The official LinuxCNC EtherCAT driver (based on the IgH EtherCAT master userspace library) does not currently support Ethernet over EtherCAT (EoE) because it lacks handling for non-application datagrams, which are essential for tunneling standard Ethernet traffic (e.g., TCP/IP) through the EtherCAT segment. This is required for EoE-capable slaves like the Inovance SV660N servo drive, which uses EoE to enable networked access for configuration, diagnostics, and parameter tuning via tools like InoDriverShop without disrupting real-time CoE (CANopen over EtherCAT) process data.

EoE support in the IgH master is available but requires explicit integration in the application layer. The core issue is that the standard `ecrt_master_send()` function only handles application datagrams (e.g., PDO/SDO for motion control), while EoE relies on `ecrt_master_send_ext()` for non-application datagrams (e.g., tunneled Ethernet frames). This function is kernel-only in the stock IgH library but can be ported to userspace via ioctl calls when using real-time frameworks like Xenomai or RTAI.

Prerequisites

- **Real-time kernel**: Use a PREEMPT-RT or RTAI kernel for low-latency cyclic execution. Xenomai is recommended for userspace real-time support (RTDM).
- **IgH EtherCAT master build** : Compile with EoE enabled:

```
git clone https://gitlab.com/etherlab.org/ethercat.git
cd ethercat
./bootstrap
./configure --sysconfdir=/etc --enable-userlib --enable-eoe --with-userlib-rt=posix --disable-8139too
make
sudo make install
```

  - `--enable-eoe` : Activates the EoE module, which creates a virtual Ethernet interface (`eoe0`) for tunneling.
  - `--enable-userlib` : Builds the userspace library (`libethercat`).
- **linuxcnc-ethercat build**: After the above, rebuild the driver:

```
git clone https://github.com/linuxcnc-ethercat/linuxcnc-ethercat.git
cd linuxcnc-ethercat
make configure
make
sudo make install
```

- **SV660N-specific setup** (from Inovance documentation):
  - Enable the virtual Ethernet port on the slave via SDO (object 0xF800:01, bit 14 = 0) or ESI XML import in your configurator.
  - Assign IPs in the same subnet (e.g., master: 192.168.198.1/24, slave: 192.168.198.x/24).
  - Use the master's "Sync slave gateway" equivalent (manual IP assignment via SDO if needed).

Suggested Code Modifications

The key changes are in `src/ethercat.c` (the main driver source) to integrate `ecrt_master_send_ext()` into the cyclic real-time loop. This loop (typically in `update_pins()` or the RT task handler) already calls `ecrt_master_send()` and `ecrt_master_receive()` periodically (e.g., every 1 ms). Add EoE handling there.

1. **Add userspace `ecrt_master_send_ext()` declaration**:
   - In `src/ethercat.h` (or a new header), add a conditional prototype for RTDM/Xenomai userspace:

```
#ifdef EC_RTDM
#include <rtdm/rtdm.h>  // For Xenomai RTDM
size_t ecrt_master_send_ext(ec_master_t *master);
#endif
```

2. **Implement userspace `ecrt_master_send_ext()`** :
   - Create a new file `src/eoe_userspace.c` (or inline in `ethercat.c`) with an ioctl-based port (inspired by community patches like ribalda/ethercat for RTDM support):

```
#ifdef EC_RTDM
#include <rtdm/ethercat.h>  // Adjust for your RT framework

size_t ecrt_master_send_ext(ec_master_t *master) {
    int ret;
    size_t sent_bytes = 0;

    // Use ioctl to trigger non-app datagram send (EoE frames)
    ret = rt_dev_ioctl(master->device, EC_IOCTL_SEND_EXT, &sent_bytes);
    if (ret) {
        rt_printk("EoE send_ext failed: %d\n", ret);
        return 0;
    }
    return sent_bytes;
}
#endif
```

     - This calls the kernel's `EC_IOCTL_SEND_EXT` via RTDM ioctl, queuing EoE datagrams without blocking the RT cycle.
     - Compile with `-DEC_RTDM -DEC_EOE` flags in `Makefile.am` or `configure.ac` .

3. **Integrate into the cyclic loop**:

3. **Integrate into the cyclic loop**:
    - In `src/ethercat.c`, locate the real-time update function (e.g., `ethercat_rt_thread()` or `update_pins()` in the HAL component). It looks similar to:

```
// Existing cyclic loop (simplified)
static void update_pins(void) {
    // ... HAL pin updates ...
    if (ecrt_master_send(master) < 0) { /* error */ }
    if (ecrt_master_receive(master) < 0) { /* error */ }
    // ... process domains ...
}
```

    - Modify to include EoE send after application send (to prioritize RT data):

```
static void update_pins(void) {
    // ... HAL pin updates ...

    // Send application datagrams first
    if (ecrt_master_send(master) < 0) {
        lcec_error("Failed to send");
    }

    // Send non-app datagrams (EoE) if enabled
#ifdef EC_RTDM
    if (eoe_enabled) {  // Flag from config (e.g., INI param)
        ecrt_master_send_ext(master);
    }
#endif

    if (ecrt_master_receive(master) < 0) {
        lcec_error("Failed to receive");
    }

    // ... process domains ...
}
```

        - Add a config flag `eoe_enabled` (e.g., via `lcec.ini` or HAL pin) to toggle EoE without recompiling.
        - Call this every cycle (e.g., 500 µs–1 ms, matching your EtherCAT update rate).

4. **EoE module activation**:
    - In `src/main.c` or init code (e.g., `lcec_init()`), after `ecrt_master_activate()`:

```
#ifdef EC_EOE
if (eoe_enabled) {
    // Request EoE virtual interface creation
    ecrt_master_eoe_request(master);  // If exposed in userspace lib
    // Or via SDO to slaves: enable EoE mailbox (0xF800:01 = 0x0000)
}
#endif
```

    - Load the EoE kernel module: `sudo modprobe ec_eoe` (creates `eoe0` interface).

5. **Network configuration** (post-build, runtime):
    - Bring up the virtual interface:

```
sudo ip link set eoe0 up
sudo ip addr add 192.168.198.1/24 dev eoe0  # Match SV660N subnet
```

    - Enable IP forwarding on the host: `sudo sysctl -w net.ipv4.ip_forward=1`.
    - Add routes for slave access (e.g., for InoDriverShop): `sudo ip route add 192.168.198.0/24 dev eoe0`.
    - For SV660N, use SDO writes (via `lcec` conf or tool) to set slave IP (e.g., 192.168.198.2) and enable virtual port.

Testing and Validation

- **Basic EoE check**: After loading, verify `eoe0` interface with `ip link show`. Ping the SV660N IP from the host.
- **SV660N integration**: Import the updated ESI XML (with EoE params) into your `lcec` config. Scan slaves with `ethercat slaves`—EoE slaves should show as "EoE capable."
- **Full stack**: Run LinuxCNC with a test HAL file including your modified driver. Monitor with `ethercat eoe` for frame stats. Use InoDriverShop over the EoE IP to tune the drive while CoE runs motion.
- **Debug**: Enable IgH debug (`echo 0x2000 > /proc/ethercat/master0/debug`) and check dmesg for EoE errors. If cycles jitter >1 µs, tune send interval with `ecrt_master_set_send_interval(master, 1000)` (1 ms).

Limitations and Alternatives

- **Latency**: Userspace EoE adds ~10–50 µs overhead; test under load. If unstable, switch to full kernel modules (RTAI kernel, `--disable-userlib` in IgH configure).
- **Complexity**: Full EoE routing (e.g., NAT for external PCs) requires host firewall rules (`iptables -t nat -A POSTROUTING -o eoe0 -j MASQUERADE`).
- **If modifications fail**: Disable EoE temporarily (`--disable-eoe` in IgH configure) for CoE-only SV660N operation, or use a commercial master like Beckhoff TwinCAT (supports EoE natively but not open-source).
- **Community resources**: See LinuxCNC forum thread on EoE config and ribalda/ethercat repo for patches. Submit a PR to the repo for upstreaming.

These changes should enable basic EoE tunneling for SV660N drives while preserving LinuxCNC's real-time performance. Test incrementally to avoid RT deadlocks.