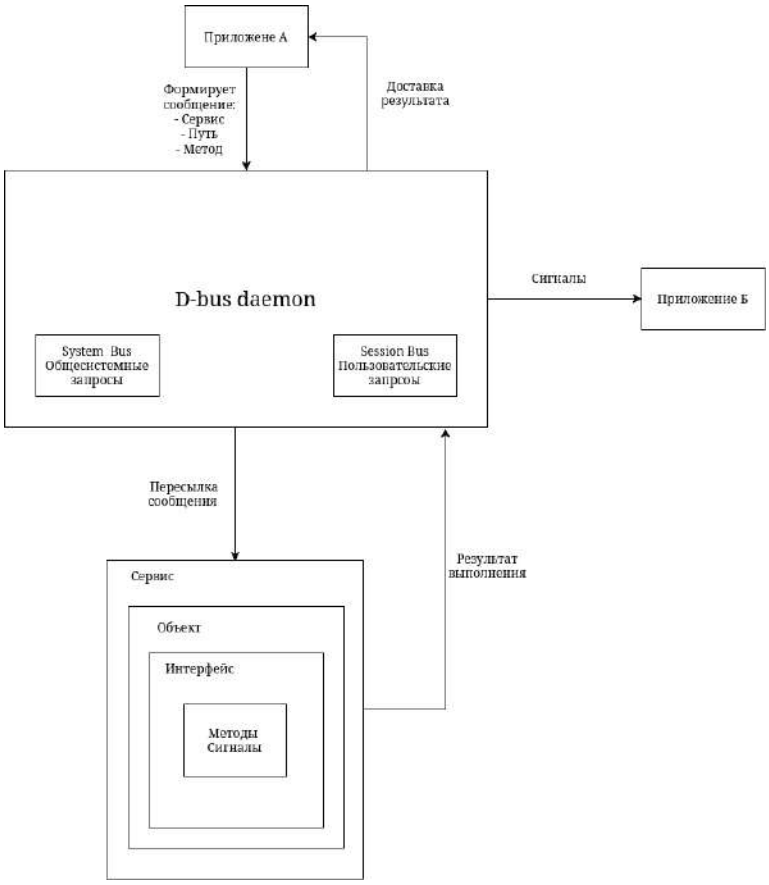

Рис. 1. Структурная схема IPC

Сервис NetworkManager и его интеграция с D-Bus

NetworkManager – это системный демон, его основной задачей является стандартизация и упрощение процессов настройки и поддержания сетевых соединений. Он автоматически обнаруживает сетевые устройства, управляет профилями подключений и выбирает оптимальное соединение [2].

Полностью раскрыть свой потенциал NetworkManager может через D-Bus. Он регистрируется на системной шине под именем org.freedesktop.NetworkManager и предоставляет набор объектов для управления. Интерфейс D-Bus предоставляет возможность:

– получать список сетевых интерфейсов и их состояние;
– создавать, изменять и удалять настройки/конфигурации подключения к сети;
– активировать и деактивировать соединения;

– подписываться на сигналы об изменении состояния сети, устройствах или точках доступа Wi-Fi.

Иными словами, D-Bus – это не способ обмена информацией, а фундаментальный слой, на котором построена вся современная архитектура управления сетью в ОС Альт.

Методология исследования

Объект исследования: подсистема управления сетью операционной системы Альт.

Предмет исследования: функциональные возможности по управлению сетью, предоставляемые через D-Bus API сервиса NetworkManager.

Среда проведения: исследование проводилось на операционной системе BaseALT. В качестве основных инструментов для взаимодействия с D-Bus использовались утилиты командной строки gdbus и busctl, а также язык программирования Python с библиотекой python-dbus для разработки программного агента.

Экспериментальная часть была спланирована таким образом, чтобы последовательно исследовать все аспекты взаимодействия: от изучения структуры API до создания автоматизированного решения. Общий план экспериментов представлен в таблице 1.

Таблица 1 – Информация о проведенных экспериментах

	Цель эксперимента	Средства	Ожидаемый результат
	Автоматическое переключение на резервный канал	NM, nmcli, D-Bus API, Python	Сравнение двух методов переключения, которые реализованы посредством кода
	Динамическое изменение контекста безопасности	nmcli, Python, D-Bus API, iptables	Python-агент, меняющий политики безопасности firewall в зависимости от подключения

Эксперимент 1. Автоматическое переключение на резервный канал

Цель данного эксперимента заключается в том, чтобы исследовать и сравнить эффективность двух разных методов реализации: Polling-метода (опросный) и событийно-ориентированного метода (D-Bus) на базе NetworkManager.

Polling-метод.

В основе данного метода лежит периодический опрос состояния сетевого интерфейса посредством утилиты nmcli. Для реализации был написан Bash-скрипт, который через заданный интервал проверяет состояние основного интерфейса и при обнаружении сбоя/отказа инициирует переключение на резервное подключение. Ниже приведен листинг скрипта:

```
#!/bin/bash

CHECK_INTERVAL=1

MAIN_DEV="enp0s3"
MAIN_CON="main-nat"
BACKUP_CON="second-nat"

echo "Запуск мониторинга (Polling). Интервал: ${CHECK_INTERVAL}c"
echo "Нажмите Ctrl+C для остановки"

while true; do
  STATE=$(nmcli -g GENERAL.STATE device show $MAIN_DEV | cut -c 1)
  #echo $STATE

  if [ "$STATE" == '3' ]; then
    echo "[FAIL] Обнаружен отказ $MAIN_DEV в $(date +%H:%M:%S.%N)"
    echo "Переключение на $BACKUP_CON..."

    # Замер времени начала переключения
    start_time=$(date +%s%N)

    nmcli connection up "$BACKUP_CON"
```

```
    end_time=$(date +%s%N)
    echo "Переключение завершено за $(( ($end_time - $start_time) / 1000000 )) мс"

    # Выходим, чтобы не зациклиться, если второй тоже упадет
    break
  fi

  sleep $CHECK_INTERVAL
done
```

Для тестирования необходимо выключить резервный интерфейс. После запускаем скрипт ./polling_test.sh и отключаем основное устройство при выключенном втором nmcli dev disconnect enpos3. Ниже представлен листинг работы скрипта при обнаружении неисправности работы основного интерфейса:

```
[root@rtr test_area]# ./polling_test.sh
Запуск мониторинга (Polling). Интервал: 1с
Нажмите Ctrl+C для остановки
[FAIL] Обнаружен отказ enp0s3 в 16:42:30.635068158
Переключение на second-nat...
Подключение успешно активировано (активный путь D-Bus:
/org/freedesktop/NetworkManager/ActiveConnection/86)
Переключение завершено за 716 мс
```

Из листинга выше видно, что процесс переключения на резервное подключение занял 716 мс.

Событийно-ориентированный метод.

В данном методе используется прямое взаимодействие с NetworkManager через D-Bus. Для проверки работоспособности был написан Python-скрипт, который подписывается на сигнал PropertiesChanged сетевого устройства и реагирует на изменение его состояния. При получении сигнала о сбое/отключении основного устройства, Python-агент немедленно вызывает метод ActivateConnection посредством D-Bus, тем самым активируя резервное подключение. Листинг программы представлен ниже:

```
import dbus
import dbus.mainloop.glib
from gi.repository import GLib
import time

MAIN_IFACE = "enp0s3"      # Интерфейс, который мониторим
BACKUP_CON_NAME = "second-nat" # Имя подключения для переключения
BACKUP_IFACE = "enp0s8"    # Физический интерфейс для бэкапа

NM_DEVICE_STATE_ACTIVATED = 100
NM_DEVICE_STATE_DISCONNECTED = 30
NM_DEVICE_STATE_FAILED = 120

def get_device_path(interface_name):
  """Находит объектный путь устройства по имени (например, enp0s3)"""
  bus = dbus.SystemBus()
  nm = bus.get_object('org.freedesktop.NetworkManager', '/org/freedesktop/NetworkManager')
  devices = nm.GetDevices(dbus_interface='org.freedesktop.NetworkManager')

  for dev_path in devices:
    dev_obj = bus.get_object('org.freedesktop.NetworkManager', dev_path)
    iface = dev_obj.Get('org.freedesktop.NetworkManager.Device', 'Interface',
                dbus_interface='org.freedesktop.DBus.Properties')
    if iface == interface_name:
      return dev_path
  return None
```

```
def get_connection_path(connection_name):
    """
    Находит объектный путь подключения (connection path) по его имени.
    Это обязательно, так как D-Bus работает с путями, а не с именами.
    """
    bus = dbus.SystemBus()
    settings_proxy = bus.get_object('org.freedesktop.NetworkManager',
                                    '/org/freedesktop/NetworkManager/Settings')
    settings_iface = dbus.Interface(settings_proxy, 'org.freedesktop.NetworkManager.Settings')

    connections = settings_iface.ListConnections()

    for conn_path in connections:
        conn_proxy = bus.get_object('org.freedesktop.NetworkManager', conn_path)
        conn_settings = dbus.Interface(conn_proxy, 'org.freedesktop.NetworkManager.Settings.Connection')


        config = conn_settings.GetSettings()

        # config['connection']['id'] - это то самое имя, которое мы видим в nmcli
        if config['connection']['id'] == connection_name:
            return conn_path

    print(f"[ОШИБКА] Подключение '{connection_name}' не найдено в настройках!")
    return None

def properties_changed(interface_name, changed_properties, invalidated_properties):
    """Обработчик сигнала об изменении свойств основного устройства"""
    if 'State' in changed_properties:
        new_state = changed_properties['State']

        if new_state == NM_DEVICE_STATE_DISCONNECTED or new_state == NM_DEVICE_STATE_FAILED:
            print(f"[SIGNAL] Фиксация отказа основного канала (State: {new_state}) в {time.time()}")

            start_time = time.time()

            try:

                nm_proxy = bus.get_object('org.freedesktop.NetworkManager', '/org/freedesktop/NetworkManager')
                nm_iface = dbus.Interface(nm_proxy, 'org.freedesktop.NetworkManager')

                print(f"-> Активация {BACKUP_CON_NAME} на {BACKUP_IFACE}...")

                nm_iface.ActivateConnection(
                    backup_conn_path,
                    backup_dev_path,
                    "/"
                )

                end_time = time.time()
                elapsed_ms = int((end_time - start_time) * 1000)

                print(f"[УСПЕХ] Переключение выполнено за {elapsed_ms} мс (Pure D-Bus)")

            except dbus.exceptions.DBusException as e:
                print(f"[ОШИБКА] Не удалось переключиться: {e}")


            loop.quit()
```

```
if __name__ == "__main__":
    dbus.mainloop.glib.DBusGMainLoop(set_as_default=True)
    bus = dbus.SystemBus()

    print("--- Инициализация Pure D-Bus Failover ---")

    backup_dev_path = get_device_path(BACKUP_IFACE)
    if not backup_dev_path:
        exit(1)
    print(f"Устройство бэкапа найдено: {backup_dev_path}")

    backup_conn_path = get_connection_path(BACKUP_CON_NAME)
    if not backup_conn_path:
        exit(1)
    print(f"Подключение найдено: {backup_conn_path}")

    main_dev_path = get_device_path(MAIN_IFACE)
    if not main_dev_path:
        exit(1)
    print(f"Мониторинг устройства: {main_dev_path}")
    print("Ожидание событий...\n")

    bus.add_signal_receiver(
        properties_changed,
        bus_name='org.freedesktop.NetworkManager',
        path=main_dev_path,
        dbus_interface='org.freedesktop.DBus.Properties',
        signal_name='PropertiesChanged'
    )

    loop = GLib.MainLoop()
    loop.run()
```

Далее необходимо предварительно выключить резервный интерфейс, запустить выполнение скрипта и эмулировать сбой основного интерфейса. В результате получаем такой вывод скрипта:

```
[SIGNAL] Фиксация отказа основного канала (State: 30) в 1770039928.9765048
-> Активация second-nat на enp0s8...
[УСПЕХ] Переключение выполнено за 29 мс (Pure D-Bus)
```

Из вывода программы можно заметить скорость выполнения переключения на резервное подключение – 29 мс, что значительно превосходит по скорости реакции традиционный Polling-подход. Использование D-Bus не создаёт дополнительной нагрузки на CPU, поскольку мониторинг состояния сетевого интерфейса осуществляется событийно – за счёт обработки сигналов NetworkManager, а не посредством постоянного опроса состояния системы. Сравнительный анализ представлен в виде таблицы ниже:

Таблица 2 – Сравнительный анализ подходов

Замеры	Время, мс	Процессор, %	Память, %
Polling	741.6	0.0-0.7	0.2
D-Bus	22.8	0.0	1.0

Таким образом, можно сделать вывод о том, что использование D-Bus в подобных вопросах является более выгодным решением с точки зрения нагрузки на систему.

Эксперимент 2. Динамическое изменение контекста безопасности.

Целью данного эксперимента является исследование возможности динамического изменения сетевой безопасности на основе получаемых сигналов от NetworkManager.

Суть эксперимента лежит в разработке микросервиса, который ловит сигнал переключения подключения и на его основе меняет настройки firewall.

Предварительно было создано новое подключение:

```
# nmcli con add type ethernet con-name local-private-net ifname enp0s9 ip4 77.77.0.121/24
```

Python-агент подписывается на сигнал StateChanged от NetworkManager. При активации подключения, оно идентифицируется, после чего применяется соответствующая политика безопасности. Для приватной сети SSH разрешен, а для обычной – SSH блокируется. Далее представлен листинг агента:

```
import dbus
import dbus.mainloop.glib
from gi.repository import GLib
import subprocess

TRUSTED_UUID = "96e6f92e-8dc7-4b5f-b89e-c12748e84c8f"
UNTRUSTED_UUID = "6761581e-4c8e-49ec-a7aa-a9d98d115ca7"

BLOCK_RULE = "INPUT -p tcp --dport 22 -j DROP"

def get_uuid_by_device_path(dev_path):
    """Получает UUID активного подключения по пути устройства"""
    try:
        bus = dbus.SystemBus()
        dev_proxy = bus.get_object('org.freedesktop.NetworkManager', dev_path)


        active_conn_path = dev_proxy.Get('org.freedesktop.NetworkManager.Device', 'ActiveConnection',
                                         dbus_interface='org.freedesktop.DBus.Properties')

        if not active_conn_path or active_conn_path == "/":
            return None

        ac_proxy = bus.get_object('org.freedesktop.NetworkManager', active_conn_path)
        uuid = ac_proxy.Get('org.freedesktop.NetworkManager.Connection.Active', 'Uuid',
                            dbus_interface='org.freedesktop.DBus.Properties')
        return uuid
    except Exception as e:
        print(f"Ошибка при получении UUID: {e}")
        return None

def apply_security_policy(uuid):
    """Логика безопасности"""
    if not uuid:
        return

    print(f"[*] Анализ безопасности для UUID: {uuid}")

    if uuid == UNTRUSTED_UUID:
        print(f"[ALERT] Недоверенная сеть. Блокируем SSH...")
        try:
            check = subprocess.run(["iptables", "-C", *BLOCK_RULE.split()], capture_output=True)
            if check.returncode != 0:
                subprocess.run(["iptables", "-A", *BLOCK_RULE.split(), "-m", "comment", "--comment", "DBUS_SEC_AGENT"],
check=True)
                print("   -> Правило добавлено (SSH закрыт).")
            else:
                print("   -> Правило уже активно.")
        except subprocess.CalledProcessError:
            pass # Ошибки игнорируем (правило уже есть)

    elif uuid == TRUSTED_UUID:
```

```
        print(f"[INFO] Доверенная сеть. Открываем SSH...")
        try:
            while True:
                res = subprocess.run(["iptables", "-D", *BLOCK_RULE.split(), "-m", "comment", "--comment",
"DBUS_SEC_AGENT"],
                                     capture_output=True)
                if res.returncode != 0:
                    break
            print("   -> Правило удалено (SSH открыт).")
        except Exception:
            pass

def device_state_changed(new_state, old_state, reason, sender_path):
    """
    Аргументы сигнала StateChanged NM:
    new_state (uint32)
    old_state (uint32)
    reason (uint32)
    sender_path (str) - это мы просим добавить через path_keyword
    """
    NM_DEVICE_STATE_ACTIVATED = 100

    if new_state == NM_DEVICE_STATE_ACTIVATED:
        print(f"\n[EVENT] Интерфейс {sender_path} стал АКТИВНЫМ")
        uuid = get_uuid_by_device_path(sender_path)
        if uuid:
            apply_security_policy(uuid)
        else:
            print("Не удалось определить UUID (возможно, интерфейс без IP).")

if __name__ == "__main__":
    dbus.mainloop.glib.DBusGMainLoop(set_as_default=True)
    bus = dbus.SystemBus()

    print("--- Агент безопасности запущен ---")
    print(f"Trusted: {TRUSTED_UUID}")
    print(f"Untrusted: {UNTRUSTED_UUID}")


    bus.add_signal_receiver(
        device_state_changed,
        bus_name='org.freedesktop.NetworkManager',
        dbus_interface='org.freedesktop.NetworkManager.Device',
        signal_name='StateChanged',
        path_keyword='sender_path'
    )

    loop = GLib.MainLoop()
    loop.run()
```

Для проверки работы, в одном терминале запускается скрипт, а в другом меняются подключения через nmcli con up.

При проверке политик для разных подключений можно заметить корректность работы агента:

```
# приватное подключение
INPUT (policy ACCEPT)
target     prot opt source               destination
DROP       tcp  --  anywhere             anywhere             tcp dpt:ssh /* DBUS_SEC_AGENT */
```

```
Chain FORWARD (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination

#обычное подключение
[root@rtr test_area]# iptables -L
Chain INPUT (policy ACCEPT)
target     prot opt source               destination

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
```

Данный эксперимент показывает, что с помощью D-Bus можно создавать сложные системы для гибкого обеспечения безопасности сети.

Заключение

В ходе исследования было подробно изучено управление сетью в BaseALT с помощью NetworkManager и шины D-Bus. При проведении экспериментов были наглядно показаны возможности взаимодействия с сетью через API, цель была достигнута.

Было выяснено, что D-Bus – это не просто набор команд, а целая система управления сетевой инфраструктурой. Эксперименты подтвердили возможность построения систем, способных в реальном времени реагировать на изменения состояния сети. Благодаря D-Bus API появляется возможность создавать более надежные, гибкие и быстрые решения, которые будут интегрированы в ядро ОС.

С практической точки зрения ценность данного исследования в демонстрации силы D-Bus как инструмента автоматизации.

Библиографический список

1. Шевченко, Д. А. Применение D-Bus для управления KDE в ОС АЛЬТ / Д. А. Шевченко, И. С. Боготов // Вестник науки. – 2025. – № 6. – С. 324–333.
2. Part III. D-Bus API Reference // NetworkManager Developer Documentation. – URL: https://networkmanager.dev/docs/api/latest/spec.html (дата обращения: 06.12.2025).
3. D-Bus // freedesktop.org Documentation. – URL: https://www.freedesktop.org/wiki/Software/dbus/ (дата обращения: 30.09.2025).
4. Уймин, А. Г. Нормирование показателей при организации киберполигона по сетевым технологиям / А. Г. Уймин // Нефть и газ – 2024 : тезисы докладов 78-й Международной молодежной научной конференции, Москва, 2024. – Москва, 2024. – С. 1253–1254.
5. Резник, В. Г. Операционные системы. Часть 2 : учебное пособие / В. Г. Резник. – Томск : ТУСУР, 2016. – 216 с.
6. Wheeler, D. Вводное руководство D-Bus / D. Wheeler, J. Palmieri, C. Walters // freedesktop.org. – 2020. – URL: https://dbus.freedesktop.org/doc/dbus-tutorial.html (дата обращения: 30.09.2025).
7. Потапов, А. А. Работа сетевой подсистемы в отечественных ОС / А. А. Потапов, Е. В. Чулисов // Мировая наука. – 2025. – № 1(94). – DOI: 10.5281/zenodo.14889548.

References

1. Shevchenko, D. A., & Bogotov, I. S. (2025). Using D-Bus to control KDE in ALT OS. Vestnik Nauki, (6), 324–333.
2. NetworkManager Project. (n.d.). Part III. D-Bus API Reference. NetworkManager Developer Documentation. Retrieved December 6, 2025, from https://networkmanager.dev/docs/api/latest/spec.html

3. freedesktop.org. (n.d.). D-Bus. Retrieved September 30, 2025, from https://www.freedesktop.org/wiki/Software/dbus/
4. Uymin, A. G. (2024). Normalization of indicators in the organization of a cyber polygon on network technologies. In Oil and gas – 2024: Abstracts of the 78th International Youth Scientific Conference (pp. 1253–1254).
5. Reznik, V. G. (2016). Operating systems. Part 2. Tomsk State University of Control Systems and Radioelectronics.
6. Wheeler, D., Palmieri, J., & Walters, C. (2020). D-Bus tutorial. freedesktop.org. Retrieved September 30, 2025, from https://dbus.freedesktop.org/doc/dbus-tutorial.html
7. Potapov, A. A., & Chulisov, E. V. (2025). The work of the network subsystem in domestic operating systems. Mirovaya Nauka, 1(94). https://doi.org/10.5281/zenodo.14889548

Информация об авторах

Тищенко Эльдар Валерьевич – студент ФКБ ТЭК, ФГАОУ ВО «РГУ нефти и газа (НИУ) имени И. М. Губкина», г. Москва, Российская Федерация, e-mail: eldar.tishchenko@gmail.com.

Корякин Владислав Юрьевич – студент ФКБ ТЭК, ФГАОУ ВО «РГУ нефти и газа (НИУ) имени И. М. Губкина», г. Москва, Российская Федерация, e-mail: v.koryakin4@yandex.ru.

RESEARCH OF NETWORK MANAGEMENT FUNCTIONALITY USING THE D-BUS SUBSYSTEM IN THE ALT OS

E. V. Tishchenko[1], V. Y. Koryakin[1]

[1]National University of Oil and Gas «Gubkin University»

Abstract. The article examines the possibilities of managing network connections in the ALT operating system by using the universal D-Bus system bus and the NetworkManager service. The relevance of the study is determined by the need to develop flexible tools for automating the network subsystem, which are able not only to execute predefined commands but also to respond to changes in interface states in real time. The paper considers the theoretical foundations of D-Bus interprocess communication, object addressing principles, the use of services, methods and signals, as well as the architecture of NetworkManager as a key mechanism for managing wired and wireless connections in modern Linux distributions. The practical part includes two experiments: a comparison of the polling approach with event-oriented failover switching through the D-Bus API and the development of a Python agent that changes firewall rules depending on the active network connection. The results show that the D-Bus event model provides a faster response to a failure of the primary interface, reduces the need for constant system polling and makes it possible to create automated security management tools. It is concluded that the NetworkManager D-Bus API can be used to build reliable and extensible solutions integrated into the ALT OS infrastructure.

Keywords: D-Bus, ALT OS, network interfaces, signals, NetworkManager, API, automation.

Information about the authors

Eldar Valeryevich Tishchenko is a student of the Federal Design Bureau of Fuel and Energy Complex, Gubkin Russian State University of Oil and Gas (NRU), Moscow, Russian Federation, e-mail: eldar.tishchenko@gmail.com.

Vladislav Yuryevich Koryakin is a student of the Federal Design Bureau of Fuel and Energy Complex, Gubkin Russian State University of Oil and Gas (NRU), Moscow, Russian Federation, e-mail: v.koryakin4@yandex.ru.