A Scalable, Smart, Self-learning Router for CANopen

Olaf Pfeiffer, Christian Keydel and Andrew Ayre, Embedded Systems Academy

Bridges, gateways and routers cannot only extend the maximum length of a CAN based network, they can also help increase overall bandwidth, as
network segments can be made shorter and faster and
network segments only receive network traffic that involves nodes connected to it.
In the past, configuring a CAN bridge was a time consuming process, as the bridge had no way of knowing which messages need to be forwarded to which network segments.
This changes if the protocol used is a higher-layer protocol such as CANopen. For CANopen a bridge or router can be build to be self-learning: due to a pre-defined connection set for identifiers, the device just listens to the network traffic and learns "on-the-fly" which messages need to be forwarded to which segments.
This paper summarizes the benefits and drawbacks of bridges and routers in a CANopen environment and gives an implementation outlook for a scalable, smart, self-learning router.

1. Background information

Using devices such as repeaters, bridges and gateways to extend the physical length of networks is nothing new. On the Internet, these devices are common practice and due to the high volume became so affordable, that they even found their way into private homes with high-speed Internet access.

For CAN, the physical limits of length and maximum speed are a serious limitation for several applications. On one hand there are applications where CAN was not yet even seriously considered because of these limits (such as large scale networking between gaming machines in casinos) and on the other hand there are more and more existing CAN applications that run out of available bandwidth because more and more nodes need to be added

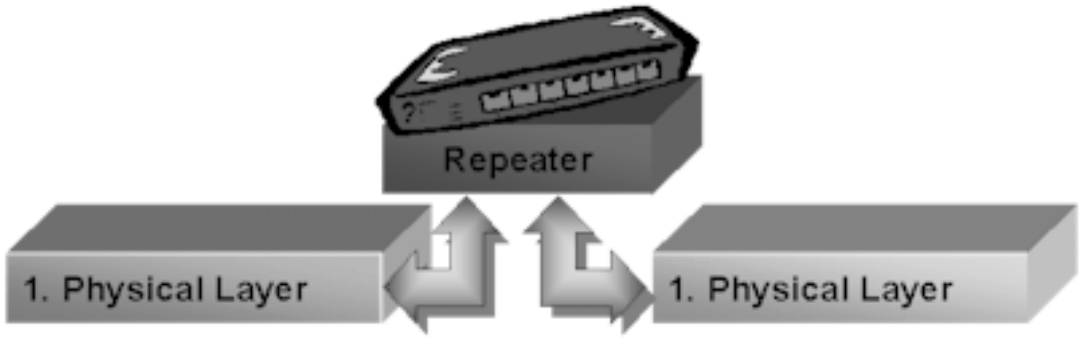

Figure 1 – Network layer interaction of a repeater

(for example high-end automotive and industrial control applications).

A good, affordable bridge or router for CAN would help to overcome the physical limits of CAN to a certain extend.

Before we further investigate possible options and implementation routes, let's compare the commonly used forwarding techniques in network environments: Repeaters, bridges, gateways and routers.

Repeater

A repeater acts on the physical layer and directly repeats the physical signal from one side of the repeater to the other. A repeater cannot be used to extend the length of a CAN network. CAN requires that a bit propagates over the entire bus before the next one can start. A repeater operating entirely on the signal level does not allow extending the total length of the network.

Bridges

A bridge acts on the data link layer and forwards entire message frames from one network to another. Bridges can extend the maximum length of a CAN network. On the downside, they have to completely receive a message frame, before they can forward it. So even the fastest bridge can easily have delay times of more than 150 bit times (the worst case message length can even be longer). This imposes some limits for applications with very high real-time demands.

On the upside, because bridges receive entire data frames, they can be built "smart". Smart, self-learning bridges are commonly used in the Internet. They do not simply forward all messages from one network segment to the other segment. They evaluate the packages and forward only those messages, that need to be forwarded. Smart, self-learning bridges can help to increase overall bandwidth, as communication local to each segment will not be forwarded and not occupy bandwidth on the other segment.

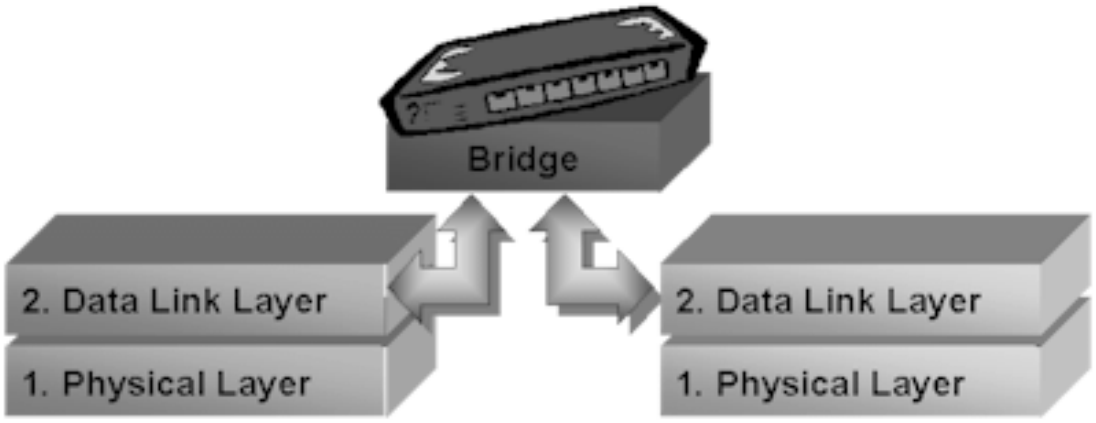

Figure 2 – Network layer interaction of a bridge

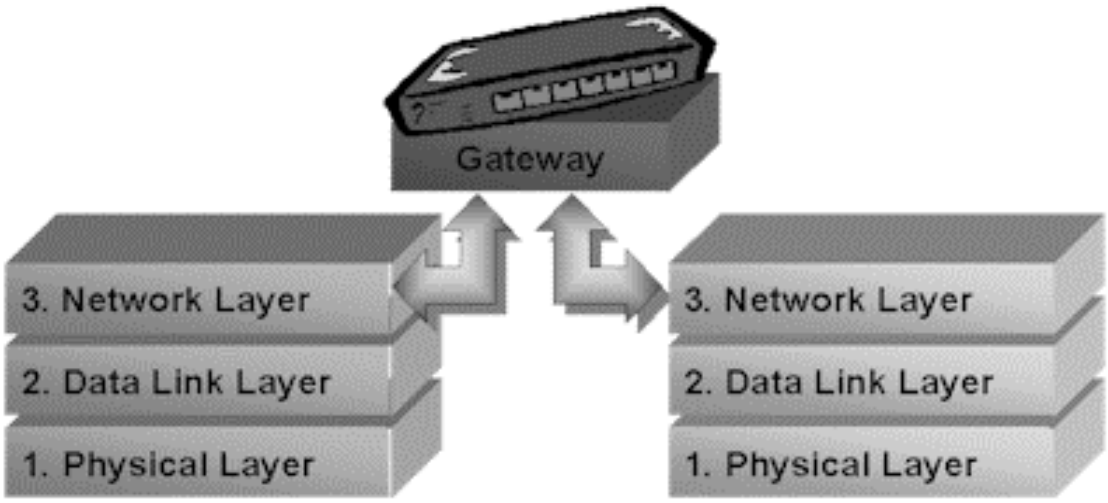

Figure 3 – Network layer interaction of a gateway / router

On a regular CAN network without a standardized higher layer protocol the configuration of smart bridges a challenge. Plug-and-play as available with self-learning bridges is close to impossible, as someone needs to configure the bridge manually. Otherwise, it just wouldn't know which messages to forward and which ones to keep local. While it may be possible to detect the identifiers used for sending in one segment of the network it is impossible to find out automatically which identifiers the nodes in a segment are listening to.

Gateways and routers

Gateways and Routers act on the 3rd or higher layers of the network protocol stack. They are used for long distance forwarding or forwarding between completely different network types (for example CAN and Internet). In general we call a device just connecting two network segments a gateway, whereas a device that connects multiple segments is called a router.

As CAN defines only the lower two layers of the network protocol stack, gateways and routers would need to be implemented for a specific higher layer protocol. The awareness of the higher-layer protocol also enables such forwarding devices to come much closer to plug-and-play as now self-learning can be implemented. Depending on the protocol it can be possible to learn about all forwarding requirements simply by listening to all communication.

When it comes to real-time behavior, the same limits as mentioned for bridges apply. Each message would need to be received entirely before a forwarding mechanism can begin.

Choosing to build a router

If we compare the previously shown forwarding technologies of repeater, bridges, gateway and router, then the router gives us the best benefits.

A repeater does not help with the extension of a CAN network. A bridge without higher protocol awareness requires extensive manual configuration. And a gateway only connects two

network segments, whereas a router could be build with multiple ports, such as 4 or even 8.

(see figure 4). Assuming a bit rate of 1Mbit/s on all segments the overall bandwidth of such a system is now in

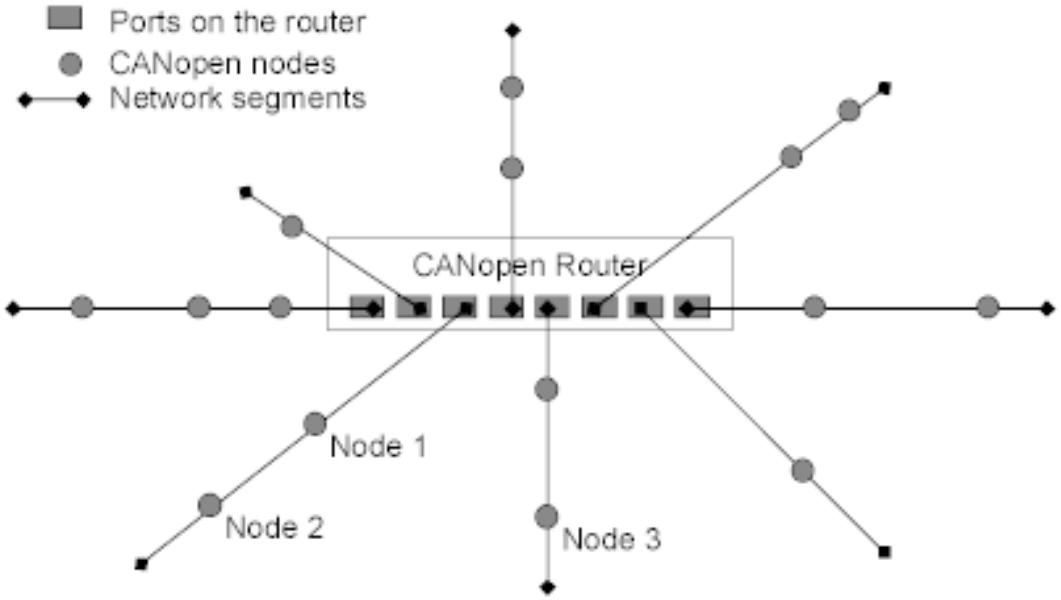

Figure 4 – Network layout with an 8-port router

For these reasons we picked the router for further examination and the following questions need to be answered:

Can we build a smart, self-learning router for CANopen networks? And what kind of benefits and drawbacks would we get?

2. The benefits of a router

Let's take the "What if?" approach: What if - there would be a CAN router with 8 ports that is smart, self-learning and fast, meaning it only forwards packages to network segments / ports as required and the delay time within the router is a "virtual" zero.

Bandwidth

This hypothetical device could be used to create a star topology with 8 segments and the router in the center

the range of 1 to 8Mbit/s. The worst case of 1Mbit/s occurs when one single segment needs to handle ALL messages transferred.

For example, if all messages need to be forwarded to one network segment (for example node 3 in figure 4 needs to receive ALL messages). The best case occurs when the router forwards no messages and independent, local communication of 1Mbit/s is used on all network segments (for example nodes 1 and 2 in figure 4 communicating with each other). As both cases do not make much sense, the real-world average would be somewhere in between and surely highly depends on network layout and requirements. However, an overall bandwidth of 3-4Mbit/s should be achievable.

It should be noted, that in general all network segments could operate at different baud rates.

Flexible topological structure

Although the maximum distance would be only twice as much as in a single CAN network segment, the total cable length in the flexible star layout would be eight times longer than a single CAN network segment.

The star topology is far more flexible than a single line bus, especially for network layouts that have to cover a larger 2- or 3-dimensional area.

Depending on the intelligence of the router, you can think of far more flexible and fault tolerant layouts by combining several routers (see summary and outlook at the end of the paper).

Configuration

To which level a forwarding device such as a router is auto-configuring or self-learning depends on the higher-layer CAN protocol supported by the device and on the level of integration into the system.

Goal of this paper is to show that with CANopen it is possible to achieve a self-learning, plug-and-play router that by simply listening to the network traffic learns which messages need to be forwarded to which segments.

Error handling

Serious communication faults like bad wiring or a bad node resulting in error frames on any one segment do not directly affect the other segments.

As long as the nodes in the affected segment are not crucial to the entire system, the system can still operate.

A router is a perfect instrument to separate crucial or safety-relevant communication from the remaining communication.

3. The drawbacks of a router

As mentioned before, any kind of forwarding device that entirely needs to receive a CAN message before it can start the forwarding mechanism imposes some drawbacks onto the system. The drawbacks primarily affect applications with high demands on the real-time behavior or those that require a good synchronization of inputs and/or outputs.

Delays

The following delays can occur when forwarding a message:

- One message length for every message forwarded (it needs to be received entirely, before a forwarding mechanism can start)
- Router internal processing delay (to keep this as short as possible is the challenge)
- Arbitration delay on destination segment (other higher priority messages might get through first)
- Forwarding buffer delay in router (if destination segment is very busy, router might need to buffer several outgoing messages)

How big the total maximum delay is, depends highly on the network layout and required communication between segments.

For applications requiring the usage of the CANopen SYNC message, a reasonable work-around might be to make the router the SYNC producer and to ensure that all SYNCs are produced on all network segments in parallel.

Overruns

One of the potential bottlenecks for such a system that could even result in message losses occurs if too many consumers are on one network segment. In that case too many

producers from too many segments could send more messages than the one destination segment could handle. This is something that the system designer would need to consider when laying out the network.

For a prototype implementation existing components should be used – so a completely new, dedicated hardware based on new CAN controller designs is not further pursued at this moment. Instead we will focus on building the

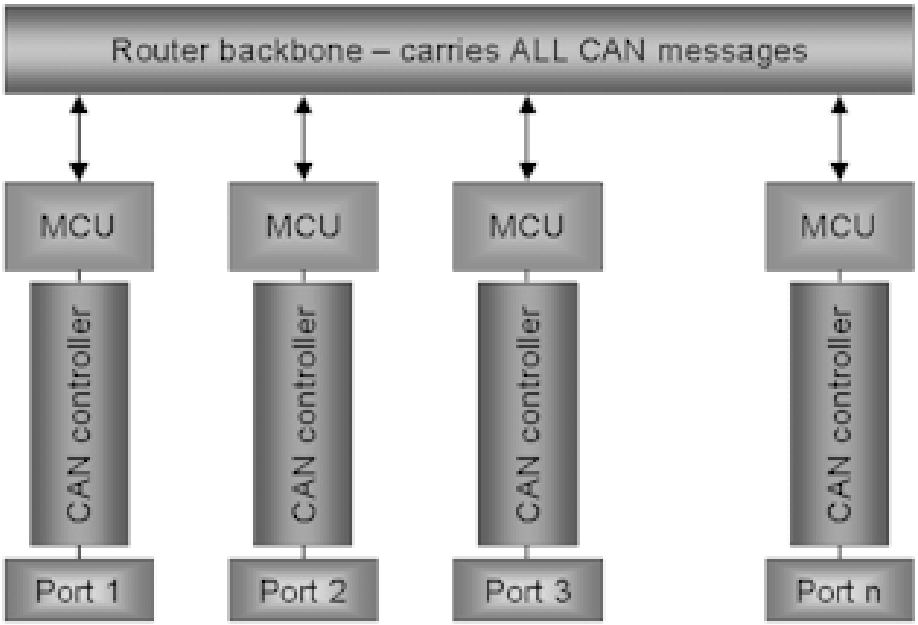

Figure 5 – A router with one MCU per port

router based on existing microcontroller and CAN controller solutions.

Obviously, a single processor driven router would probably sooner than later run into performance problems. There will always be a hard limit on how many CAN ports a single processor could handle at 100% busload.

That's why a scalable router (easily configurable for different numbers of ports) would need to be implemented with multiple processors. The most flexible layout would require one processor handling each CAN port. Additionally, these processors would need to have access to a high-speed backbone interconnecting all the processors within the router (see figure 5). This backbone would need a

4. Implementation Challenges

Now that we outlined what an 8-port smart, self-learning router could do for us, let's evaluate which technical challenges we face to implement one.

The Hardware

Any router would need to be able to handle 100% busload on each port – in worst case all at the same time. Assuming an 8-port system, this would mean an overall 8Mbit/s bandwidth to be handled. NOTE: For this scenario, a 100% busload caused by the shortest CAN data frames would result in more than 150,000 CAN messages per second to be received by the router!

bandwidth big enough to handle the worst-case throughput of the largest router configuration. For a router with 8 ports, that would be about 4Mbit/s (Remark: if all segments produce 100% busload, there is no bandwidth on any segment left, so the router could NOT forward any message and the used backbone bandwidth would be 0MBit).

The basic idea for the forwarding algorithm would be that any incoming message frame on any port gets broadcast via the backbone to all other ports. Using local lookup tables, each processor decides individually if a particular message needs to be forwarded to its local CAN network segment.

As we do not want to introduce yet another networking technology within the router, one of the simpler approaches to implement such a backbone comes from techniques used in massive parallel computers. In these computers, data can be shared with neighboring processors through a synchronized shifting process. In these synchronized access cycles, processors can access a neighbor's processor memory.

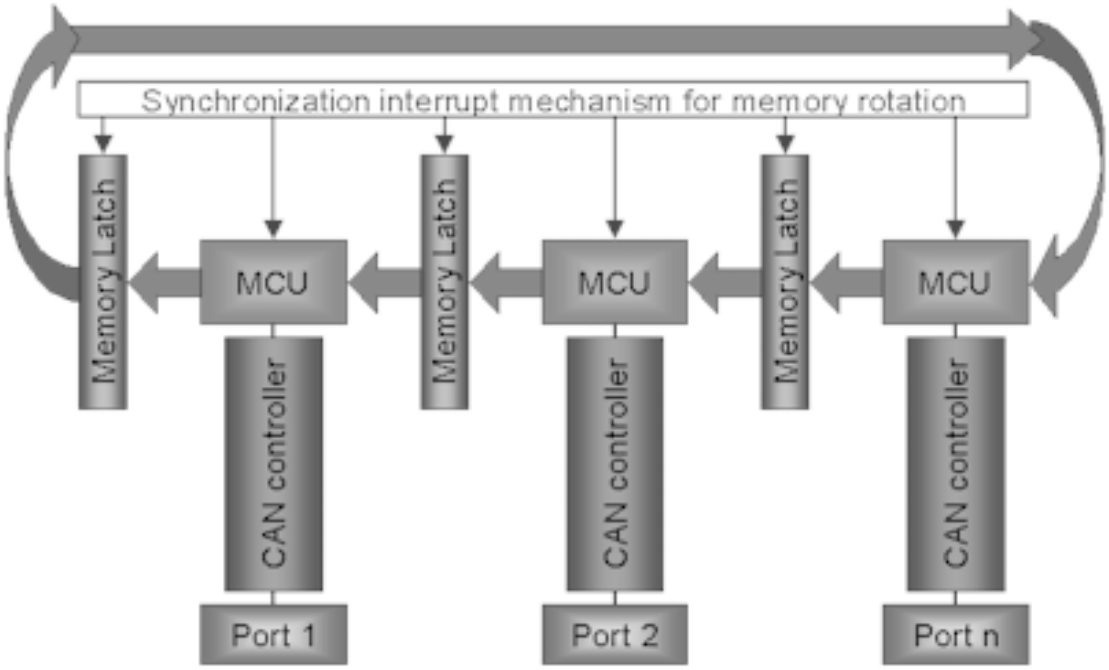

Figure 6 – Parallel data shifting between the MCUs

To adopt this technology to the router requires a synchronized "write to my left neighbor and read from my right neighbor" shift cycle. One simple method would be to use latches between the processors and upon a synchronized interrupt signal all processors would first write to the latches on one side and then read from the latches of the other side (see figure 6).

Selecting Microcontrollers

Due to the high demands on the throughput of the backbone of the router common 8-bit microcontrollers would be

stretched beyond their limits. On the other hand, common 16-bit microcontrollers have so much performance, that besides handling the backbone, they could also handle 2 instead of just one CAN port. And if we look beyond 16-bit, a 32-bit ARM derivative should easily handle about 4 CAN ports and still be able to maintain the high-speed backbone.

A first prototype would most likely be based on a 16-bit microcontroller with 2 CAN interfaces supporting router configurations with 2, 4, 6 or 8 CAN ports.

The Software

We currently envision two software versions for the CANopen router:

One that does not act as a CANopen device, it does not have a CANopen ID number and cannot be addressed. Its presence in the network is completely invisible to the other CANopen nodes.

The other version would be a CANopen device with its own node ID number and an Object Dictionary allowing for customized configurations.

The benefit of version one is that it is a true plug-and-play solution backward compatible to any existing CANopen network.

The benefit of version two is that the master can be made aware of the presence of a router. Knowing its presence the master can access router configuration or forwarding statistic data, reset the router or even take its presence into account when configuring other nodes. The router could also generate an emergency if overload, delays or overruns occur on any segment.

For the scope of this paper, we will focus on the plug-and-play version.

Self-learning

To support true self-learning, there are a few conditions that must be met when using the router in a CANopen network:

1. All nodes in the network must be CANopen conform.
2. All nodes must startup with the default, pre-defined connection set (if other connection sets are used, they must be configured by the master during the pre-operational state of each node)
3. The router must be powered-up and operational when the first boot-up message occurs on the network.

To implement the self-learning forwarding mechanism, the router needs a lookup table for each destination network segment / port. To handle 11-bit IDs (0 to 2047), a lookup table with 1 bit for each identifier is required (256 bytes). If a corresponding bit is set, it means that a message with that identifier is forwarded to the corresponding segment. Per default, all bits are set, meaning that every single message is forwarded to all segments.

The self-learning steps require, that with each message received, the router needs to double-check if the information contained (either by pure ID number, but in some cases also by the data) allows making conclusions about the future forwarding mechanism.

The scope of this paper does not allow listing all eventualities. However, a few examples can illustrate the mechanism:

When receiving a boot-up message on any port, the router can recognize which node ID produced this boot-up message. Knowing that a particular node is on a particular port allows us to clear all bits in the look-up tables of the other segments representing Receive PDOs or Receive SDOs associated with

that node. In other words, once the boot-up message is received and interpreted, messages sent to that node via SDO or PDO will never be forwarded to any other segment.

In a similar scenario, the NMT message send by the Network Management Master allows the router to determine the port that the master is connected to. All transmit PDOs and transmit SDOs do only need to be forwarded to the segment with the master, so the corresponding bits (a total of 5 * 127 = 635 bits) can be cleared in the look-up tables of all other segments.

A more complex example deals with the linking of PDOs. The default settings for PDOs are not very likely to be used in a real-world network. In most CANopen networks the COB IDs used for PDOs are changed. The master can change a COB ID with one expedited SDO transfer for each ID that needs to be changed.

In order to correctly handle these cases, the router needs to monitor all expedited SDO download requests issued by the master (ID and data contents). The following information needs to be recognized:

- CAN ID used – marks SDO request to certain node ID
- 1st data byte – marks expedited SDO download requests
- 2nd to 4th data byte – marks the index and subindex of the Object Dictionary entry affected
- 5th to 7th data byte – contains the data

If the router recognizes, that a PDO is assigned a new COB ID using the CAN message above, it needs to change the look-up tables accordingly. That could mean that bits that were previously cleared are now set again to allow the forwarding of the PDO to the appropriate destination segment.

5. Summary and outlook

Smart, self-learning CANopen routers as described in this document would not only extend the physical length of a CANopen network, they would also increase overall network bandwidth or throughput and allow system layouts with a far more complex network topology than ever before.

The router is “plug-and-play” as it does not need to be configured (other than baud rates used). However, the system integrator would need to pay more attention when selecting the triggering and scheduling mechanisms for each individual node to avoid bottlenecks.

Some applications with hard real-time requirements might not benefit from such a router, although some work-arounds are available for applications requiring synchronized IO.

Once a router as described in this paper becomes available, the next logical step would be to think about further software enhancements as found in Internet routers.

If a CANopen router is a configurable CANopen node with its own ID, it could be programmed to recognize other CANopen routers in the system. This could allow setting up “true rings” requiring the routers to use “true routing algorithms” similar to those used by Internet routers.

Figure 7 shows an example where multiple routers build a “backbone ring” consisting of 2 connections to each neighbor. Such a system could still work, even if any up to three network segments of the backbone ring break.

Depending on the complexity of the routing algorithms, the routes taken

could also be determined dynamically depending on the busload on each segment. If the busload on a particular segment is already high, the routers could automatically send messages via the longer route, if that is less congested.

As with any CAN / CANopen implementation, feasibility of bridging or routing highly depends on the particular application. Although primarily targeted at applications requiring longer maximum distances between nodes, the router introduced in this document can also help application that require an overall higher throughput/bandwidth than the maximum 1Mbit CAN offers.

Embedded Systems Academy
50 Airport Parkway
San Jose, CA 95110
www.esacademy.com

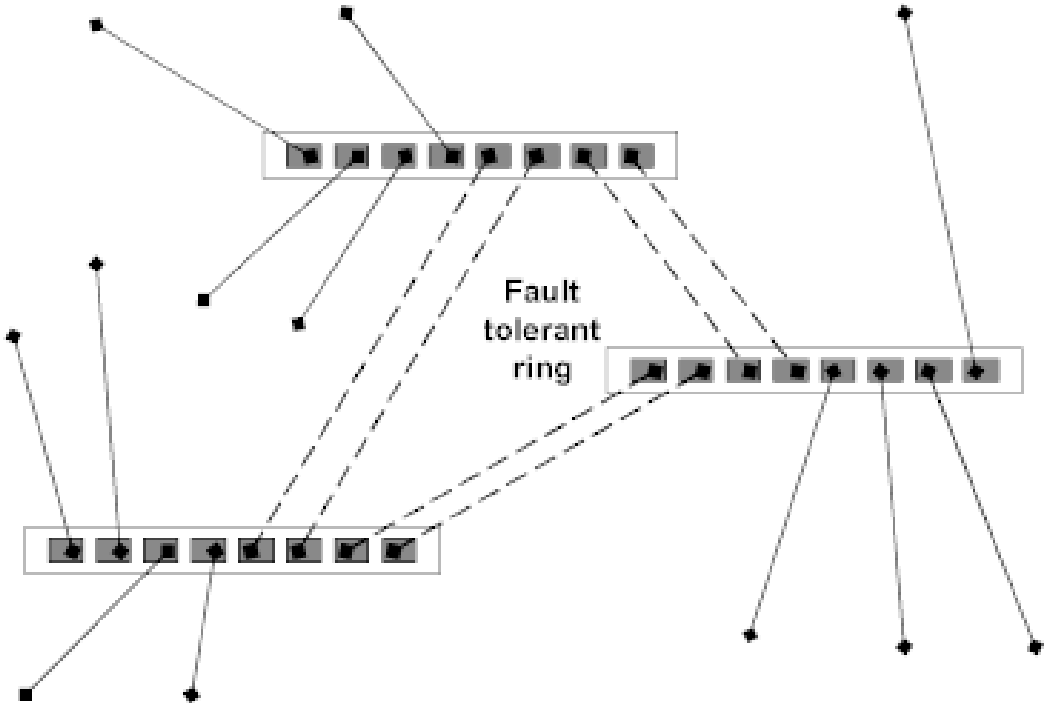

Figure 7 – Flexible, fault tolerant network topologies

Olaf Pfeiffer
P: (408) 910-7899
opfeiffer@esacademy.com

Christian Keydel
P: (408) 910-8418
ckeydel@esacademy.com

Andrew Ayre
P: (520) 885-7438
aayre@esacademy.com