A Strategic Search Method for Requirement-Aware Functional-Block Selection and Composition in a Flexible Network

Daniel Günther, Nathan Kerr, Paul Müller
Integrated Communication Systems (ICSY)
University of Kaiserslautern
Germany
{guenther, kerr, pmueller}@informatik.uni-kl.de

Journal of Digital Information Management

ABSTRACT: *Internet applications have reached such a high complexity that the network stack concept introduced decades ago is not suitable anymore. Future Internet research activities try to increase the flexibility of the Internet in order to allow the network to adapt to present and future requirements and applications. A well known approach is to build protocol graphs, which will be used instead of inflexible network stacks, by connecting functional blocks together. A functional block encapsulates a particular network function. The protocol graph that should be used is the one most suitable to the application's requirements. To find the most suitable graph, all possible protocol graphs must be evaluated. However, the number of possible protocol graphs increases exponentially as the number of functional blocks increases. This article presents a method of modeling the protocol graph search space as a set of search trees and then using search strategies to manage the inherent complexity of the search process. We evaluate our proposed method by simulation.*

Categories and Subject Descriptors
E.1 [**Data Structures**]; Graphs and network: **I.2.8** [**Problem Solving, Control Methods, and Search**]; Graph and tree search strategies: **C.2.6** [**Internetworking**]; Standards

General Terms: Internet Protocols: Protocol Graphs, Next Generation Networks

Keywords: Future Internet, Flexible Networks, Selection, Composition, Complexity, Protocol Graph

Received: 12 January 2012, Revised 8 April 2012, Accepted 12 April 2012

1. Introduction

In the current layered stack model of the Internet, protocols dictate the availability and composition of network functionality. A flexible architecture, as often discussed in the Future Internet research area, will be able to support the desire to dynamically choose certain mechanisms based on the requirements of a particular application. In the 90s many approaches dealt with flexible configurations or functional modules. This idea was the vision for projects like Adaptive [6], DaCaPo [11], and FCSS [9]. More recent projects with similar goals are 4WARD [12 ,14], NENA [13] and SONATE [5]. This research has grown into a big scientific area. The research seeks to provide domain specific network compositions, more flexibility through functional composition and application aware compositions. In general, the basis for providing this functionality consists of functional blocks, which represent network functionality and are combined into protocol graphs.

These approaches generally work in the way shown in Figure 1. An application provides its requirements which are combined with network offers and any addition criteria from the system. A selection and composition system draws from a pool of functional blocks to create a protocol graph which fulfills the application's requirements.

This paper assumes that functional blocks, as shown in Figure 2, take packets as input and produce packets as output. Functional blocks are connected together to form protocol graphs, which are directed a cyclical graphs consisting of nodes with two connections at most. One connection is for the entry port, the other for the exit port. Both the sender and receiver use the same protocol graph with the down and up (input/output) flows reversed.

2. Motivation

The number of possible protocol graphs depends on the

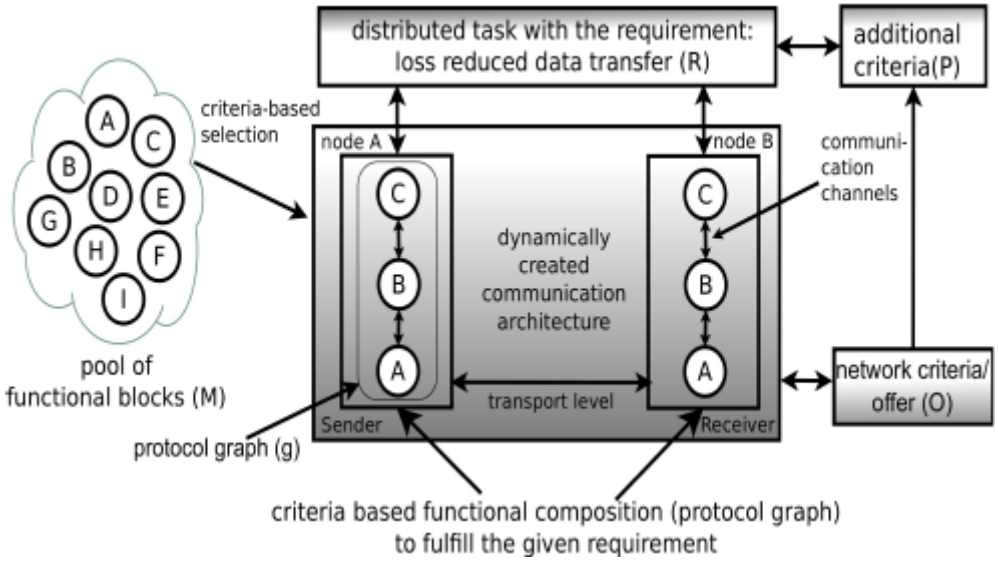

Figure 1. General Model for Functional-Block-Based Future-Internet Approaches

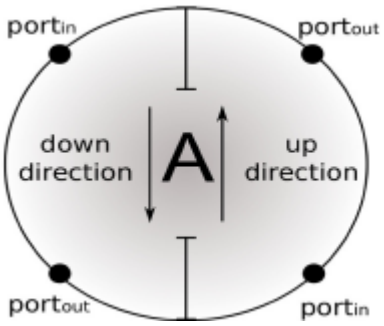

Figure 2. Functional Block

number of functional blocks in the pool. The growth of possible protocol graphs is exponential.

To find the most appropriate protocol graph from the set of possible protocol graphs, all possible protocol graphs need to be considered. Protocol graphs that do not fulfill the requirements and constraints can be dropped, while those that can be used need to be compared to determine which one is most suitable.

This paper presents a method of representing the protocol graph search space as a set of search trees and defines three strategies for selecting which protocol graphs will be extracted from the search space for later comparison.

Section 3 presents related work. Section 4 presents our method of representing protocol graphs as a set of search trees and presents the complexity-reduction strategies which we propose to use. Section 5 evaluates our method. Sections 6 and 7 present our conclusions and the larger context for this work.

3. Related Work

Several different approaches exist to reduce or explore the protocol graph search space [15-17]. *NENA* [13] provides a set of human composed protocol graphs. This reduces the runtime selection process to a smaller set of precomposed graphs.

Template-based approaches use experts to build outlines or skeletons of protocol graphs that have slots for functional blocks that fulfill a specific class of functionality. For example, a template may have a slot for encryption, but does not specify which encryption mechanism is used. While templates reduce the protocol graph search space, they have the same problem as fully designed network stacks in that they may not do just what an application needs. This problem can be mitigated by introducing a set of situational templates (e.g., one for file transfers, one for video conferencing). This is one step better than the fully manual process, but it is still limited to the available templates.

Genetic algorithm approaches [7] fully embrace time constraints. These approaches can return the best graph found in a set amount of time. There are no guarantees about the quality of the protocol graph, or even that an appropriate graph will be found, assuming that one exists.

Both of these approaches assume that the base problem of searching the entire set of possible protocol graphs is too time consuming. This paper looks at ways of efficiently searching the entire set of possible protocol graphs. Only by searching the whole space can the most appropriate solution be guaranteed to be found.

One way of efficiently searching an entire space is with search trees. This is the approach applied here.

4. The RACR-SP Approach

The Requirement Aware Complexity Reduction – Selection

Process (*RACR-SP*) consists of two parts. First, the protocol graph search space is represented as a set of search trees. Second, the tree is searched using a set of requirement aware strategies to reduce complexity.

The result of *RACR-SP* is a set of protocol graphs which fulfill the requirements. All graphs which do not fulfill the requirements are excluded from further evaluation. This reduces the number of protocol graphs that need to be directly compared to determine which one best fulfills the requirements.

4.1 Modeling the Search Space

To fully explore a protocol graph search space, we represent protocol graphs as a set of search trees. This representation assumes that each node in a protocol graph can only have one entry edge and one exit edge. This means that protocol graphs look like network stacks, but without defined layers like the ISO/OSI layer model. Each functional block receives packets as input and produces packets as output.

Furthermore, a functional block can only appear once, if at all, in a protocol graph. All possible protocol graphs are encoded into a set of trees. The root of each tree is a network offer. Each branch begins with one functional block and then spreads out with the other functional blocks. Figure 3 shows the tree generated for a pool of three functional blocks: *A*, *B*, and *C*. *N* is a network offer.

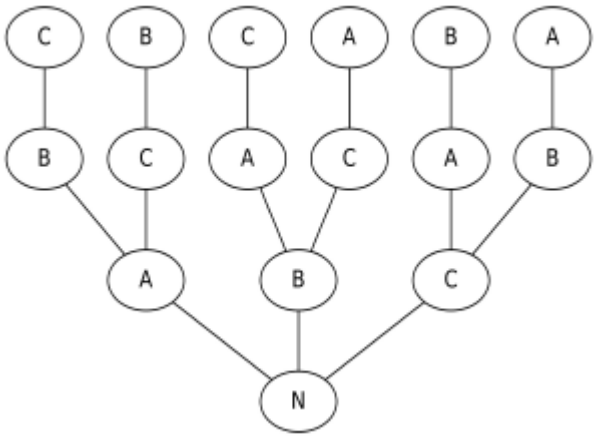

Figure 3. An Example Model of the Search Space

4.2 Search Process

Once the search space has been modeled, it needs to be searched. Algorithm 1 shows the search algorithm. This algorithm relies on an iterator which performs depth-first, iterative-deepening search [4] of the search tree, referred to in this algorithm as *graphs*. By using this iterator, the search algorithm becomes quite clear. First the set of usable (suitable) protocol graphs, *P*, is initialized as an empty set. The iterator will search the tree to the specified depth, *k*. The iterator is then used to loop through all the protocol graphs defined in the search space. For each protocol graph, the Impact Evaluation Model (IEM) [3] is used to determine if the protocol graph fulfills the goals, or not. If the graph is suitable, it is added to the set of usable graphs (*P*). After evaluating every possible protocol graph, the algorithm returns the set of usable protocol graphs.

Algorithm 1: Search Process

```
P is an empty set
graphs.depth = k
while graphs.has_next() do
  g = graphs.next()
  is_suitable = suitable(IEM.impact(g), goal)
  if is_suitable then
    P.add(g)
  end if
end while
return
```

4.3 Complexity Reduction

The search process shown in Algorithm 1 will evaluate every possible protocol graph. The number of possible protocol graphs increases exponentially with the number of functional blocks. The number of functional blocks that fulfills even a single functionality such as encryption can grow quite large. Therefore, a means to reduce the number of evaluated protocol graphs, and therefore the complexity of the search process, is required.

To reduce the occurring complexity we introduce the concept of search strategies. To handle the complexity in different ways we define three strategies.

4.3.1 Concept: Search Strategies

A search strategy is made up of two elements which indicate which action the search process should take. The two elements are:

should_stop indicates that the search process should stop; no more protocol graphs should be evaluated

should_prune indicates that the iterator should prune the elements further down the current search branch after the current protocol graph.

4.3.2 Strategy: All Graphs Found (AGF)

All Graphs Found (AGF) searches the entire space. This presents the worst case for all three strategies presented in this article. The other methods will stop searching or prune branches when a suitable graph is found. This strategy does not. As a result, it will find all the protocol graphs which fulfill the application requirements.

If there is a suitable graph, at least one graph will be returned. As more than one graph may be returned, the most suitable of these graphs must be selected. This selection process is covered in [1].

4.3.3 Strategy: First Suitable Graph (FSG)

First Suitable Graph (FSG) returns the first suitable protocol graph that is found. This is the fastest method,

and requires no further work to select the best available graph, as it only returns one graph. There is no guarantee that the best graph will be found, only that a suitable graph – if one exists – will be found.

The worst case for this strategy is that all possible protocol graphs will be searched and evaluated. This is no worse than AGF.

4.3.4 Strategy: Prune If Fits (PIF)

Prune If Fits (PIF) works by not continuing to follow a branch when a suitable graph has already been found on that branch. The unsearched portion of the branch is pruned from the search space. This method is adapted from the idea of forward pruning[8].

Doing this should produce fewer suitable graphs than AGF. By pruning branches, the total number of protocol graphs considered is reduced at the cost of not guaranteeing that the most suitable graph is found (if one exists). This tradeoff should reduce the time required to select a graph while not overly reducing the possibility that the most suitable graph is not found.

While finding the most suitable graph is not guaranteed, the graphs not considered use more functional blocks than the ones considered. The use of more functional blocks will generally consume more resources and increase the impact of the protocol graph (e.g., increased computation, data rate, delay).

4.4 Strategic Search Process

The strategic search process (Algorithm 2) is similar to the search process described earlier. The first difference is that the graph iterator also needs to support forward pruning [8]. The main loop has two additions. After finding out if the protocol graph which is being evaluated is suitable or not, the indicator for the *should_stop* element of the selected strategy is checked to see if the search process should continue, or not. If the search process continues, the *should_prune* strategy element is checked and the branch is forward pruned if indicated.

Algorithmus 2: Strategic Search Process

```
P is an empty set
graphs.depth = k
while graphs.has_next() do
  g = graphs.next()
  is_suitable = suitable(IEM.impact(g), goal)
  if is_suitable then
     P.add(g)
  end if
  if should_stop(is_suitable, strategy) then
    return P
  end if
  if should_prune(is_suitable, strategy) then
  graphs.prune()
  end if
end while
return P
```

The combined indicators for the *should_stop* element for AGF, FSG, and PIF are checked in algorithm 3. The only strategy that indicates that the search process should stop is FSG, which stops the process when a suitable graph is found.

Algorithmus 3: should_stop(suitable, strategy)

```
if suitable && strategy == FSG then
  return true
end if
return false
```

The combined indicators for the *should_prune* element are shown in algorithm 4. Only PIF uses this indication when a suitable graph is found to forward prune the search tree. This reduces the number of graphs that will be evaluated.

Algorithmus 4: should_prune(suitable, strategy)

```
if suitable && strategy == PIF then
  return true
end if
return false
```

4.5 Example Search Process

Figure 4 shows how a strategic search process could run. This example uses three functional blocks: *A*, *B*, and *C*. *N* represents the network offer. The maximum depth searched is three.

At first the algorithm checks to see if the network offer (*N*) can fulfill the application requirements. In this example it does not. Now all the protocol graphs with a length (*k*) of one are checked: *NA*, *NB*, and *NC*. This example assumes that some combination of the provided functional blocks will fulfill the requirements.

NB and *NC* are found to fulfill the application requirements. The *PIF* strategy method would then prune the branches extending from *NB* and *NC*. *FSG* would stop evaluating graphs at *NB* (*NC* would not be evaluated), and return the protocol graph *NB*.

After *k* is incremented by one, the *AGF* and *PIF* methods will continue by evaluating *NAB* and *NAC*, which are found to not fulfill the requirements. The next graphs to be evaluated would be *NBA* and *NBC*. However, the PIF strategy dropped those branches because *NB* and *NC* already fulfill the application requirements. *AGF* will evaluate these nodes now and then continue with *NCA* and *NCB*, which also would have been dropped by *PIF*.

After *k* is incremented by one, *NABC* and *NACB* are evaluated by both *AGF* and *PIF*. Neither of these graphs fulfill the application requirements.

If the *PIF* strategy is used, evaluation stops here and the

graphs *NB* and *NC* are returned.

AGF would continue evaluation with *NBCA*, *NBAC*, *NCAB*, and *NCBA*; none of which fulfill the requirements. As there are no more graphs to evaluate, *AGF* returns the graphs *NB* and *NC*.

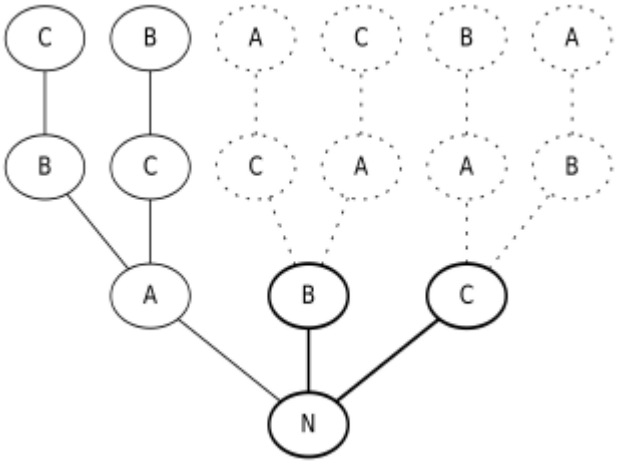

Figure 4. Example Search Process

As shown in this example, *RACR-SP* can reduce the number of protocol graphs evaluated. In the next section, we use simulation to evaluate our approach.

5. Evaluation

To evaluate our approach we first implemented the defined strategies (*AFG*, *FSG*, and *PIF*) by using the *MPM* library [2] and the *OMNeT++* [10] simulation environment. This implementation allows us to execute each strategic method given a set of application requirements, a network offer, and a set of functional blocks. The specific details for each of these items are not specified here, as we are interested in how the strategies perform, not how well the returned protocol graphs fulfill the requirements.

Our simulation specified application requirements, a network offer, and a set of four (4) functional blocks with their associated impact functions. Impact functions are used to calculate the effect of a functional block on the network offer. We used the simulation to measure how many protocol graphs there could be with the given maximum protocol graph depth (k_{max}), how many protocol graphs were actually evaluated, and finally how many protocol graphs were usable (i.e., fulfilled the application requirements). Figure 5 shows all the data we gained from our simulated scenarios.

The purpose of the strategies *RACR-SP* uses is to reduce complexity, which in this case directly relates to the number of protocol graphs evaluated. Therefore, to see the effect of the strategies on search complexity, we will look at them from three perspectives.

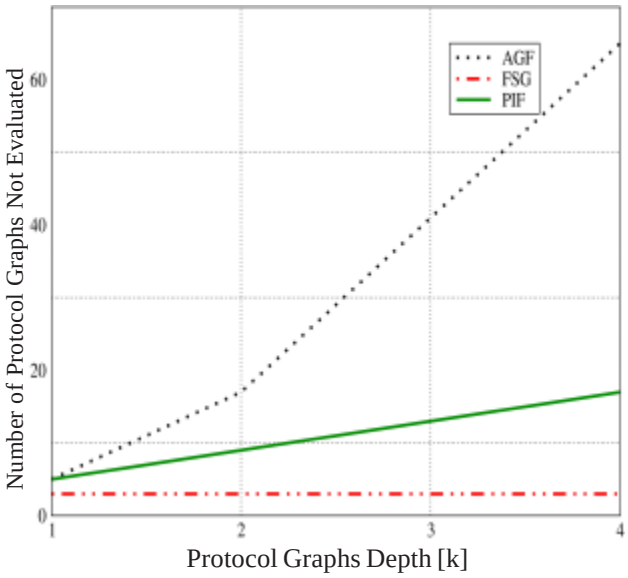

Figure 6. Number of Evaluated Protocol Graphs

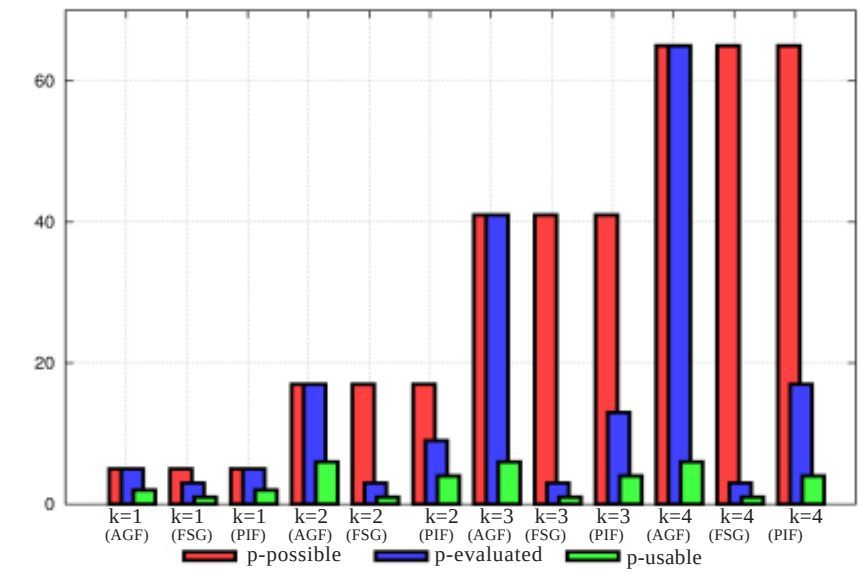

Figure 5. The Influence of Used Strategies

The first perspective, shown in Figure 6, shows how many protocol graphs are evaluated by each strategy. The best case from this perspective is to evaluate one protocol graph, where that protocol graph happens to be the most suitable protocol graph to fulfill the given requirements. The worst case is to evaluate every protocol graph.

AGF always evaluates every protocol graph; this line is the worst case for all three strategies. *FSG* stops evaluation as soon as it finds a graph which fulfills the application requirements. *PIF* prunes branches when a requirements fulfilling graph is found. Except in the worst case, *PIF* evaluates fewer graphs than *AGF*, but more than *FSG*. We see that *PIF* reduces the number of protocol graphs evaluated, but does so in a manner that we think it likely that the most suitable protocol graph will still be evaluated based on the explanation previously given for *PIF*.

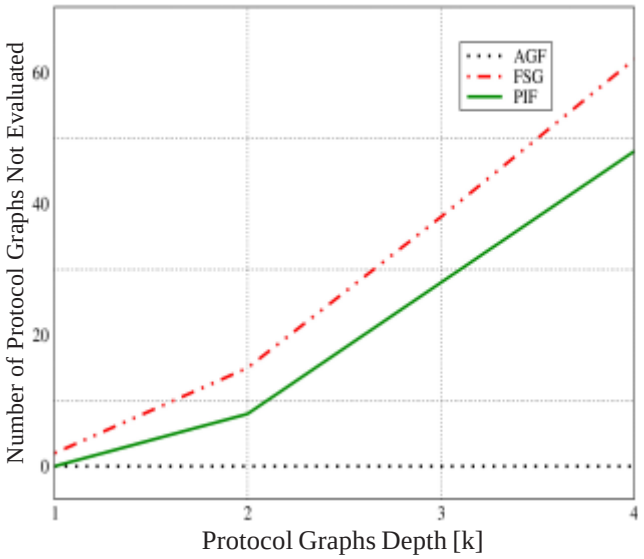

Figure 7. Number of Protocol Graphs not Evaluated

The second perspective, shown in Figure 7 shows the number of protocol graphs which are not evaluated. The theoretic best case would only evaluate the most suitable protocol graph, which means the best number of protocol graphs to not evaluate is the number of possible protocol graphs minus one. The worst case is that no protocol graphs are not evaluated (meaning that every protocol graph is evaluated).

AGF evaluates every possible protocol graph, and so the number of protocol graphs not evaluated is zero. This is the worst case possible. *FSG* evaluates protocol graphs until it finds one which is suitable. This is practically the best case, except that there is no guarantee (or reason to expect) that the suitable graph which is found is the most suitable.

PIF works in between these two extremes by increasing the number of protocol graphs not evaluated as compared to *AGF*, while not hitting the practical limit of *FSG*. Therefore, *PIF* looks to be a good compromise.

The third perspective, shown in Figure 8, shows the number of usable (suitable) protocol graphs found by each strategy. The best case would be to find a single protocol graph, where that protocol graph is the most suitable. The worst case is to find all the usable protocol graphs.

AGF, as usual, follows the worst case line, finding every usable protocol graph. *FSG* finds only a single usable protocol graph, but again there is no guarantee or reason to expect that that protocol graph is the most suitable one. Once again, PIF treads the middle ground, returning a subset of all possible usable protocol graphs, increasing the possibility that the most suitable graph is found while not reaching the worst case.

This empirical evaluation only shows the type of results that could be expected in a situation similar to that which was simulated. A theoretical evaluation would be needed to show anything more. As we needed an implementation to continue our direction of research, we think this evaluation is sufficient.

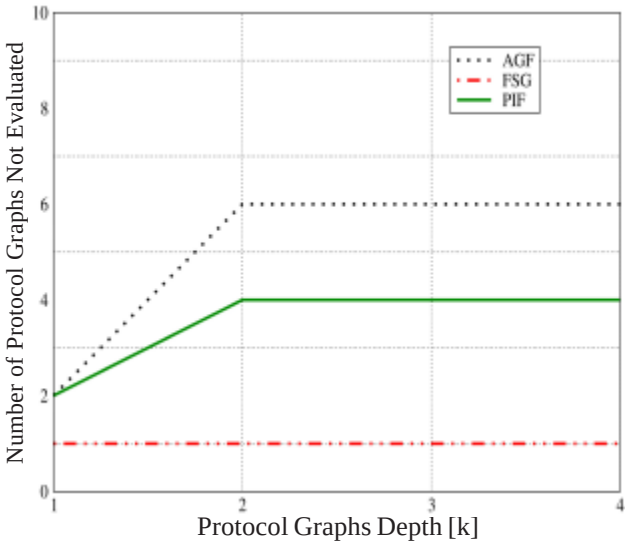

Figure 8. Number of Usable Protocol Graphs

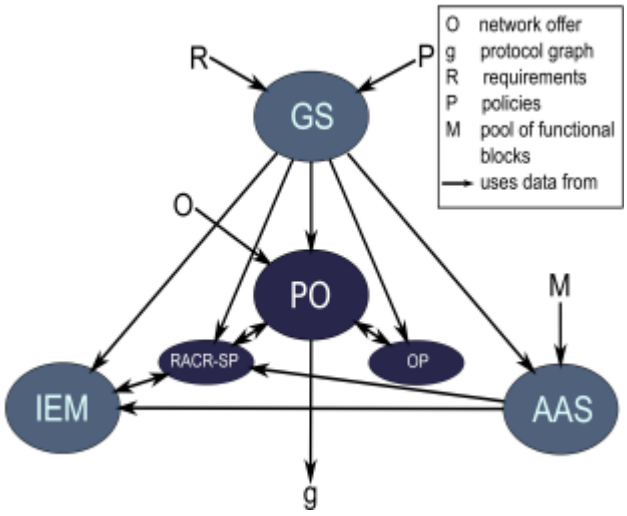

Figure 9. MPM

6. Conclusion

This paper presents *RACR-SP*, a method of representing

the protocol graph search space as a set of search trees and the requirement aware strategies used to control the search of those trees.

Three strategies are presented. All Graphs Found (*AGF*) evaluates all possible graphs. This method will find every protocol graph that fulfills the application's requirements. First Suitable Graph (*FSG*) will evaluate as many graphs as needed to find a suitable graph. This may be as few as one, or as many as every possible graph.

Prune If Fits (*PIF*) searches down every branch of the search tree; pruning the unsearched part of the branch once a suitable graph on that branch has been found. Except in the worst case, this process evaluates fewer graphs than *AGF*. While PIF will not find every suitable graph, the excluded graphs are deeper (have a greater k) than the already suitable graph on that branch. We assume that a deeper graph has greater impact and will therefore be less suitable. This simple method reduces the number of graphs that will have to be compared later in a way that will probably not remove the best graph from consideration.

Our empirical evaluation indicates that the strategies function in the manner we envisioned.

7. Context of This Work

This work describes one of the components in the Multi-Step Process Model [2], which is a systematic approach to solve the functional selection and composition problem in functional-block-based future internet approaches. The overall structure of MPM is shown in Figure 9. Other developed components of MPM include the Impact Evaluation Model (*IEM*) [3] and the Optimization Process (*OP*) [1].

8. Acknowledgment

This work has been created as part of the G-Lab project, funded by the German Federal Ministry of Education and Research (*BMBF*).

References

[1] Günther, D., Kerr, N., Müller, P. (2012). Auswahl von netwerktransporteangeboten in einer future-internet-architektur basierend auf funktionalen blöcken. *In:* 5. DFN-Forum Kommunikationstechnologien, Regensburg, Germany.

[2] Günther, D., Kerr, N., Müller, P. (2012). A multistep process model for selecting and composing functional blocks in a future internet architecture. *In:* Proceedings of The Sixth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS-2012), Palermo, Italy.

[3] Günther, D., Kerr, N., and Müller, P. (2012). A simulation model for evaluating the impact of communication services in a functional-block-based network protocol graph. 15th Communications and Networking Simposium (CNS'12) at SpringSim'12, Orlando, USA.

[4] Richard, E. Korf. (1985). Depth-first iterative-deepening: An optimal admissible tree search. *In:* Artificial Intelligence, p. 97–109. Elsevier Science Publishers B.V.

[5] Müller, P., Reuther, B. (2008). Future internet architecture - a service oriented approach. it - *Information Technology*, 50 (6) 383–389.

[6] Schmidt, D. C., Box, D. F., Suda, T. (1993). Adaptive: A dynamically assembled protocol transformation, integration and evaluation environment. Concurrency - Practice and Experience.

[7] Dennis Schwerdel, Bernd Reuther, Paul Müller. (2010). On using evolutionary algorithms for solving the functional composition problem. *In:* 10th Würzburg Workshop on IP: Joint ITG, ITC, and Euro-NF Workshop, Visions of Future Generation Networks.

[8] Smith, J. J., Stephan, Dana Nau, S. (1994). An analysis of forward pruning. *In:* Proceedings of the AAAI-94.

[9] Stiller, B., Fukss, (1994). Ein funktionsbasiertes kommunikationssubsystem zur flexiblen konfiguration von kommunikationsprotokollen. GI/ITG-Fachgruppe Kommunikation und Verteilte Systeme.

[10] András Varga, Rudolf Hornig, (2008). An overview of the omnet++ simulation environment. *In:* SimuTools, p. 60.

[11] Vogt, M., Plagemann, T., Plattner, B., Walter, T. (1993). A run-time environment for da capo. *In:* Proceedings of INET93 International Networking Conference of the Internet Society.

[12] Völker, L., Martin, D., Khayat, I. E., Werle, C., Zitterbart, M. (2008). An architecture for concurrent future networks. *In:* 2nd GI/ITG KuVS Workshop on the Future Internet.

[13] Völker, L., Martin, D., Werle, C., Zitterbart, M., Khayat, I. E. (2009). A node architecture for 1000 future networks. *In:* Proceedings of the International Workshop on the Network of the Future 2009. IEEE.

[14] Völker, L., Martin, D., Werle, C., Zitterbart, M., Khayat, I. E. (2009). Selecting concurrent network architectures at runtime. *In:* Proceedings of the IEEE International Conference on Communications (ICC).

[15] Zahary, Ammar.,Ayesh, Aladdin (2010).A Comparative Study for Reactive and Proactive Routing Protocols in Mobile Ad hoc Networks, *Journal of Intelligent Computing*, 1 (1) 20-29.

[16] Muhammad, S. Aslam, Susan Rea, Dirk Pesch, (2011).An innovative Hybrid Architecture and design for Wireless Sensor Networks, *Journal of Networking Technology*, 2 (1) 29-35.

[17] Tran, Tich Phuoc., Nguyen, Thi Thanh Sang., Dang, Kien Cuong., Kong, Xiaoying (2011).An Efficient Web-Page Recommender System using Frequent Pattern Discovery and Dynamic Markov Models, *International Journal of Web Applications*, 3 (1) 1-11.

Daniel Günther After extended vocational training in energy electronics engineering in 1997, he received his intermediate diploma degree in 2003 and his diploma degree in 2006, both in computer science, from the Faculty of Mathematics, Natural Sciences and Computer Science at the University of Cottbus.

His current research activities are mainly the functional and management aspects of flexible networks and future internet research topics, including the areas of simulation and electronics. Since 2009, Daniel Günther is currently working towards a Ph.D. degree in computer science at the University of Kaiserslautern as a member of the scientific staff at the Integrated Communication Systems Lab.

Nathan Kerr After receiving his B.A. in Journalism and Mass Communication from the Walter Cronkite School of Journalism at Arizona State University, Nathan returned to studying Computer Science by earning an M.S. in Computer Science at ASU while working as an administrator for the school's High Performance Computing Initiative. He is now working on his Ph.D. in Computer Science at the University of Kaiserslautern as a member of the scientific staff at the Integrated Communication Systems Lab.

Paul Müller Before studying mathematics, information-technology, and economics at the University of Bochum, Paul Müller worked as an engineer for SEL (Alcatel) with a focus on telecommunication. He started his scientific career as a researcher at the University of Tübingen where he developed a large computer-based statistical information system in 1975. In 1981 he joined the Federal Statistical Office of Germany in Wiesbaden, where he was responsible for the further development and implementation of this statistical system. Furthermore, he designed and implemented several statistical software packages during his time there. 1982 he started working with the University of Ulm where he earned his doctoral degree in mathematics in 1983. Thereafter, he was responsible for various research projects and the development of a state-wide computer network. In 1995 he accepted an offer from the University of Kaiserslautern on a full professor position in the department of computer science in conjunction with heading the university's central computing department.