Moving the Control from Senders to Receivers

Teemu Rinta-aho

***Abstract*—Internetworking in the current form provides a possibility for all hosts connected to the Internet to send data to any other host by using the destination host's IP address as the destination address of the IP packet. This send-receive paradigm is not only the base for IP networks, but also for most other current network technologies, such as the Ethernet. The publish-subscribe networking is a completely different paradigm, which can be used to design a network architecture. It is based on the idea that data is more important than the network end-points. By selecting the publish-subscribe paradigm instead of the send-receive, the control of what is being transmitted in the network is moved from the sender to the receiver.**

***Index Terms*—Future Internet, Post-IP, Publish-Subscribe, Network architectures**

I. Introduction

Internet is facing several problems. One major problem is the shortage of available IP addresses in IPv4. A solution to that is to upgrade the IP protocol from version 4 to version 6. Another huge problem is the unwanted traffic – not only the unwanted SPAM e-mail, but also the unwanted IP packets sent by e.g. DDoS attackers. Solutions to these problems include SPAM filtering with different methods and installing firewalls that block IP packets with certain rules.

Many, if not all, of these problems are caused by the fact that the IP network is designed by the send-receive model. In send-receive the sender has the control: it can select when and where to send a message and the network is designed to help the sender to reach its goal – to get the message to the specified destination. As the sending is cheap, this attracts some people to send messages to recipients who are not willing to receive such data.

There are many efforts on various fields of research and engineering to overcome the shortcomings in the IP architecture. On one hand IP is extended or patched with several new protocols and extensions to support the changed computing and communication environment, such as Mobile IP [1] and firewalls. On the other hand, there are more radical efforts that extend the networking architecture, such as HIP [2] and DONA [8].

This paper examines a completely different model that looks a promising choice to build internetworking upon, which is called *Publish-Subscribe* (later 'PubSub') model. This new model is essentially moving the control from the sender to the receiver, and keeping in mind that "the receiver is the king" and the data is more important than the end-points.

The section II of the paper will provide a brief look on the background of PubSub, in section III some ideas for the PubSub internetworking architecture are presented, in section IV are some first prototyping results, and in section V the paper is concluded.

II. Background

A. The Publish-Subscribe Paradigm

Publish-Subscribe (PubSub) is an asynchronous messaging paradigm where publishers of data don't send the data (publications) to specific subscribers. Actually the publishers don't even necessarily know if there are any subscribers to the publications they publish. Each publication has an identifier and the subscriber can subscribe to such a publication using the identifier even without knowing if the publication has yet been published or by whom it is or will be published. This is called *space decoupling* [3].

Subscribers and publishers do not need to participate in the publish-subscribe process at the same time. A publisher may create a publication, and then go off-line, and assuming that the network is storing the publication, the publication may be subscribed by a subscriber any time. This is called *time decoupling*.

Similarly, if there are many publishing and subscribing events in the network, they don't need to be synchronized, i.e. the subscriber doesn't need to block if it has subscribed something that is not yet published, nor has the publisher need to wait for the publication to be "consumed" before publishing the next message. This is called *synchronization decoupling*.

While the space, time and synchronization decoupling add flexibility into the network, they are only desirable for applications without real-time constraints on data delivery, such as file transfer or e-mail, but not for e.g. VoIP.

As the subscribers usually don't want or can't be provided with all the publications in the network, the publications need to be filtered. There are two ways to do this: topic-based or content based.

In content-based PubSub the filtering of messages delivered to a subscriber has a wider filter, i.e. the publishers can publish many types of data, and if the publication matches the attributes of the filter, the message is delivered to the subscriber. It is up to the publisher to classify the attributes of the publication so that the subscriber gets the correct data.

Manuscript received November 22, 2007. Presented on "Postgraduate Course on Networking Technology" (S-38.4030), Networking laboratory, Department of Electrical and Communications Engineering, Helsinki University of Technology.

T. Rinta-aho is with Ericsson Research, Oy L M Ericsson Ab, 02420 Jorvas, Finland. E-mail: teemu.rinta-aho@ericsson.com.

In topic-based PubSub the filtering is narrower; the publications are filtered by their name. It is up to the subscriber to find out the names of the publications it wants.

B. Related Research

Most research efforts so far have been concentrating on the content-based PubSub such as the JEDI [4] and the Java Message Server [5], and like them, many are assumed to be run on top of the current Internet protocols.

Some examples of protocols or architectures that use some PubSub-like ideas and are used in practice are mostly in the field of P2P file sharing, for example Kazaa [6] or BitTorrent [7]. First of all, in those systems, the content of the data is the most important thing. Data is not identified by the file name or location, but an ID, which is usually a hash of the data. Receiving the hash and the data the receiver can check that the data has been received correctly. Secondly, it is of no interest to the subscribers of the data who is or was the publisher, as long as they get what they wanted. Using these principles, it is possible to share the storage and/or transmission costs between the nodes participating in the P2P network.

Data Oriented Network Architecture [8] is a clean slate redesign of the naming architecture of the Internet. It provides persistence, availability, and authenticity to the data allowing the data to be cached, or moved without changes to the method that can be used to reach it.

The names in DONA are self-certifying and of the form P:L, where P is a cryptographic hash of the principal's (publisher's) public key, and L is a string chosen by the principal. The architecture supports two primitives: FIND(P:L) and REGISTER (P:L). When a client sends a FIND(P:L) request, the DH either locates the location of the nearest copy of the publication in its registration table and forwards the request to it, or forwards the request to its provider DH. When a DH receives a REGISTER(P:L) it puts the location information in its cache and forwards the request to its provider and peers. Thus, the DH of tier 1 AS will have location information of all registered publications.

III. A PubSub Internetworking Architecture

The PubSub architectures proposed so far are assumed to be run on top of current IP networks. The goal of the upcoming EU 7th FP project PSIRP, however, is to replace the IP with a clean slate PubSub architecture. This requires rethinking of everything that forms an internetworking architecture: naming, routing and forwarding. Besides internetworking, we feel that it is worthwhile to use PubSub also on local links and even in the node-internal architecture. This will require a new way to attach to the network, new security mechanisms and, most importantly, it requires a new way of thinking. In this section some of the preliminary thoughts that could be used as the basis for the new architecture are presented.

A. Motivation

One of the biggest problems today is the unwanted traffic at the receiving nodes. This is traditionally counterattacked by installing firewalls that filter the traffic with certain rules. However, this only blocks some of the unwanted traffic, for example DDoS on IP layer for certain port numbers. It is still possible to DDoS a web server, for example, because it is not possible to block TCP port 80 to a publicly available server, e.g. www.cnn.com.

Another observation that can be made in current wireless and wireline network technologies today, is that many of them are basically shared broadcast mediums with a MAC protocol on top providing a virtualization of a per-node resource. When one node is transmitting, others have to wait for their time slot Instead of just waiting they might receive the data being transmitted by other nodes. Could the PubSub model help increase the total capacity of a network?

If the API needs to be rewritten due to a radical change in the way applications will use the network, should we also rethink other operating system components and their role in the future? It might be feasible to put e.g. network and local disk behind the same API?

B. Architecture

1) Identifiers

Our proposed architecture falls into the topic-based PubSub category. A publication has two identifiers: a public *Subscription ID* (SubID) and a private *Publication ID* (PubID). They are bound to each other by some cryptographic (hash) function. The PubID is only known by the publisher (the owner) of a publication, and the SubID is publicly known ID that can be used to subscribe to the publication. If the PubID depends on the content of the publication, then it is possible to form self-certifying publications where the subscriber can check that the publication originated from a legitimate publisher and that the data has not been tampered during the transport. With self-certifying publications it doesn't matter anymore where the data is received from. Remember that the application subscribed to data, and there is no need for an end-to-end connection.

It is left for further research what is/are the exact method(s) to form the PubID, and if PubID has any structure or is it a flat naming space.

2) Primitives

There are only two primitives: *publish* and *subscribe*. Actually, it can be argued that there is only *one* primitive: publish, and subscribe is built on top of it – subscribing means *publishing a subscription.*

The publish primitive is a process of binding *publication metadata* to the actual published data. Publish has a scope which defines the metadata and the required *compensation mechanisms*.

3) Publication Metadata

Publication metadata contains information required to store, transmit and authenticate a publication, such as type, PubID, SubID, lifetime and scope. A publisher may use e.g. PKI to bind the metadata to the application data presented by a publication so that it can prove the ownership of the metadata during the lifetime of a publication.

4) Compensation Mechanisms

If a publisher wants to publish in the network, it needs to compensate for the resources needed for the publication. If the scope of the publication is node-local, then obviously a compensation mechanism is not needed – the publish operation will succeed if there is enough free memory or disk space to use for storing. Publishing in your own (home) network is a bit more complex, but in most cases it is adequate to just be authenticated to the network. Publishing outside your own network or globally requires more complex compensation mechanisms depending on the business models used. For example, if you want to publish a web page through your ISP, your monthly rate may cover certain amount of storage in the ISP network, and then this business agreement for network usage needs to be somehow referred in the publication process for it to succeed. If you publish the web page on your own home server but want it to be available outside your home network, then you only need to publish the related metadata through your ISP and compensate for its storage and possible routing and directory service costs.

5) Authentication Mechanisms

Because the data is the main identifiable object, it is often enough to authenticate data. In some cases, however, a node publishing metadata needs to be authenticated. This can be done with PKI, but it may become too heavy for publications that update frequently. An option to PKI could be e.g. TESLA [9]. TESLA could be used for example to authenticate periodic link information sent by the local router.

6) Functional Model

The PubSub internetworking architecture consists of three different roles that a node may have: *a publisher*, *a subscriber* and *a sprouter*. The name sprouter is chosen to differentiate the node from an IP *router*.

When a publisher publishes, it announces to the local sprouter that it has a certain publication (assuming the publication is not of local scope only). The sprouter stores the metadata and if it receives subscriptions, it will send the data to the subscribing subscriber or sprouter (see Figure 1). If the local sprouter is caching the publication, then further subscriptions to the same publication will be served directly from the cache without consulting the publisher. It is possible that the publisher doesn't even know of the subscribers.

The network architecture consists of three separate functions: *rendezvous*, *routing* and *forwarding*. Rendezvous is the function where the subscription and publication's location information meet. Once the origins of the subscription and the publication are known, the routing function is consulted for a route between these two locations. After that the forwarding path is set up and the forwarding plane takes care of the actual transmission of the data.

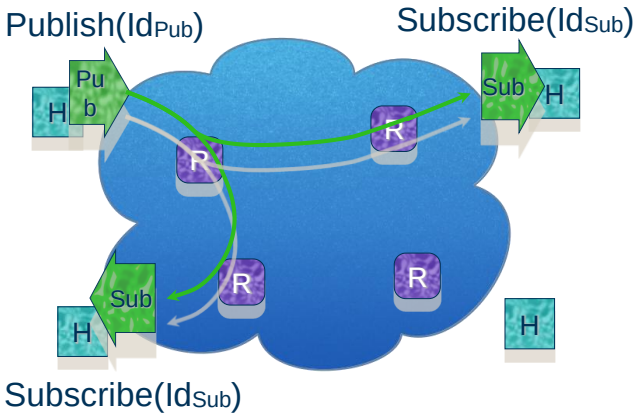

Figure 1: PubSub functional model

C. Use Cases

File sharing is the simplest application for the PubSub internetworking architecture. While the data itself is identified and self-certifying, the network can use caching to reduce load. Because the data is being self-certified, there is no need to receive the data from a specific node. There are problems to be solved, though. One of them is the fragmentation of large publications. In our current thinking the link layer frames themselves are publications with a maximum size, there needs to be a way to group several frames together which would then represent the "higher layer" publication. Another problem which is not really specific to the file sharing use case is the implementation of rendezvous and routing, i.e. how to do file sharing in large scale networks.

Two-way connections can be constructed by using two publications, one to each direction. An application that wants to listen for incoming connections can periodically publish an "invitation publication". Anyone can subscribe this invitation, containing a puzzle. Solving the puzzle reveals the actual channel which will lead to the listening application. The channel is the SubID that the "listening" host is subscribing and where data can be published by the initiating host. Connecting application can send the SubID of the other direction during the connection setup. Using a puzzle is necessary to prevent DoS attacks.

Streaming brings transport issues into the PubSub architecture: how to support versioning of application data, how to do congestion control, how to implement a reliable transport over unreliable links. As we envision that all transport is multicasting, these problems are common with the current multicast transport research and not all are "caused" due to the PubSub. These issues are considered in the ongoing and future research.

IV. PROTOTYPE

We have implemented a PubSub prototype in our lab. It is running on Linux and it is implemented completely in user space. Everything above the link layer is implemented "from scratch", i.e. no existing protocols are being used. The prototype consists of a library implementing the PubSub API, a PubSub daemon and some test applications written in C and Java (see Figure 2). The daemon is using libpcap and libnet to receive and send Ethernet frames. However, the source and

destination fields of the Ethernet frame are ignored, and all sent frames use the broadcast address as the destination.

A. Architecture

The internal architecture of a node itself is also following the PubSub paradigm. The components of the protocol stack are not in a stack in the traditional sense, but they use a blackboard approach to access the publications. The blackboard in this case is a common directory which holds the stored publications. The directory can be a memory file system to prevent delays introduced by frequent disk accesses.

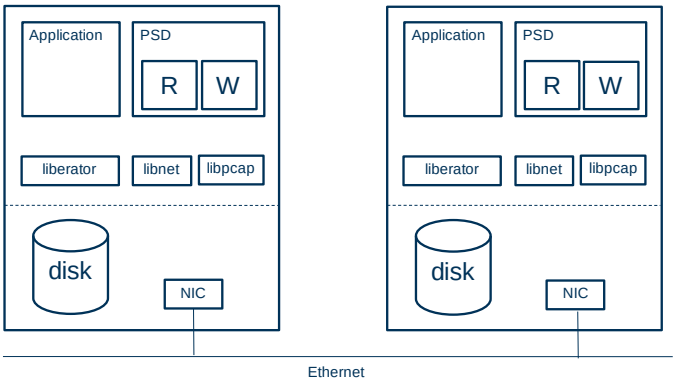

Figure 2: Prototype Architecture

B. Operation

When an application publishes a file, it calls a library function *create* that creates two files per publication: one for metadata and one for the data coming from the application. It maps these files to the memory space of the calling process and fills in the metadata and data. Once ready, the call to the *publish* function will bring the new publication visible to other applications and the PubSub daemon. Depending on the metadata, a special publication called *publication list* may be updated. This publication list is periodically broadcast on the Ethernet link so that other nodes are aware of the available publications.

When an application calls *subscribe* function with the SubID of the subscribed publication, the library either maps the cached publication (if it is already available) and returns a pointer to the caller process or it creates a new publication of type *subscription* into the disk. PubSub daemon will notice this subscription and broadcasts it to the Ethernet link(s).

When receiving a publication from the network, the daemon checks whether it is a subscription or subscribed data. If it is a subscription, it will broadcast the publication back if the publication is stored locally. If the received publication is data, it will be stored on the disk and it becomes available for the application.

In a sprouter node, the daemon is run with ”sprouter” flag set. In this case, it will store subscriptions sent by other nodes. If the subscribed data is already cached locally at the sprouter, it will be delivered to the subscriber. Otherwise the sprouter subscribes to the data on other links and after the data has been received, it forwards the data to the original subscriber. It will also forward the broadcasted publication lists on other links, so that all nodes in the connected networks are aware of the available publications. The nodes, however, are not aware of the presence of the sprouter; it is basically a transparent proxy.

The actual forwarding of a large publication may require fragmentation. An Ethernet frame can hold up to 1540 bytes of data. Our PubSub header currently occupies 32 bytes (1 byte for publication type and 31 bytes for SubID). In the current prototype the fragmentation is solved by introducing a new type of publication: *fragment list*. A fragment list publishes the SubIDs of *fragments* (see Figure 3). When a subscriber receives a fragment list, it needs to subscribe to all fragments. After receiving all fragments, the daemon on the receiving side can create a copy of the original data from the fragments. There is a simple re-subscribe timer to recover from lost fragments and a simple rate limiting timer to prevent from causing too many collisions on the Ethernet.

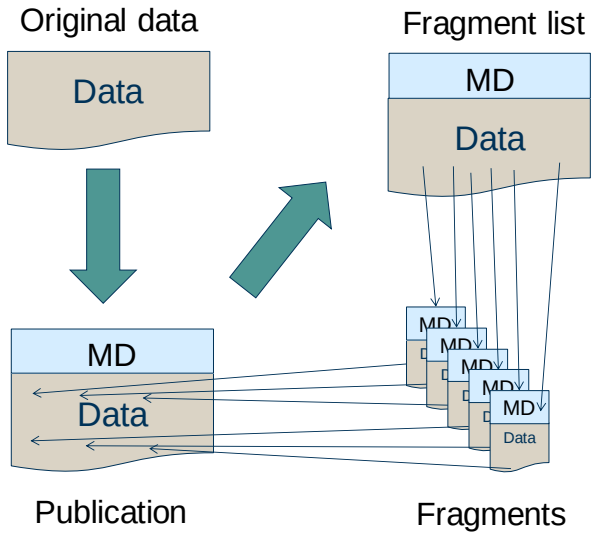

Figure 3: Fragment list

C. First Test Results

The prototype has been run with the following setup: two Ethernet links which are connected by a sprouter. On Link 1 there are two subscribers and on Link 2 there is one publisher (see Figure 4). All nodes are using a picture sharing application which can be used to publish and subscribe e.g. JPEG files. Currently we can publish and subscribe to a number of files and the sprouter is supporting links with different MTUs by creating separate fragment lists on each link. We also have caching function on the sprouter: once the file has been published on Link 2 it will be forwarded to the subscribers on Link 1 from the sprouter cache – even if the original copy is lost at the publisher.

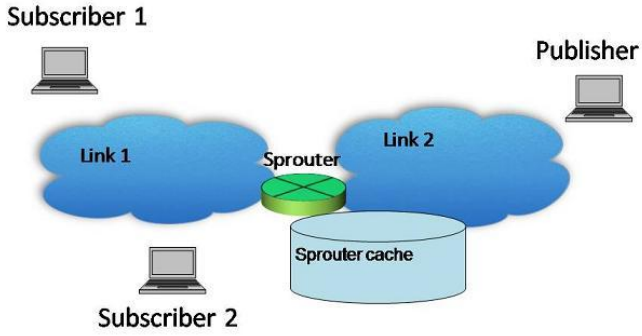

Figure 4: Prototype network

D. Next Steps

Next steps in the prototype development will include support for streaming applications, networks of multiple sprouters, network attachment, protection of metadata and self-certifying data.

V. Conclusion

This paper provided an overview of the Publish-Subscribe networking paradigm, which is a radically different approach to solve the shortcomings of the current Internet architecture. After some initial thinking and prototyping in our lab we have not faced any problems that would seem a reason to stop developing the initial thoughts towards a more refined proposal.

The proposal for a project Publish-Subscribe Internet Routing Paradigm (PSIRP), has been accepted in the first call of the ICT programme of EU's 7th Framework Programme. The project will launch in the beginning of 2008 and it will be a joint effort of eight partners for two and a half years.

Acknowledgment

I would like to thank Pekka Nikander, Petri Jokela, Mikko Särelä, Christian Vogt, Jan Melén and Jukka Ylitalo. Their work and many of our discussions have been an invaluable input to this paper.

References

[1] D. Johnson, C. Perkins and J. Arkko, “Mobility Support in IPv6”, RFC 3775, IETF, June 2004
[2] R. Moskowitz and P. Nikander, “Host Identity Protocol (HIP) Architecture”, RFC 4423, IETF, May 2006
[3] P. Eugster, P. Felber, R. Guerraoui and A. Kermarrec, “The Many Faces of Publish/Subscribe”, ACM Computing Surveys, Vol. 35, No. 2, June 2003, pp. 114–131.
[4] G. Cugola, E. Di Nitto and A. Fuggetta, “The JEDI Event-Based Infrastructure and Its Application to the Development of the OPSS WFMS”, IEEE Transactions on Software Engineering, Vol. 27, No. 9, September 2001, pp. 827-850
[5] Java Message Server, http://java.sun.com/products/jms
[6] Kazaa, http://www.kazaa.com/us/index.htm
[7] BitTorrent, http://www.bittorrent.com/
[8] T. Koponen, M. Chawla, B. Chun, A. Ermolinskiy, K. H. Kim, S. Shenker, and I. Stoica, “A Data-oriented (and Beyond) Network Architecture”, In Proceedings of SIGCOMM '07.
[9] A. Perrig, D. Song, R. Canetti, J. D. Tygar and B. Briscoe, “Timed Efficient Stream Loss-Tolerant Authentication (TESLA)”, RFC 4082, IETF, June 2005.