Using the Network as Bus System for Long Discharge Data Acquisition and Data Processing

Ch. Hennig[a,*] , T. Bluhm[a], P. Heimann[b], H. Kroiss[b], G. Kühner[a], H. Kühntopf[a], J. Maier[b], M. Zilker[b]

[a] *Max-Planck-Institut für Plasmaphysik, EURATOM Association, Teilinstitut Greifswald, Wendelsteinstraße 1, 17491 Greifswald, Germany*
[b] *Max-Planck-Institut für Plasmaphysik, EURATOM Association, Boltzmannstraße 2, 85748 Garching, Germany*

Abstract

Data acquisition in shot-based fusion experiments is characterized by a large amount of data collected within a short time. The stellarator experiment W7-X will produce a considerable higher amount of data in a continuous and steady-state operation within half an hour. Therefore it is not sufficient to wait until all data has been acquired and stored in the database before being able to view and analyze data. Continuous and robust data transport is needed for (1) a distributed data monitoring system to visualize trends, (2) continuous data acquisition of data available at the network and (3) message signaling.

The paper introduces a **network channel** concept covering a system-wide available data description, half sync/half async communication pattern, data serialization, identification and matching of data. It is focusing on the IP Multicast/UDP implementation for the data monitoring system and presents results of performance measurements. Suitability and limitations are discussed and an appropriate answer to the question "When is IP Multicast/UDP reliable?" is given.

Keywords: Wendelstein 7-X; Network Communication; Long Duration Discharge; Data Monitoring, IP Multicast, UDP

1. Introduction

The fusion experiment Wendelstein 7-X will put high demands on data acquisition: Data taking will be the task of about 300 PCs. The experimental phases will last about half an hour.

It is planned to use the LAN with commercially active components for periodic and asynchronous communication. Fields of application include the data monitoring system, continuous acquisition of data available at the network and message signaling.

Therefore a general and easy-to-use network channel concept has been introduced. In contrast to existing TCP/IP based data pull techniques, the network is treated as a bus system. For data monitoring the

* Corresponding author. Tel.: +49-3834-88-2410
fax: + 49-3834-88-2509;
E-mail address: Christine.Hennig@ipp.mpg.de (Ch. Hennig).

underlying protocol is IP Multicast/UDP in a data push model with modules sending periodically and modules joining and listening. No additional protocol overhead is added because the data structure is globally defined for all modules in a common available database. Plug in of other hardware, like PCI or field bus systems, is supported and has been proven for a shared memory hardware.

2. Fields of application for fusion devices

At W7-X are planned two Ethernet based networks: the LAN and a dedicated Ethernet network (**rtNet**) for control purposes [1].

2.1. Data Monitoring

For long running fusion experiments an online data monitoring system [2] is indispensable. To achieve this, (1) acquiring components send a representative portion of the measured data to the network, (2) a server reads, processes and sends processed data ready to visualize and (3) all clients who are interested in this data receive it.

2.2. Network Data Acquisition

Components of the Device Control System [3] contain measuring equipment inside PCs with a real time operating system to control the device within milliseconds. They use rtNet as a real time bus system [1]. If data has to be exchanged, the component sends it to the rtNet at high speed.

More data producers are sensor components with LAN access, like PLCs, that send data to the LAN on timescales in the range of seconds.

Thus, acquiring data already spread at the network is advantageous. The data acquisition [4] collects the data from the network using the channel concept (chapter 3) instead of doubling the measurement.

2.3 Commands and messages

Interaction between the central control and subordinate system components will be done by control commands sent to the network. Moreover, system components have to periodically send status messages.

3. The channel-concept for communication

The network data communication described in chapter 2 is done via the network channel concept. Every data signal sent to the network is treated as an experiment-wide global signal. For every signal a connectionless channel is created per partner.

3.1. Global definition and description of a data signal

A list of all signals is available in the configuration database of the experiment. The description of a signal is defined per sender. It contains all information necessary for the sender as well as the receivers. As an example, a typical description of a PLC signal is given:

- **name** = “DataBrick1” (a human readable name)
- **refProducer** = “PlcCryo” (reference to sender)
- **netMedium** = “StdUdp” (logical medium for transport; here LAN)
- **dataFormat** = "2 float, 9 byte, 7 int" (type and format of data)
- **address** = {“239.255.131.130”, 3004} (here IP Multicast address and UDP port)
- **id** = 95 (for identification)
- **netType** = 0x4000 (for identification)

3.2. Writing and reading data

If a partner has to **write** data it immediately sends it to the network. **Reading** works as follow: The partner needs to listen to the network addresses specified in the channels it has to scan. At the hardware layer an interrupt signals the arrival of data. The data has to be queued until the service layer is ready to read. Therefore the half sync/half async pattern [5] is employed (Fig. 1).

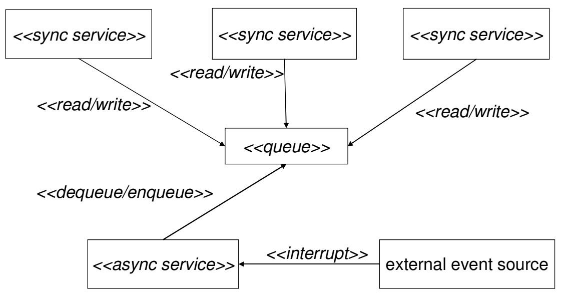

Fig. 1. The half sync/half async pattern [5]

3.3. Implementation of OSI layers 2 and 3

The layer 3 implementation is based on IP Multicast/UDP using the operating system independent Java multicast socket API which is not free of bugs ([6]; search for Multicast). If listening on different multicast addresses but same port any unicast traffic to that port arrives at a **random** multicast socket. Therefore either it needs to be guaranteed that no **unrequested** unicast traffic is using the same port or it is possible to filter unwanted data (section 3.4).

The layer 2 implementation based on MAC multicast is written in C++ and wrapped to Java via Java Native Interface. MAC multicast hardware filtering is enabled at the network interface card. The implementation for Windows is using winpcap [7], the only documented facility to send layer 2 packets. The API is well documented. The hardware filtering works fine ([8]; search for OID_802_3_MULTICAST_LIST). A Linux implementation is based on packet socket and directly using the socket link layer API. Complete control over the hardware filtering was not possible. If the number of MAC multicast addresses is larger than the memory the operating system puts the card automatically in a hashing mode.

3.4. Identification, filtering and security

Any receiver reading from a channel gets **everything** that is coming in at the specified address. This may include misdirected or broken data as well as LAN intrusion attacks. Identification of data including removal of unrequested data is necessary.

A couple of match criteria are part of the signal description: address, id and netType (section 3.1). Their matching, combined with checking the length and checksum of the frame, allows primitive filtering and security protection.

3.5. Multi homed hosts

Some PCs work as a bridge between the separated Ethernet networks LAN and rtNet. They need two network interface cards working with different IP addresses. For Windows systems it is necessary to configure only one default gateway per machine ([8]; Microsoft tech tips; search for multi homed host).

Multicast implementations for multi homed hosts require network hardware information: At which network interface has the PC to send or receive the multicast traffic? This host-specific mapping of logical networks to hardware devices needs to be added to the configuration in the database for every partner.

3.6. Serialization and de-serialization

Marshalling data to the network needs a serialization into a byte array. To save bandwidth and because the data in a channel has always the same format, a simple serialization is in use. In contrast to web technologies like Service Oriented Architectures, the type definition is not written into the byte array, only the “raw” data itself. No protocol is added. The format specification is contained in the description.

4. Multicast Tests

In this section the focus will be the IP Multicast/UDP implementation that has been developed and tested for data monitoring (section 2.1). It pointed out some losses in our prototype. Moreover, experiences from other fusion labs suggested testing robustness and scaling. The following questions will be addressed by the tests:

(1) How reliable is UDP in connection with IP Multicast? How much loss occurs and under which circumstances?
(2) Does a pause between send cycles guarantee delivery of data?
(3) Does Multicast scale with respect to the…
  - (3a) Size of data? (including the effects of UDP fragmentation)
  - (3b) Number of senders and receivers?
  - (3c) Number of multicast groups?

(4) Is there a difference between OSI layer 2 and 3?
(5) How does the normal network traffic affect the multicast traffic?

4.1. The test bed

Two Windows PCs running Microsoft XP with network interface cards SMC EtherPowerII are connected to a layer 2 Foundry BigIron 8000 switch. The backbone consists of two layer 3 Foundry BigIron 8000 switches; DVMRP enabled. Thus, two “hops” are between the PCs. The bandwidth in the backbone is 1 Gbit/s. The PCs in the office net are connected with 100 MBit/s (Fig. 2). The test bed is part of the LAN and all usual LAN traffic is present. All IP Multicast/UDP packets are transmitted with TTL=16 (local area scope).

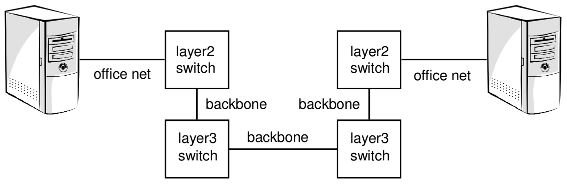

Fig. 2. The topology of the network tests

4.2. Test Scenarios

To test connectionless network performance, two scenarios have been used [9]. The **see-saw** scenario is a round-trip transaction between two partners in a cycle: Partner A sends, Partner B receives, Partner B sends, Partner A receives and so on. The **one-way** scenario matches the fields of application at W7-X. One sender sends periodically. One or many receivers receive. All receivers have a java Thread.sleep (1000) timeout which has been measured to be nearly 2 ms in average. All tests have been repeated about ten million cycles.

4.3. Test parameters

A set of test parameters (A) to (G) is defined to match the questions (1) to (5) in ascending order (Table1). One parameter is varied per question, all others are in their **default** setting (marked bold).

(A) test scenario {**one-way**; see-saw}
(B) pause between transmissions {**1 ms**; 10ms} (one-way scenario only)
(C) number of bytes in a datagram packet {**100**; 50,000*} (*layer 3 only)
(D) number of receivers {**1**; 96}
(E) number of multicast addresses in use {**1**; 100}
(F) OSI layer {**3**; 2}
(G) network load {**normal**; high}

4.4. Remarks and results

(A) Results from see-saw and conclusions to one-way
The see-saw scenario showed up very promising results. All 10,000,000 packets have been successfully received. The sum of both directions took approx. 0.48 ms. Therefore, a 1 ms pause between transmissions for the one-way scenario seemed to be appropriate.

(B) Varying the pause between sends
With 10 ms pause 9,958,406 packets had been received. This is a loss of approx. 0.5%. With 1 ms pause 9,690,541 packets had been received, the loss increased to approx. 3%.

(C) Varying the number of bytes
Sending 50,000 bytes with 1 ms pause would exceed the network capability. Therefore, in this test a 10 ms pause is applied. 8,326,563 packets had been received, thus the loss is approx. 17%. All UDP packets had to be fragmented into 34 Ethernet frames (max. 1500 byte). The loss is in the range as expected: 0.5 (loss in test B) * 34 (number of frames) = 17%.

(D) Varying the number of receivers
96 receivers have been incorporated. Only 21 of them did not have premature timeout. At 70 receivers, a timeout greater than 1,000 ms (which is 1,000 times the pause) occurred after 14 seconds and at 5 receivers the timeout occurred after 7 minutes. This leads to the supposition that there is a congestion at the switch or the receivers
17 out of 21 successful receivers reached losses less than 1%. The overall loss rate of the 21 receivers is about 4.8% and ranges from 0.004 to 44.8%.

(E) Varying the number of Multicast addresses
When sending to 100 different addresses instead of a single one 9,999,200 packets had been received. This is a loss of approx. 0.008%.

(F) Varying the OSI layer
For layer2 the test bed is modified. There are no hops between the participating PCs possible. In the test 9,984,042 packets had been received. This is a loss of approx 0.2%.

(G) Varying the network load
In this test a high network load has been produced by subsequently copying a huge file (5 Gbyte) over the

test bed. The additional load was about 18.2 Mbit/s.

In the tests 9,921,371 packets had been received. This is a loss of approx. 0.8%.

Table 1
Summary of the IP Multicast/UDP test results

question ⇔ test*	**Packets received**	**Loss in %**
default settings	9,690,541	3.095
1 ⇔ A	all 10,000,000	0.000
2 ⇔ B	9,958,406	0.416
3a ⇔ C	8,326,563	16.734
3b ⇔ D	average 9,517,272	4.827
3c ⇔ E	9,999,200	0.008
4 ⇔ F	9,984,042	0.159
5 ⇔ G	9,921,371	0.786

*The matching of tests to answer questions (1) to (5) by varying parameter (A) to (G)

A summary of all tests is shown in Table 1. With the default parameters, no data losses have been predicted because the test data amount is small compared to the network bandwidth. If reproducing the tests the losses vary. Further tests are necessary to clarify the reasons for the data losses detected.

One hypothesis is that the robustness of multicast depends on the performance of the active component and the implementation of the multicast routing protocol. Test (D) buttresses this. The typical commercial application using IP Multicast is Video Conferencing. Possibly the switches are optimized for that purpose.

Performance becomes poor especially if data has to be fragmented or the sending rate is very high. Technologies in use by commercially available video camera server using IP Multicast/UDP with streaming technology could be an alternative for the monitoring of video data.

5. Summary

A flexible channel concept has been introduced for communication between partners. It is independent of the underlying hardware that is treated as a network with senders pushing data and receivers listening.

Three fields of application for continuously operating fusion devices have been introduced: (1) data monitoring, (2) network data acquisition and (3) signaling between system components.

The data monitoring system will be an indispensable application for long duration fusion experiments. For this it is planned to use IP Multicast/UDP on standard network hardware. Performance tests tests show limitations for large size data, complex network infrastructure and many receivers. Further tests are necessary to clarify the reasons for the data losses detected.

It is already widespread that manufacturers integrate network capabilities into sensor components. They push data to the network and other subsystems have to collect and store them in the database. For this purpose network data acquisition plays an important role.

The channel concept is flexible enough to allow substitution or addition of concrete implementation technology.

References

[1] Laqua H., Niedermeyer H. and Willmann I. Ethernet based real time control data bus. IEEE Trans. Nucl. Sci. 49 (2002), pp 478-482.

[2] Hennig Ch. et al. A concept of online monitoring for the Wendelstein 7-X experiment. Fusion Engineering and Design 71 (2004), pp 107-110.

[3] Laqua H. et al. Control system of WENDELSTEIN 7-X experiment. Fusion Engineering and Design 66-68 (2003), pp 669-673.

[4] Heimann P. et al. Status report on the development of the data acquisition system of Wendelstein 7-X. Fusion Engineering and Design 71 (2004), pp 219-224.

[5] Schmidt D. C. et al. Pattern-Oriented Software Architecture Vol. 2. Patterns for Concurrent and Networked Objects. Wiley, Chichester, 2000.

[6] http://bugs.sun.com/bugdatabase/index.jsp.

[7] http://winpcap.polito.it/.

[8] http://msdn.microsoft.com.

[9] Irey P. M. et al. Techniques for LAN Performance Analysis in a Real-Time Environment. Real-Time Systems 14 (1998) pp. 12-44.