STUDIA INFORMATICA 2013
Volume 34 Number 3 (113)

Marcin DĘBICKI
Tyszkiewicz High School, BWS
Zdzisław ONDERKA
Akademia Górniczo-Hutnicza,
Department of Geoinformatics and Applied Computer Science

STRUCTURE AND DATA TRANSFER ORGANIZATION IN EDEN SERVERLESS NETWORK

Summary: The paper presents the model of the new serverless computer network called EDEN. The basic network components are defined and their functionality. The specification of the package and subpackage header construction, the addressing problem, hypernode searching algorithm and the organization of data transfer are presented. The comparision with the OSI and TCP/IP models is discussed. At the end, the test of hypernode searching is described - problem from which mainly depends on connection establishment and data transfer.

Keywords: serverless network, addressing, package, data transfer.

STRUKTURA I ORGANIZACJA TRANSFERU DANYCH W SIECI BEZSERWEROWEJ EDEN

Streszczenie: W artykule przedstawiono model nowej bezserwerowej sieci komputerowej o nazwie EDEN. Zdefiniowano podstawowe elementy sieci oraz ich funkcjonalność. Przedstawiono definicję pakietów, podpakietów, budowę nagłówka podpakietu, problem adresowania, algorytm poszukiwania hiperwęzłów i organizację transferu danych. Model EDEN porównano z modelami OSI i TCP/IP. Na koniec, przedstawiono testy wyszukiwania hiperwęzłów – problemu, od którego głównie zależy ustanowienia połączenia i transfer danych.

Słowa kluczowe: sieć bezserwerowa, adresowanie, pakiet, transfer danych.

1. The EDEN Network

EDEN – *Enhanced Data Exchange Network* this is a theoretical model of the new network communication and organization, allowing the replacement of existing computer network operating in a client-server architecture (for example TCP/IP) by the modern network platform based on the public services.

The purpose of this type of network design is to facilitate the computer network organization and optimization of the connections between multiple hosts in order to maximize network performance (for example increased efficiency of data transfer using multiple routes). The network is assumed to be able to self-organize so that the administrator role has been reduced to oversee that this feature occurred according to a preestablished parameters. Another assumption of the EDEN network is the ability to easily run multiple network environments within a single host.

Most of the existing solutions for the serverless networks concern potential solutions for P2P networks such as security problems for specific data streaming [1], services such as chat systems [2] or descriptions of the best practices in P2P overlay building and P2P scheduling using examples of the deployed P2P applications [3] and how a serverless network can be formed with capabilities that enables further development on P2P applications [4]. Other works concern the problem of the serverless network file systems, to provide better performance and scalability than traditional file systems [5] or there are solutions of the variety of issues dedicated to RFID (*Radio Frequency Identyfication*) serverless networks [6][7]. However, the above-mentioned solutions does not define a new network architecture but use existing technology based on the OSI model.

The basic elements of the EDEN network components are defined as:

- **Node** – node is responsible for any network activity by one process that is assigned to the current node. This is a gateway between application and network itself. Nodes manage connections between applications, which means that if applications A and B are connected to each other then nodes representing those applications also must be properly connected together (directly or not directly). Each node has its own name, which must be unique within a given hypernode.
- **Supernode** – the type of node that may not represent any application. The supernodes constitute a cache mameory for data. It is a kind of proxy that can be used to reduce the network load. For example, if several nodes are connected to supernode X and X is connected to node Y (which is the source of data) and some nodes connected to X are downloading the some set of data from Y then data should be sent only once from Y to X (data will be stored in the resources of X). One of possible uses for supernodes is to cache data

that is requested by several different nodes. It doesn't matter if from single hypernode or from totally different hosts.

- **Hypernode** – this is an object responsible for:
  - the organization of the communication between processes running within the same network environment (each process running on the current system has to belong to exactly one hypernode),
  - the node management within one network environment. Hypernode can create and delete nodes or change their names.

  Each node and supernode belongs to some hypernode. Each computer connected to the network must have at least one configured and running hypernode. Hypernode must have a unique name. The hypernode creates kind of virtual environment for nodes. With such solution it is possible to reduce the number of used network addresses to necessary minimum (used only during establishing connection phase and host lookup phase).
- **Slot** – It is a virtual interface of hypernode. Each node and supernode in hypernode sees slot as another node with the exception that between two nodes/supernodes there can be one-way connections (or two-way), while between slot and node/supernode ther are always two-way connections. Slot represents physical network interface (like network adapter, Wi-Fi device, bluetooth adapter and so on). The physical network interface can be represented by only one slot in single hypernode but, if we have multiple hypernodes than all of them can use this interface (each of those hypernodes must have separate slot representing this interface).
- **CPI (*Collision Prevention Interface*)** - the lowest instance assigned to the only one network interface. Each physical and virtual network interface, which is used by at least one hypernode must have assigned exactly one CPI module. This module is responsible for transfer of received packages to the proper hypernodes, for which the received data are designed, is responsible for preventing packages or subpackages collisions and is responsible for the synchronization and the sending data queuing in case when one host has many hypernodes. It is also a gateway allowing connections between hypernodes. CPI module can handle exactly one network interface (any network device with a MAC address for example network card). CPI communicates with the data link layer of the OSI model to send or receive transmitted data.

  The schema of the CPI is as follows:
  1. Hypernode sends to the CPI module package ready to send,
  2. CPI module checks the package priority. Special packages e.g. when searching for hosts, authentication or data encryption setting between hypernodes, have high priority and are sent in the first place.

3. After sending the current packet CPI module is waiting for confirmation of packet delivery to the target CPI module.
4. Target CPI module receives the package and checks again its correctness. If error is detected target CPI sends an error notification message to the source CPI module. Source CPI sends the notification to the hypernode (it decided what to do with the package), and in case of interrupt of further sending the package is removed from the buffer. In case of package resending the CPI module continue from the step 2 (the data are already validated during the first attempt). CPI can locally increase the priority of the package to allow faster package sending. CPI, however, may not in this case modify the header (changes could not affected to the transfer to the next CPI module).
5. In case of correct package transmission to the target CPI module, it reads the hypernode name (to which the package is to be sent) and forward package to the relevant hypernode.

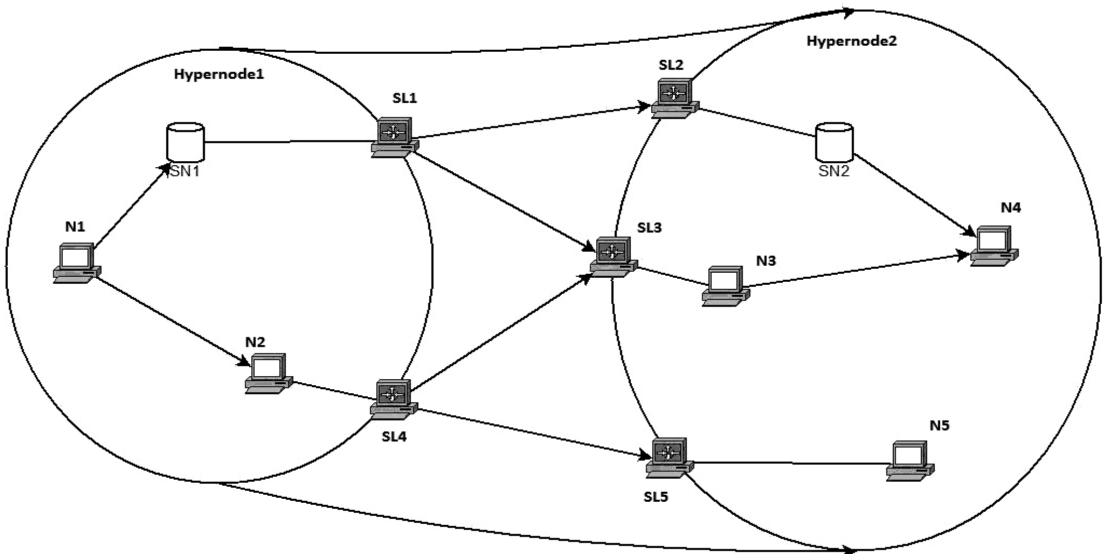

Fig. 1. Example of the EDEN network organization
Rys. 1. Przykład organizacji sieci EDEN

The example of the EDEN network is presented at the Fig. 1 (SN – supernode, N – node, SL – slot). The basic properties of such network are:

- Each hypernode must have at least one node to exist,
- Each slot represents one and only one configured network adapter,
- Hypernode connections are directional (it will be possible to create routing tables),
- Connections between two slots are directional. This direction must be the same as direction of connection between hypernodes,
- Connections between slot and node are not directional,
- There is no possibility to create multiple connections between the same two instances (nodes, hypernodes, slots),

- Connections between nodes/supernodes are directional,
- One node or supernode can be connected to many nodes/supernodes,
- One slot can be connected to many slots,
- One node or supernode can be connected to many slots,
- One slot can be connected to many nodes or supernodes,
- One hypernode can be connected to multiple hypernodes.

2. The Comparison with the OSI and TCP/IP Model

During the EDEN network design, principles for the design Internet protocols (RFC 1958) were not used, in particular the design compatible with the OSI reference model. One of the goals of the project was to simplify the network model without affecting its overall functionality. To achieve this it was necessary to look at the model from the perspective of network functionality, network protocols and specific objects that implement these protocols. For the EDEN network it is a simple task because there is a clear division of tasks between the different objects (such as the CPI or hypernodes).

In the OSI or TCP/IP model, each network protocol operates in only one layer. Similarly EDEN network, but due to the relationship between different protocols, the layers do not split due to the implemented protocols but due to objects in which these protocols are operate. Therefore, the comparison with the OSI and TCP/IP models does not reflect the characteristics of the EDEN network. For example, the equivalent to the IP protocol which, performs most of the work associated with communication for TCP/IP protocols, is part of the CPI module. Here the role of the CPI is limited and some of functions of this layer was moved to hypernode. In other words, hypernode goes beyond the transport layer. The approximate comparison of the OSI, TCP/IP and EDEN network is presented in Fig. 2.

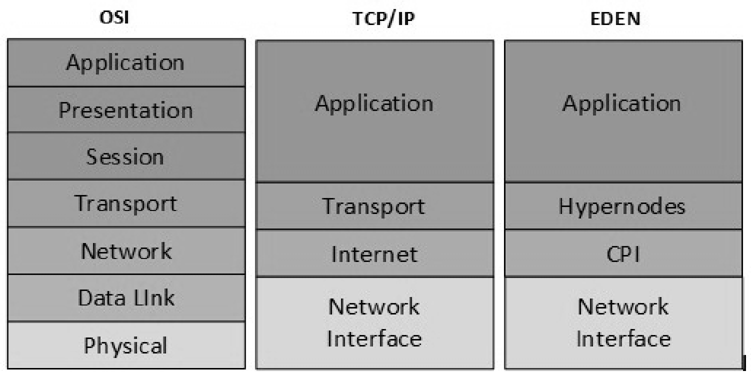

Fig. 2. The comparision between OSI, TCP/IP and EDEN
Rys. 2. Porównanie sieci EDEN z modelami OSI i TCP/IP

3. The Packages and Subpackages Structure

For the optimal data transmission in the network, it is important to define the appropriate package structure. In the current EDEN specification assumes that each package consists of its header and a series of so-called subpackages. The subpackage consists of its own header and contains the relevant data to be sent. The package header consists of the following fields:

- SrcAddr, TgtAddr – the source and the target address of data to be sent,
- PackageID – the package identifier (an integer greater than or equal to 0),
- PackageAck – the number that identifies package received by target host,
- ContentSize – the size of sending package,
- Checksum – the checksum for the package sent validation,
- OptionsData – additional parameters for {\sf Options} e.g {\sf connection_limit},
- Options – may contain additional options in the form of a set of relevant bits. The EDEN specification provides the ability to use of the following options:
  - host_search, host_search_completed – used in search hypernode algorithm,
  - connection_limit – used in the specification of the data transfer within the hypernode,
  - options for authorization and authentication of the hypernodes and for the CPI module,

The subpackage header consists of the following fields:

- SrcAddr, TgtAddr – the source and the target address of data to be sent,
- PackageID – the package identifier, to which the subpackage belongs,
- HChcks – the checksum for the subpackage sent validation,
- ID – the identifier, determines the position of the subpackage in the package,
- Ack – identifies subpackage that was received by destination host (it's ID),
- Options – the options which define the type of the subpackage (request or acknowledge) and may contain options not included in the packet header,
- SrcNode, TgtNode – the numerical identifiers specifying the source and the destination node, assigned to node by the hypernode,
- CSize – the coefficient that determines the maximum size of the subpackage content according to the formula: $2^{16} \times CSize$,
- Life – the subpackage life expectancy. Tha maximum value is 255. Every time the CPI module reads the header, this value is reduced by 1. When it reaches 0 the subpackage is dropped,
- SHN (*Source Hypernode Number*) – the hypernode identifier that contains the source node,
- THN (*Target Hypernode Number*) – the hypernode identifier that contains the target node,

- HdrSize – an additional header size. On this basis, it is possible to specify how many bits are still following the header (maximum 64 Kb),
- HeaderContent – additional part of the header, the size of which is determined by HdrSize. This content is optional and contains the path to the destination node according to:
  - In the case when, this field exists then the hypernode described in the THN and node described as a TgtNode must be included in the path. Otherwise the subpackage should be rejected as invalid;
  - In case when, using the path specified in the HeaderContent, it is impossible to explicit find the target node (heaving regard to the THN and TgtNode fields), such subpackage should be rejected as invalid,
  - In case when, in the contents of the path after the name of the target node there is additional content, then this content is then treated as a parameter that should be passed to the destination node (the parameter for the process assigned to the node).

Note that, it is possible for packet header to have up to 32 options defined and for subpackage header up to 16 options, there is still a lot of room for further expansion. This room should be used for expanding EDEN's flexibility i.e. by allowing more header content to be optional. This should be developed further in order to provide as much of behavior of TCP and UDP protocols as well as capability to provide behavior suitable to extend, support or replace protocols that don't need features of TCP yet, what UDP provides is not enough (for example UDPCP protocol).

4. Addressing and Hypernode Searching

The network address consists of five elements, with a total length of 128 bits (A.B.C.D.E):

- A – Region Identifier – first 16 bits of the address. Knowing this identifier it is possible to narrow the search to the selected region for example, to the selected province. The value 0 means the whole world. Note that, province was just an example. How exactly regions will be organized is still being evaluated as requirements for addressing should be defined by routing possibilities, not the other way;
- B – City Identifier – identifier assigned to a single city or group of cities (32 bits). One city may have several such identifiers;
- C – ISP Identifier (*Internet Service Provider*) – identifier assigned to the network service provider in the selected urban area (area covered by the specified city identifier) (32 bits). One provider may have multiple such identifiers;

- D – User Identifier – user identifier involved with the ISP identifier (32 bits). This identifier allows to identify in the network the particular user, to which the specified network address belongs;
- E – LAN identifier – the last 16 bits, are owned by the user specified by the User Identifier. It may for example be used to create the user's own network LAN.

Exceptions to the above rules are merely addresses the values 0 and 1. Address 0 denotes the default or current hypernode depending on whether, according to the content of the subpackage header it is the first or following hypernode (if first, then 0 denotes the current hypernode). Address 1 denotes the current computer.

Described above scheme of addressing allows access to the CPI module (network address is similar to the address in TCP/IP, assigned to the physical network interfaces). However, in the EDEN network, due to the network structure (the hypernode existence), such a solution may not allow access to a particular service. To connect to the desired service need more information, such as with which hypernode and with which node is to be data exchange occurs. The unambiguous method to identify the service by address (not just CPI) must be defined. For this reason, the following was defined the addressing scheme, which allows for the unambiguous identification of which processes on which devices are replace data with each other:

```
<net_addr>-<hnode_id>/<N> or <net_addr>-<hnode_name>/<N>
```

where <net_addr> denotes a five part CPI address described above and <hnode_id> denotes the hypernode identifier (the number greater or equel to 0, 0 is the default hypernode), hnode_name denotes nonempty hypernode name and <N> denotes the name of node or possibly hypernode which is connected to the <hnode_name>. It should be remembered that <net_addr>-0 is the same as <net_addr>.

Address form <net_addr>-<hnode_id> allows access to hypernode. Each hypernode connected to the present has its own unique identifier known to all hypernodes directly connected with the current one. This is the same identifier that is used in the subpackage header. This means that:

- the address 1-0 is equal to 1 (0 – default hypernode, 1 – current computer),
- the address 0-0 is equal to 0,
- the address 1-0/0 denotes the hypernode with id=0 connected to the default hypernode on the current computer,
- the address 0/0 denotes the hypernode with id=0 connected to the default hypernode,
- addresses like 0/1/2/34/5 or 1/2/5/8/3 are correct. In the first case 0 denotes current hypernode and in the second one 1 does not denotes the current computer itself but hypernode with 0 connected with the default hypernode of current computer. This address is the same as 0/1/2/5/8/3.

Take the example of address: A.B.C.D.E-0/80/resource/other_resource could be interpreted as:

- A.B.C.D.E-0 – default hypernode under the A.B.C.D.E CPI address,
- 80 – node within the default hypernode, or identifier of the next hypernode,
- /resource – node containing access to the resource /other_resource or hypernode including node named /other_resource or hypernode with connected to it next hypernode /other_resource,
- /other_resource – hypernode or node or resource (file or service).

In description above resource can be a node existing within current hypernode or another hypernode connected to current hypernode. Question is since by looking at the address, using example address above there is no way of knowing what /resource and /other_resource really are. Every node and every hypernode connected to current hypernode is supposed to have name within current hypernode. It can be used several times within whole network but within single hypernode any name can describe only one node or hypernode. It means that if there already exists node called "object 1" as long as this node keeps it's name (hypernode may be allowed by administrator to change it) no other hypernode or node can be referred to as "object 1". So, when hypernode receives address containing /resource it can easily determine if address is valid what does it refer to. If /resource is hypernode then packet is passed to proper hypernode known as /resource. Now new hypernode performs same checkup for /other_resource. However, if /resource is a node then /other_resource is not being interpreted and is passed directly to application represented by /resource. Application should know what to do with it.

Construction of the network address implies the hypernode search algorithm. It is one of the most important algorithms in the specification of the EDEN network. In this network data is not transmitted between computers, but between hypernopdes regardless of whether both hypernodes located on a single device or not. In order to achieve data transfer between different hypernodes need to find a destination hypernode and connect to it. The hypernode search algorithm is as follows:

1) Node sends an empty package with host_search bit set to hypernode (ignoring any existing connections to the current node),
2) Hypernode broadcasts the package to all connected hypernodes including the correct connections between hypernodes,
3) The next hypernode receives the package and checks to see if the package has been received recently (checking the source and destination address), and whether the destination address is the hypernode address.

4) If the destination address and the address of the current hypernode are the same then skip the next point (5),
5) If the package has recently received then stop the algorithm,
6) Add the current address to the package contents,
7) If the destination address and the current one are identical then send back the package with set host_search and host_search_completed bits,
8) If the search is not completed then send the package to all connected hypernodes taking into account only the correct connection and each hypernode repeat steps 3-8.

This is the simplest version of the hypernode search algorithm. In addition, it can be optimized through the use of maps of the area of each hypernode (future works).

5. Data Transfer at the Hypernode Level

One of requirements for EDEN project is to create network solution capable of transferring data from one hypernode to another while using multiple traces at the same time. Basically what is done in EDEN is that network itself defines possible routes between multiple hosts (links between hypernodes are already directional). In it's simplest version connections between hypernodes in EDEN look the way it is commonly organized in TCP/IP based networks (i.e. multiple data links to local hosts + one data link to some wider area network). However, as EDEN allows to create parallel links or also allows easily to attach one host to multiple networks at the same time, this in addition to ability to have multiple network independent environments on single system, allows also easy way to manage access privileges of every single process to every single network resource.

For the purpose of transfer organization there are a few assumptions. Master hypernode sets individually for each slave hypernode connection speed. Connection speed may be changed at any time. Slave hypernode always must know current connection speed limit. Master hypernode may use any means in order to determine both best connection speed and connection usage as long as those means are according to EDEN specification.

Master hypernode always has to inform slave hypernodes of any changes to connection speed by sending single package (with connection_limit bit set) to each affected hypernode. Such package contains only one number higher than 0. This number is a modifier used to determine connection speed. Slave hypernode interprets this package in following way: assuming that X is defined in package as connection speed modifier then connection speed itself is set to $X \times 32$ kilobits per second. Additionally, the slave hypernode needs to inform master hypernode how much data it can process during single cycle. It is done one time during connecting hypernodes.

On Fig. 3 there is a simple example. A can send and receive data from B with speed of 16×32 Kbps and from C 32×32Kbps. B can transfer data from D with speed of 64×32Kbps and C can receive data from D with speed of 16×32Kbps. Assuming transfer from D to A and having additional connection between B and C (48×32 Kbps) it is possible to conclude:

- Data from D to C can be transferred at the speed of 16×32 + 48×32 = 64×32Kbps,
- Data from D to A by using C can be transferred at the speed of 32×32Kbps,
- Data from D to A by using B can be transferred at the speed of 16×32Kbps,
- Connection between A and C can be utilized in 100%,
- Connection between B and D is utilized in 75%,
- Data from D to A by using both B and C can be transferred at the speed of 32×32 + 16×32 = 48×32Kbps,

The description above just briefly shows how EDEN is supposed to handle data transfer on hypernode level. Obviously hypernode needs to get enough information from it's neighbors however it cannot go too far as this would increase number of in fact useless packets being sent over the network (such data would become easily outdated). Currently it is assumed that this information can be shared only between directly connected hypernodes with possibility of involving two directly connected CPI being evaluated to make sure that both hypernodes are on different devices. Using that information in example above A should know state of B and C but should have no idea about state of D.

Also EDEN provides here only tools to be used. It does not define exact algorithm how to divide data in most optimal way. This algorithm for the time being is not part of strict EDEN specification. EDEN should support here different solutions in order to provide experience that meets best requirements of both users and software working with EDEN. 32 Kbps granularity is being used in current specification is used for the purpose of testing only. The exact value should be reevaluated as it should allow supporting all kinds of hardware and software technologies involved.

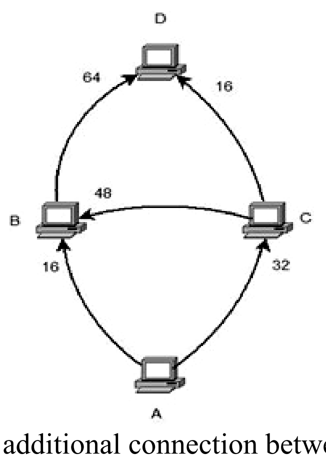

Fig. 3. Network with additional connection between B and C
Rys. 3. Sieć z dodatkowymi połączeniami pomiędzy węzłami B i C

6. Data Transfer at the Node Level

EDEN offers some solutions to improve efficiency on low bandwidth links.

Supernodes – this is the way to cache frequently downloaded files transparently to both service providing this file and process requesting data. Full control over supernode belongs to hypernode to which supernode belongs. Hypernode is completely responsible for managing it's supernodes. Basic procedure for supernodes are described below:

Data caching

1) Data are received by supernode,
2) Supernode checks if retrieved data are not the part of any special package (all special bits in both package's and subpackage's headers must be set to 0),
3) Supernode checks if received data isn't already cached,
4) Supernode checks if data can be cached (for instance if there is enough space available) and if all requirements are met. If any of those requirements can't be determined then supernode may request additional data required to check those requirements again,
5) Supernode caches received data,
6) Supernode requests remaining parts of file (including checksum for the whole file)
7) Source node starts sending requested data
8) Supernode caches rest of data and uses received checksum to determine if transfer was successful,
9) If any error occured then supernode may request go back to step 6 and request data again.

Supernodes may cache only files. Also checksum for file must be cached. This checksum may be received only from original source of the file. The method above is performed simultaneously with sending data to whoever requested it initially. It is possible to cache only parts of file however, in this case in step 6 supernode should request checksum only for parts to be received.

The above procedure in case of any file as long as it is possible to recreate any part that is supposed to be cached (supernode either received it initially or requested it at a later time) of the file by using supernode's cache. In every case whole file name has to be saved (including full network address for original file).

Sending data

1) Supernode receives request for data and sends it to source node,
2) Supernode checks what parts of the file are cached. If there are any parts cached then supernode sends it to node that requested data,
3) Supernode receives list of parts to be sent,
4) Supernode starts sending data

Mirrors – on the Fig. 4 was presented how mirrors work. As this feature heavily depends of supernode specification without going into details mirror is a supernode that is specifically recognized by source node (the one that contains original files) as external source for data requested by client process. The simple mechanism for mirrors is as follows:

1) The node A requests a file,
2) Source node starts sending file and also sends list of supernodes that may have copy of this file,
3) The node A requests chosen parts of the file from mirror supernodes,
4) Every mirror supernode sends requested parts or data that transfer is not possible with explanation (i.e. file is not available).

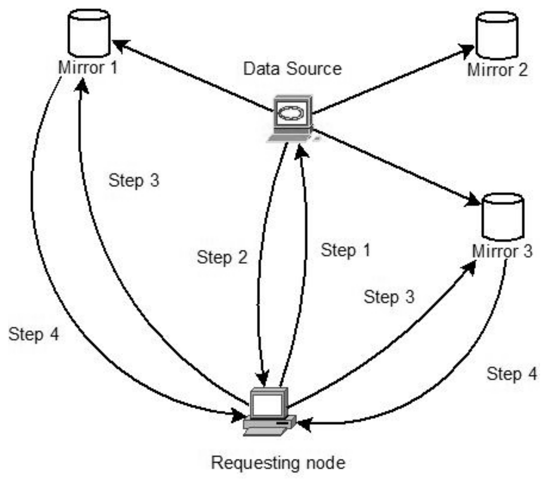

Fig. 4. The schema of data extraction of mirrors
Rys. 4. Schemat pobierania danych z serwerów lustrzanych

Note that supernode can't support caching all kind of data. Network streams can't be cached. Any data that may be modified at a later time should not be cached. Files that need authorisation should not be cached too. There are also a few other conditions that need to be met so the file may be cached. Decision about what can and what cannot be cached is made by application serving file.

When supernode requests data for the purpose of caching it, node may refuse it. Supernode in that case must end caching procedure immediately and drop all already cached parts of requested file. Also supernode can't serve any single part of the file from it's cache before it receives confirmation from source node. By confirmation it is understood that source node does not refuse request and caching process goes on the as it is described above. Supernode by default should support only HTTP protocol. However if supernode represent some process it is assumed that this process can handle more protocols according it's design. In that case supernode's implementation should provide proper API allowing process to decide what to cache and what is supposed to be dropped. Note that execution of this procedure is performed as independent to sending or receiving any data therefore it should not affect user's experience. Possibly before supernode attempts to cache rest fo the file, user already got all parts of requested resource.

7. Tests of Hypernode Searching

The primary function of the test was to check the possibility of setting up the data transfer between any two randomly selected hypernodes, i.e. was to verify, whether the destination hypernode is able to retrieve the data from the source hypernode considering all connections settings according to EDEN network specification. To accomplish this, target hypernode must check whether the source hypernode is visible i.e. if there is at least one route between the two hypernodes.

Tests were carried out for the network containing only hypernodes and slots. Each of the three performed tests consisted of: generating 10 different networks, and conduct random searches of 100 randomly selected network addresses starting with the randomly selected hypernode. The tests were performed under the Windows 2000 operation system.

The results of each test were saved in the file: each line contains separate result and starts from hypernodeX (the hypernode name) where X is the number of the generated hypernode. If there is possibility of connection after the hypernode name is printed a list of addresses that should be used to establish communication in order: from the left there is the address for the nearest hypernode and most on the right is the destination hypernode address.

Each network address used to designate the path between hypernodes is assigned to exactly one hypernode. Because the resulting file contains 100 lines for each generated network below it was shown only the a few results:

```
hypernode0;;000000000001;000000000034
hypernode2;;000000000001;000000000028;000000000037
hypernode2;;000000000012;000000000028;000000000009
hypernode1;;00000000005C;000000000039;000000000054
```

In the test application speed optimization has not been applied. The purpose of the test application was not checking the speed of the hypernode search operation but the checking the hypernode search algorithm. Each performed test shows that each first hypernode search was successful. The average number of strokes taken during the search of the hypernode is 4 including the source and destination hypernode. It should be noted that all of the generated network are completely random what has a significant impact on the received results.

8. Conclusions

Described in this paper EDEN computer network is able to replace most of the modern network based on IP protocols. Some of the solutions proposed in the work is based on existing solutions in various network standards. But EDEN is the first network linking these different solutions in a single, cohesive solution for the entire network. Examples are supernodes that have been inspired by proxy servers, with the major difference is that supernode works in a fully automatic and independent from user way. It does not require any additional configuration, so it works completely transparently to users.

EDEN network limits the number of protocols that are used in the TCP/IP networks at the network and transport layer. An example is the construction of the subpackage header, through which it is possible to maintain the protocol analogous to TCP or UDP. As for the data transfer efficiency by the new realization of the connections between hosts the only real limitation of the transfer may only be lower layer technologies. However EDEN at it's current state still needs a lot of work to be done and is definitely not ready for large scale networks. EDEN even if now may work with small networks of several computers, it still needs a lot of work in order to make it ready for larger networks of thousands computers connected. In order to resolve those issues and extend the capabilities of the EDEN network are carried out work on the following issues:

BIBLIOGRAPHY

1. Seedorf J.: Security Issues for P2P-Based Voice- and Video-Streaming Applications, iNetSec 2009 - Open Research Problems in Network Security, Springer-Verlag, vol. 309, 2009, pp. 95-110.
2. Lass R. N., Nguyen D. N., Millar D. W., Regli W. C., Macker J., Adamson R. B.: An Evaluation of Serverless Group Chat, Military Communications Conference, 2011, MILCOM 2011, IEEE Conference Publications, pp. 1639-1644.
3. Jin Li: On peer-to-peer (P2P) content delivery, Peer-to-Peer Networking and Applications, Springer Science+Business Media, LLC 2008, Volume 1, Issue 1, pp. 45-63.
4. Yeoh C. T., Kendall E. A., Khan A. I.: An agent based approach to discovery and formation of the Serverless Core Network, Proceedings of the 2005 13th IEEE International Conference on Networks, IEEE Computer Society, Piscataway USA, pp. 692-697.
5. Buyya R., Cortes T., Jin H.: Serverless Network File Systems, High Performance Mass Storage and Parallel I/O,Technologies and Applications, Wiley-IEEE Press pp. 364 -385.

6. Jiwhan L., Sangjin K., Heekuck O., Donghyun K.: A Designated Query Protocol for Serverless Mobile RFID Systems with Reader and Tag Privacy, Tsinghua Science and Technology, ISSN 1007-0214 04/10, vol. 17, no. 5, 2012, pp. 521-536.
7. Tan C. C., Sheng B., Li Q.: Secure and Serverless RFID Authentication and Search Protocols, IEEE Transactions on Wireless Communications, 2008, 7(3): 1400-1407.

Received on March 24th, 2013.

Omówienie

W artykule została przedstawiona specyfikacja struktury i organizacji transferu danych dla nowej sieci komputerowej EDEN. Model takiej sieci może z powodzeniem zastąpić istniejącą sieć o organizacji opartej na protokołach TCP/IP. Omówiono również podobieństwa i różnice do modeli OSI i TCP/IP. Zdefiniowano precyzyjnie elementy sieci, takie jak: węzeł, hiperwęzeł, slot, moduł CPI (ang. *Collision Prevention Interface*) i szczegółowy algorytm działania tego modułu. W dalszej części zdefiniowano budowę nagłówków pakietu i podpakietu (zdefiniowane pola nagłówka) oraz schemat adresacji w sieci, a także algorytm wyszukiwania hiperwęzłów, od którego jest uzależniony efektywny transfer danych. Transfer danych przedstawiono na poziomie hieprwęzła oraz na poziomie węzła. Na koniec zaprezentowano testy aplikacji testowej, generującej losowe sieci EDEN i testującej algorytm wyszukiwania hiperwęzłów. Pomimo pozytywnych testów, losowość generowanych sieci ma istotne znaczenie – w praktyce sieć może być odpowiednio skonfigurowana przez administratora dla otrzymania dużej efektywności algorytmu wyszukiwania hiperwęzła.

Adresses

Marcin DĘBICKI: Tyszkiewicz High School, BWS, ul. Nadbrzeżna 12, 43-300 Bielsko-Biała, Poland, marcin.debicki@me.com
Zdzisław ONDERKA: Akademia Górniczo-Hutnicza University of Science and Technology, Department of Geoinformatics and Applied Computer Science, Al. Mickiewicza 30, 30-059 Kraków, Poland, zonderka@agh.edu.pl