Chapter 5

ONA Planning Algorithm

The ONA planning algorithm is essential. In Chapter 3 we presented the requirements for the planning algorithm. In this chapter we will show how our design addresses these requirements.

As shown in Chapter 3, heuristic search is a good approach to use for planning. The heuristic approach is fast, but it cannot guarantee that an optimal plan is found. In the worst case, a heuristic search might fail to find any plan, even though a feasible plan exists. To increase the potential for a good heuristic search, a proper set of heuristics should be developed.

5.1 Heuristic Search

As we showed in previous sections, the search tree in the space of the potential plans is huge. The plans vary in adapter selection, adapter order, and adapter locations. Among all potential plans, not all plans are feasible. The efficiency of a plan must be measured in some units, allowing the selection of a better plan. The formula that calculates the efficiency should be sensitive to the priority of different factors of the

connection, such as channel bandwidth, link security, node execution resources, etc. These may vary from network to network, and from connection to connection.

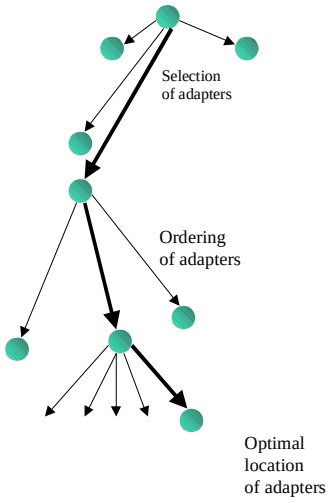

Figure 5.1: Selection of adapters, ordering of adapters, and location of adapters run sequentially

We chose to apply adapter selection, adapter ordering, and adapter location optimization sequentially, one by one. Selection introduces real adapters into the planning process making feasibility verification possible since the beginning. Ordering can be done using templates calculated off-line.

Thus, once the adapters are selected, the adapter ordering and adapter location optimization run without adapter re-selection. Once the adapters are ordered, adapter location optimization preserves their order. These heuristics prune the search tree drastically, as shown on Figure 5.1. This approach significantly reduces the space of potential plans. We chose to run the adapter selection phase first because the ordering and optimization require actual adapters to verify the feasibility of a plan. The result of

the adapter selection phase is a chain of local plans. The adapter ordering phase requires a much simpler ordering algorithm if applied to a primitive local plan than if applied to a complex plan, e.g., where adapters cover more than one link, or where one adapter overlaps multiple areas that have multiple adapters applied to them.

The result of adapter selection and adapter ordering is a chain of local plans. In the case of incremental planning, local planners run these steps on every connection node. In the case of central planning, the central planner runs these steps before it runs the last step, which is the optimization of the location of adapters. The result of adapter selection and adapter ordering in central planning can be different from incremental planning because central and incremental planning use different sources of adapters. Central planning selects adapters from adapter storage sites. Incremental planning selects adapters already located on the connection nodes. The subsequent sections show the steps of the heuristic search.

5.1.1 Adapter selection in two-level adaptation database

The adapter selection procedure consists of two basic phases:

- The interpretation of a situation in the networks and problem detection
- Choosing particular adapters to solve these problems.

In Chapter 4 we described the planning data, which consists of data stream characteristics, user application preferences, and connection conditions. The interpretation of the network situation works as follows. The planner compares the stream characteristics from one side and user application preferences and network conditions from the another side.

For example, assume that stream throughput is 2000 Kbps and the bandwidth of one of the links is 1500 Kbps. This situation is interpreted as a problem of insufficient bandwidth that requires the compression of the stream to fit the link bandwidth.

Let us consider another example. The stream is not encrypted by the user application; hence it is characterized as insecure. However, the user application preference is to keep the transferred data confidential, i.e., the data must be secure. Network conditions contain the security characteristics of the connection links. Normally they are characterized as secure or insecure. If all connection links are secure, the planner does not detect any problem. However, if one of the connection links is insecure, the planner interprets it as a problem. Therefore, if the data is encrypted by the user application, or if the user application does not require security, or if all connection links are secure, there is no security problem. If the data is not encrypted, and the user application requires security, and at least one of the connection links is insecure, the situation is interpreted as a security problem.

The problems that occur in networks are mostly well known and well described. Every problem has an assigned identification code known to the planner designers and adapter designers. Therefore, the planner has a list of the connection links associated with problem codes that occurred in the connection. Clearly, the planning system relies on this presumption of pre-knowledge of the kinds of problems that can occur in networks. Discovery of a previously unknown kind of problem might require substantial redesign of the planning algorithm and the adapters it uses. As noted, discovery of new classes of network problems is rare.

When all connection problems are detected, the planner begins the process of selecting the adapters that should solve them. The planner learns about an adapter software package from its database where each record associates a problem with a number of adapter software packages that can solve the problem (according to the adapter software package designers). Every record of the planner database contains the following data:

- Problem code
- Name of the adaptation package
- Interface to the database of the adaptation package

If the planner finds more than one adapter package that can solve a particular problem, it must choose which package to use. It can use the first package found, or a randomly chosen package, or a package that the planner has good experience with, or it can apply other self-learning techniques. A particular package may not contain an adapter of a desired efficiency. For example, if the previously chosen package does not contain a compressor that is able to compress a 500 kbps stream to pass through a link with 56 kbps bandwidth, then the planner should try searching other packages looking for a compressor of the corresponding strength.

Every adapter software package contains a software module that makes a low-level intelligent adapter search within the adaptation package's local database upon the request of the planner, using the problem code and user preferences as parameters. The database of the package is designed and maintained by the author of the package. The search module accepts the code of the problem with a preferred user-specified solution

method and calculates the result of the query containing the list of adapters that solve the problem. The list may contain one or more possible adapters from the adapter package. Each item from the list contains the name of an adapter, the set of adapter parameters, and the description of the adapter that is necessary in order to apply the adapter properly. Adapter description data is discusses in Section 5.1.2.

The design of the adapter software package database is totally up to the adapter designer. The database of the package may keep all possible adapters of the package in separate records, or it may keep the information in a form other than a table. The adapters can be stored separately from the adapter software package database, which should be located closer to the planner. The adapters can be located in a remote adapter storage site. The location field in the package database contains the location of the package adapters.

The important thing is that the database contains homogeneous data completely monitored for by adapter designers. At the same time, the planner has access to the information about adapters of the adapter package.

We have the problem of how to define a flexible but precise interface between the planner and the adaptation package. The planner uses the problem ID and user preferences in a format understandable by the search module of an adapter software database. As the result of a query, the planner receives the adapter information necessary to further the planning procedure.

If a particular adapter appears to be a bad choice for a particular situation, the planner may repeat its query to another adapter package database. If more than one

adapter is found for a particular problem, the best one can be chosen from the list by a certain criteria, such as,

- Efficiency (for example the best compressibility)
- Speed of processing
- Amount of resources needed
- Adapter availability

However, we believe that the best criteria in this case would be the suitability of an adapter for the overall plan. This means that the planner tries to incorporate the adapters into the plan from the list of adapters, one by one. If an adapter does not satisfy the plan, the planner returns to the list of adapters and chooses the next one for consideration. The first adapter that satisfies the overall plan is the one that should be chosen to solve the problem. The planner also can use a random choice of adapter to avoid the biased use of the same adapters all the time.

When a new adapter package is created and distributed, a record of the new package must be added to the planner database.

The advantages of this approach are:

- Easy update of the planning system for a new adaptation package
- Packages keep their databases according to their customs
- Fair division of the responsibilities between adapter manufacturers and planner administration

The disadvantage is that it presumes a number of serious requirements for adapter manufacturers. The interface between problem and adapter descriptions, instead of being a planner-internal issue, becomes a communal question and requires special attention.

5.1.2 Adapter description data

The adapter description data that is kept in an adapter package is important for the ordering and location planning steps. The presentation of this data must be both flexible and known to the planner and adapter designers. We designed our own metalanguage. The description of an adapter consists of three parts: action, precondition, and postcondition. This data is presented below using the following tags:

General description:

Name of the adapter – name of the adapter in question

Arguments – the arguments provided to the adapter to solve the problem

Problem code – code of the problem it solves. One example is throughput.

Throughput – which means that the adapter solves the problem of low bandwidth.

Binary - shows whether the adapter has an UNDO part or not. If **Binary** equals yes, the UNDO part must be separately described too.

Required resources - shows the list of node resources necessary to run the adapter.

Action:

Solution code – code of the method that the adapter uses to solve the problem. For example, *Lemple Ziv compression* or *color drop.*

Efficiency – efficiency of the adapter. For example, one encryptor has **efficiency** equal to 0.6 and another encryptor's is equal to 0.3. This means that the data encrypted by the first adapter is twice as easy to crack. Measuring the efficiency of adapters can be tricky. If the efficiency of adapters is not well defined, than the choice of one of two adapters can be done arbitrarily.

Amount of data - shows how much data the amount of data being transferred is affected by the adapter. For example, a compressor compresses data, a filter drops part of the data, a forward-error-correction adapter increases the amount of data, etc. If **amount of data** is equal to 1.25, the amount of data that will be transferred will increase by 25%.

Data preservation - shows how much data was lost by a lossy adaptation. For example, **data preservation** that equals 0.85 means that 15% of data was dropped and cannot be restored. If two adapters of the same efficiency have different data preservation values, than the adapter with the larger data preservation value should be selected.

Preconditions - list of preconditions that are necessary to run the adapter. For example, it can be a data format that can be recognized by the adapter, or *Compressability Lempel Ziv* is equal to 1, which means that the data can be compressed by a Lemple Ziv compressor.

Postconditions - list of postconditions of the adaptation. For example, it can be a format that the data is converted to, or compressability color drop is equal to

0, which means that the color in this video stream is dropped and cannot be further dropped. Postconditions are the results of actions applied to preconditions.

An example of a real adapter description that is implemented in this project is presented in Appendix C.

The responsibility for the creation of an accurate description of the effects that an adapter provides to the data stream is completely lies with adapter designer. If the description is not done properly, the planner will misuse the adapter or not use it at all, which is not in the adapter designer's interests.

5.1.3 Adapter ordering

The planning process described in the previous sections presumes the creation of local plans. A local plan is a plan that distributes adaptations among two neighboring nodes that are supposed to participate in the communication session. The number of adaptations that can be deployed on these two nodes corresponds to the number of data-transmission problems that can occur on the link. The process of ordering the adapters is very important because the result of their composition affects the feasibility and efficiency of the local plan.

Exhaustive search for the adapter ordering

It is easy to show that the number of adapter positions will be $N!$, where N is the number of adapters on a single node in a local plan. Because it is hard to imagine that N can be larger than 3 or 4 in practical cases, we can find the optimal order through an exhaustive search.

The effects of unary adaptations, such as format conversions, lossy adaptations, and authentication are formalized as preconditions or postconditions of adapters or planning operators (see example in Figure 5.2). In Figure 5.2a a plan operator is shown. Each operator consists of three elements: preconditions, actions, and postconditions. An operator receives the user data packet and stacked aggregated postcondition (SAP) from the previous operator. The original user data that was generated by the user application has an initial SAP that we termed "data stream characteristics" in earlier sections. The SAP varies as the data moves from adapter to adapter. If the preconditions of an operator do not conflict with the SAP, the operator is applicable directly, and the output of the operator consists of the previous SAP and the postconditions of the operator:

SAP = SAP +postconditions

Any node on the search tree should be evaluated by an evaluation function or a goal constraint. For example, for the well-known problem concerning the order of encryption/compression adaptations, we should try both orders of the compression and encryption adapters, and the evaluation function will return a better value for the compression/encryption order, since that order saves more link resources. The goal constraint is used as a threshold for the evaluation function. If the threshold is reached, the search is stopped.

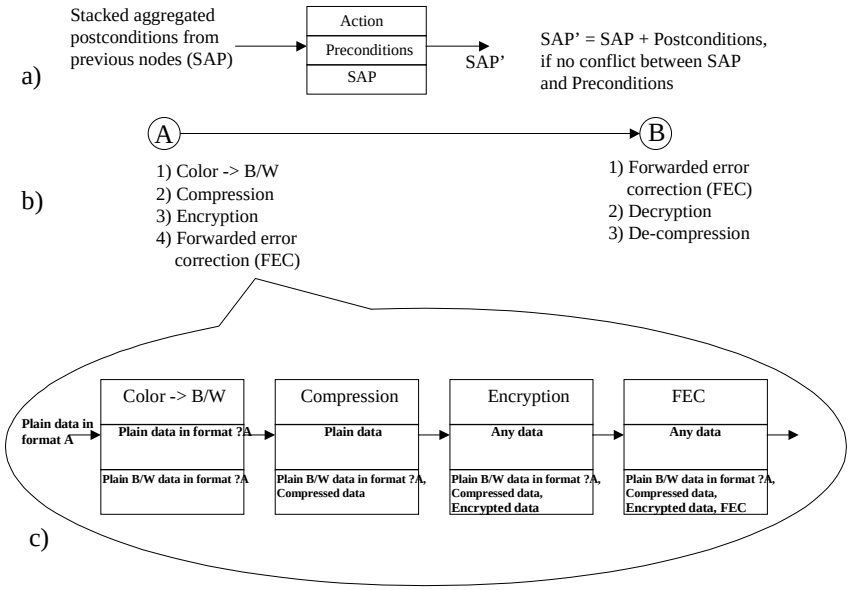

Figure 5.2: Building a local plan. a) plan operator, b) elementary network with adaptations, c) local plan of node A

Figure 5.3 shows an example of exhaustive search with the evaluation. Two adapters (compressor and encryptor) should be ordered on the connected nodes A and B. If compression is applied first, the data rate will be reduced; if encryption is applied first, the data rate will not change. In either case, the data will be transmitted with the same security. Thus, the evaluation function that calculates the amount of required link resources will return a smaller value in the first case. Another way to evaluate both options would be to compare the required resources with a threshold that indicates if success was achieved using these adapters. Assume that the threshold was 0.8 of the user data rate required. In the first case, the user data rate was reduced 0.5 times, and therefore, satisfied the threshold value; in the second case, as the data rate was not changed, the threshold was violated. Again, the first case was chosen.

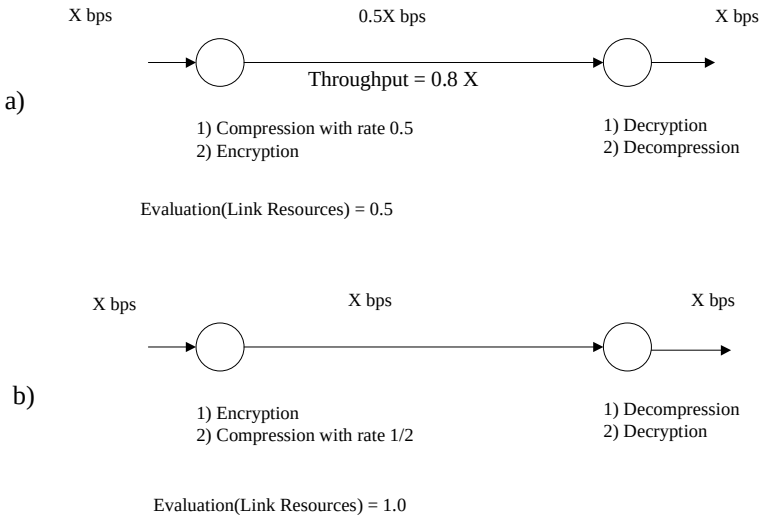

Figure 5.3: An example of exhaustive search with evaluation

Least-commitment planning

Exhaustive search might be too time-costly, since the whole planning procedure occurs in real time. For example, if the number of adapters is over ten, then hundreds of thousands of search nodes must be visited and evaluated.

Least-commitment planning is the approach that was chosen to solve this problem [Dean94], [Kambhampati94], and [Weld94]. Generic partial order plans are pre-computed and stored in a library. They are precomputed off-line, and thus the adapter designers can use exhaustive search to build these plans if necessary. The adapters in these plans are represented by generic samples of the adapters that do not have names or resource requirements. The association between these samples and real adapters is done by the adapter problem-solving methods. Whenever necessary, the plan closest to the task will be retrieved from the library and adapted. The closest generic partial plan is the

plan that contains all adapter samples that use the same solving methods as the real adapters that must be ordered. Then the real adapters are ordered in the same way as the correspondent adapter samples in the closest partial order plan.

The plans are partially ordered because some adapters cannot be ordered in advance. The constraints that allow complete ordering of the library plan are known at the moment of planning only. For example, for storage adapters such as buffer, cache, or prefetchers, there are reasons to put them before, between, or after other adapters. It depends on the particular requirements of the session. For example, encrypted data arrives at the node to be stored in a cache adapter. If the majority of end users do not want to decrypt the data themselves, the data must be decrypted before it is stored in the cache and encrypted again just for those users who have insecure links. If the majority of the users have insecure links, the data must be stored encrypted. However, in the case of unicast connections, where there is only one user, the plan can be indifferent to this problem, and a storing adapter can be ordered arbitrarily: before, between or after other adapters.

It is difficult to predict all possible constraints in precalculated library plans. These constraints become known during the actual connection establishment. If they are not defined at this moment, then the planner orders a storage adapter arbitrarily. Figure 5.4 presents the prototype of the generic partial order plan that was used for this implementation. It is easy to see that the elements of operators do not contain very many details about the adaptations. Some operators are located in parallel, which means that the order will be determined later, at the moment of plan calculation. The additional

conditions brought to the picture by particular adapters will be adjusted to the plan during the planning procedure.

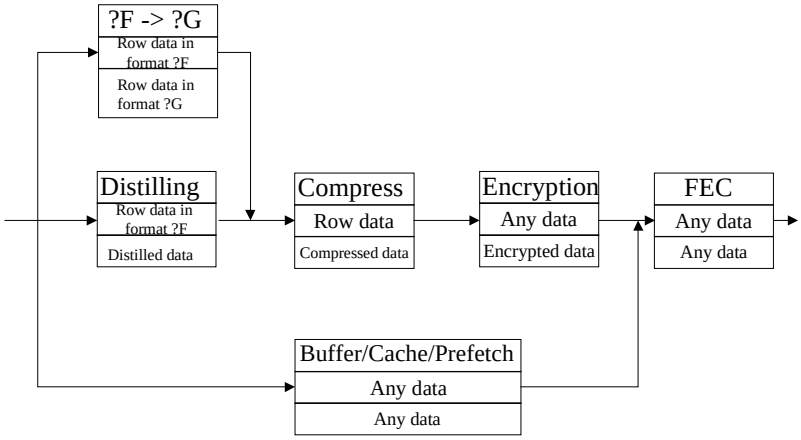

Figure 5.4: The prototype partial order library plan for adapter ordering

In the partial order plan format-sensitive adapters like the format converters shown in the figure as ?F->?G, or lossy adaptations called distillers, should be used before the data loses its format after compression, encryption, etc. Lemple-Ziv-like compression is used before encryption. The FEC adapter must be used last to ensure that the results of all adaptations provided on the link are properly encoded.

An the example of a partial order plan that represents Figure 5.4 is presented in Appendix D.

Constraints on adapter order

As shown above, a particular network condition can influence the order of adapters. However, other factors can cause more constraints on adapter order. For

example, adapter design can influence the adapter ordering. Assume that we have a one-link connection that transfers a video stream. The data must be converted from format X to format Y to meet the requirements of the user's application. The link has low bandwidth and requires a color-dropping adapter. Then the order for the X->Y format converter and color-drop filter depends on what format the color-dropping adapter requires, as described by preconditions. If it is X, the color-dropping adapter must be used before the format conversion; if it is Y, the adapter must be used after the format conversion.

Thus, the constraints in order planning come from three sources:

1. Adapter method constraints (for example, compression runs before encryption)
2. Adapter constraints formulated by adapter designers during of adapter design (for example, a format of data necessary for applying the adapter)
3. Network constraints (for example, allowing a majority of users to vote on keeping cached data encrypted or decrypted form).

The adapter method constraints are and to create partial order plans off-line. Adapter constraints and network constraints are defined later, at the moment of planning.

Conflict resolution between real adapters

Conflicts between postconditions of one adapter and preconditions of another adapter can occur during planning. Conflicts can be resolved in various ways. An approach to the conflict resolution that is based on adding another plan operator is presented in [Liatsos97]. An example of how a conflict can be recovered is shown on

Figure 5.5. Assume that two adapters, fragmentation adapter *Fragmentation* and color-dropping adapter *Color->B/W,* are in conflict because they require user data in different formats, A and B. The conflict between them is recovered by an adding an additional adapter to the plan (see Figure 5.5b). A format converter A->B was introduced between the fragmentation and color-dropping adapters.

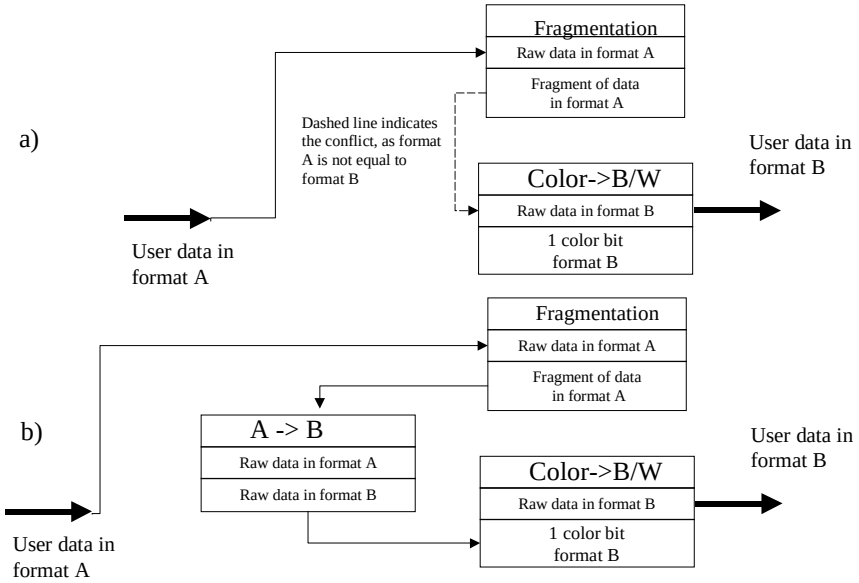

Figure 5.5: Resolution of a conflict between preconditions and postconditions
a) Planner incorporates the external constraint that user data is in format A.
b) Planner resolves conflict between the two formats by adding a format converter.

If a way to recover a conflict can not be found, the planning procedure will fail.

The result of adapter selection and adapter ordering is a chain of local plans that can be used for the connection. The incremental planning procedure stops at this point, and the plan is used for the connection. But as we saw earlier, this plan is not optimal, so central planning is applied to create an even better plan.

5.1.4 Plan optimization

Optimization of the chain of local plans consists of two tasks: extending the effects of the adapters that reduce the connection's consumption of link resources, and minimizing the number of adapters used. Minimizing the number of adapters of adapters used is desirable because every unnecessary adapter increases latency and wastes resources.

One obvious way of is to evaluate all possible plans and choose the one with the highest value. However, since our problem is NP-hard, if we have more than a few elements (nodes, links, adapters) to consider, this exhaustive approach will be computationally infeasible. We cannot predict that our system will only need to operate on small numbers of elements, so we cannot always rely on the exhaustive approach.

Recursive best-first search

Recursive best-first search (RBFS) is used for the search in the plan space [Korf93] and [Nilsson98]. The search occurs in a plan space, where each node corresponds to a plan. The search starts at some initial node. The node expands and the algorithm evaluates every possible route from the initial node to all immediate descendent nodes using an evaluation function. Then it chooses the path with minimal (maximal) value; the rest of branches are dropped. The process continues with expansion and evaluation of descendant nodes until the goal node is reached.

RBFS uses an evaluation function to guide the search for the goal plan [Dean94]. However, in our case we cannot say in advance what our goal plan looks like. So the evaluation function must provide a measure of the intrinsic value of plans in the search space, rather than comparing them to a perfect goal plan. The goal of the search then is

to find the plan with the best value of the evaluation function. This search technique can be used to address optimization problems that involve finding an object that maximizes or minimizes a particular evaluation function.

A variant of the best-first search is called the *hill-climbing* search. This algorithm is designed to maximize the optimization function and terminates in a *locally optimal* solution because no nodes of higher value are reachable by applying a single operator. However, that the solution is not guaranteed to be also *globally optimal*.

In our planning procedure we use the combination of RBFS and the hill-climbing search. RBFS allows pruning of the search tree; the hill-climbing search finds a local optimal plan, which we define as a desired solution. We will call the evaluation function the *optimization function,* which better reflects the nature of our planning methodology.

The initial plan consists of the chain of local plans. Applying transformations, we optimize this plan by:

1. Extending resource-saving adapters
2. Keeping minimal the number of links affected the adapters increase the use of network resource
3. Minimizing the number of used adapters

The process of the generating of a new candidate plan is based on the shuffling of operators of any two plans that cover two neighboring factions of a connection. A faction of a connection can contain one or more connection links. This process is called merging; the candidate plan is evaluated with an evaluation function. The process consequently merges as many neighboring plans as possible, ideally all of them. Assume

there are neighboring plans A and B. The merged plan will be PlanA+B. If the original global plan as a chain of local plans was:

PlanA – PlanB – PlanC – PlanD – PlanE –PlanF,

then the global plan after the first merge, assuming that PlanA and PlanB were merged, wiould be:

PlanA+B – PlanC – PlanD – PlanE – PlanF.

The newly built plan is closer to optimal than the original one, and if necessary it can be returned by the planner as the "optimized" plan. Otherwise the process will be resumed until, with some luck, the plan will be:

PlanA+B+C+D+E+F

When two neighboring plans are merged, the constraints of both should be preserved. If this causes a conflict that cannot be fixed by adding additional steps to the plan, the merging must be canceled and the process resumed with the next plan. For example, if PlanA+B could not merge with PlanC, then the global plan would be:

PlanA+B – PlanC+D+E+F

Merging plans

The merging of two neighboring plans is the key operation for the plan search process. As one example, two neighboring plans are located on three network nodes. Let us call the nodes A, B, and C and local plans AB and BC. The DO parts of plan AB are located on node A. The UNDO parts of AB and DO parts of BC are located on node B. The UNDO parts of BC are located on C. Each move of an adapter part is followed by the adapter ordering procedure. Thus, for example, an encryptor from node B moves to

node A and appears in sequence after a compressor already located on node A. Then the move is evaluated by an evaluation function. If the evaluation function shows a worse value, then the recently moved adapter part goes back to where it came, from and the process of merging these two plans terminates. However, it can continue with other plans that belong to the connection. The planner verifies the consistency of adapter preconditions and postconditions. If it is violated by the move, the move is rolled back and the merging of these two plans terminates. The planner verifies the feasibility of the plan by counting resources that are spent by the data stream and adapters with respect to available network resources. If the plan shows a better value of the optimization function and is feasible, it is stored as the best-found plan. If the optimization process is terminated, the best-found plan is used as a result of the plan calculation.

An example of a successful plan merge is presented in Figure 5.6a. Two local plans, one running encryption and the other one running compression, were merged, with the DO part of the compressor being moved to node A and placed before the DO part of the encryptor, and the UNDO part of the encryptor being moved to node C and placed before the UNDO part of the compressor.

Plan merging failures

The planner may not always merge two plans successfully for the following reasons:

1. Discouraged by an evaluation function
2. Insufficient knowledge about adapters
3. Unrecoverable constraint conflict

4. Insufficient computational resources of the connection nodes

5. Time limit

In Figure 5.6b, the merging of two plans failed because the cache on node B was required to store unencrypted data, since many network users don't need/want to deal with encrypted data. In Figure 5.6c, the merging of two plans failed because node C cannot execute more then one adapter.

When the planner is unable to merge plans, the optimization process may produce a fragmentally optimized global plan:

PlanA+B > PlanC+D > PlanE+F

Obviously this method cannot guarantee the optimal plan, but it can try its best to come up with a decent plan that is not worse then original global plan.

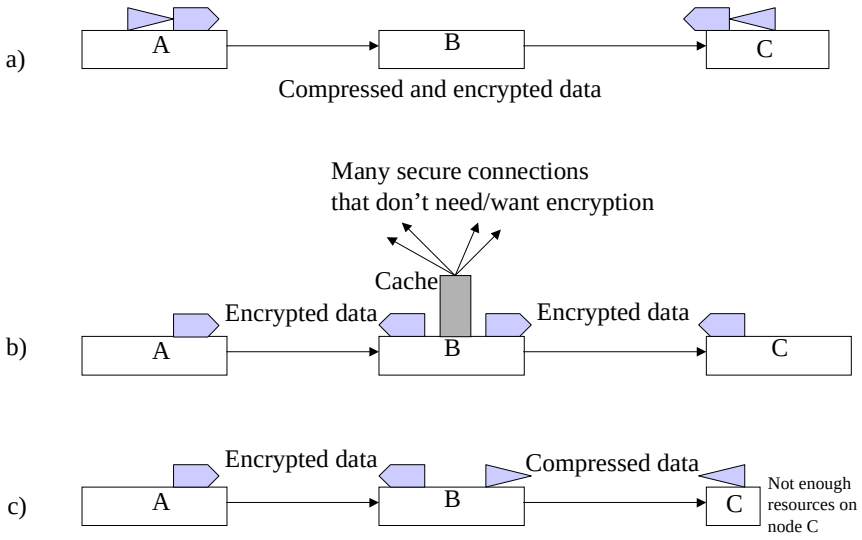

Figure 5.6: Plan merging. a) Successful merge; merge failures because of b and c. b) Network constraints. c) Insufficient resources

As mentioned above, the optimization process can be interrupted due to temporal limits, in which case the planner will return a partially optimized plan.

Plan merging policies

The result of the optimization may depend on the choice of the local plan that was chosen first for the refinement procedure. Such a local plan is called the *seed*. It is impossible to say in advance which local plan promises the most optimal result within the limited time of the optimization process. That is why the optimization process can try more than one local plan as a seed of the optimization. Several factors increase the probability that a local plan is chosen as a seed:

1. Number of adapters in the local plan
2. Particular adapters of the local plan
3. Node and link resource distribution
4. Location of the local plan toward source or destination, etc.

The influence of these factors requires further investigation. In this implementation, we use four different merging policies:

1. From source link to destination link
2. From destination link to source link
3. From the node with the most resources to destination
4. From the node with the most resources to the source and to the destination, in turn

When the planning period ends, we choose the most efficient of these four as our *central plan*.

Planning algorithm complexity

This problem is moderately complex. Assume that a global plan consists of two local plans with X and Y ordered adapters. The adapters from one local plan will be positioned next to the same-class adapters of the other local plan, making the merge complexity equal to O(X+Y). With *p* total adapters and *n* nodes, the total merging complexity is *O(pn)*. Every step of our optimization process requires an evaluation of efficiency and feasibility, which has complexity *O(n)*. Hence, the total complexity is $O(pn^2)$.

Formal definition of the optimization problem

Let us state the problem more formally:

Given:

Connection *G=(V, E), V* is a set of *N* nodes that are sequentially connected by *N-1* links *E*. Each link has certain available link resources, and each node has certain available node resources.

K is the number of link resources that are included in our model.

M is the number of node resources that are included in our model.

$Cr = \{ Cr_k(i)) \}$ is the matrix of available link resources, $1 \le i \le N-1$, $1 \le k \le K$.

$Rr = \{ Rr_m(j) \}$ is the matrix of available node resources, $1 \le j \le N$, $1 \le m \le M$.

$Fr = \{ Fr_k(i)) \}$ is a matrix of link resources required by the connection, $1 \le i \le N-1$, $1 \le k \le K$. Currently, we assume these per link requirements are equal for the whole connection.

$Gr = \{ Gr_m(j) \}$, is a matrix of node resources used for the connection, $1 \le j \le N$, $1 \le m \le M$.

If any adaptation uses more resources than available, the plan is not *feasible*.

A connection requires an adapter if, for at least one link $i \le N-1$, there exists at least one link resource $k \le K$, such that $Fr_k(i) > Cr_k(i)$

Initial state:

We can look at the adapters as tools that convert node resources into the needed extra link resources. The local plan is the chain of adapters located on the connection links:

1. $Fr_k(i) \le Cr_k(i)$, for all i and k, $1 \le i \le N-1$, $1 \le k \le K$
2. $Gr_m(j) \le Rr_m(j)$, for all j and m, $1 \le j \le N$, $1 \le m \le M$

Transformations:

Assume that there is a set of transformations $t: Gr \rightarrow Gr'$, $Fr \rightarrow Fr'$. A transformation is a move of the DO or UNDO part of the adapter from one node to another given that the original order of adapters of the initial plan must be preserved.

Rules of the transformation:

We follow four rules for the transformations.

The first rule of the transformation is to keep the original order of adapter methods according to the order of the initial plan, as described in Section 5.3.2.

The second rule of transformation is to apply an adapter's DO part on any node before the problematic link and, if there is an UNDO part, apply it on any node after that link.

The third rule points out that often there are multiple effective locations for adapters. The exact location affects the efficiency of the connection, but may be constrained by the presence of other adapters. Thus, the third rule of transformation is that an adapter can be located at a particular node only if the stream conditions and the pre/post-conditions of the adapter are consistent.

The forth rule of transformation is about merging similar adapters. A transformation can result in two adapters that provide the same action to the same set of problematic links. In this case, one of the adapters can be dropped, reducing the computational resources needed to handle the connection. Thus the rule is: drop the less effective of two adapters that provide similar actions to the same set of links.

Goal of transformations:

The goal is to find the sequence of transformations $T': t1, t2, t3\ldots \subseteq T$ that will reduce the required link and node resources, given that every link and node resource remains below the correspondent limit:

1. $Fr_k(i) \le Cr_k(i)$, for all i and k, $1 \le i \le N-1$, $1 \le k \le K$
2. $Gr_m(j) \le Rr_m(j)$, for all j and m, $1 \le j \le N$, $1 \le m \le M$
3. $\sum_{i=1}^{N} Gr_m(j) \le B_m$, $\sum_{i=1}^{N-1} Fr_k(i) \le D_k$, where B is a bound of a particular node resource m, $1 \le m \le M$, D is a bound of a particular link resource k, $1 \le k \le K$.

The last requirement expresses the intent to not just rely on available resources but also to use them below limits dictated by external circumstances.

Reducing the bandwidth of the connection links and avoiding unnecessary adaptations improves the latency of the connection. Indeed, each adaptation tends to add to latency and more data to transfer tends to add to latency, too. Minimum use of link and node resources improves the network conditions for other connections.

Each transformation changes the distribution of the node and link resources. We use an optimization function to evaluate the transformations. If the plan resulting from a transformation is feasible (i.e., does not violate node constraints on available resources) and produces a higher optimization function value, the transformation produces a better plan.

The drawback is that the procedure cannot guarantee the global minimum of the optimization function, but it can reach a local minimum [Dean 94].

Optimization function

The optimization function must consider all valuable resources that are relevant to the efficiency of the optimized global plan. We use an evaluation function of the following form to drive the optimization process:

$$f = \sum_{links} \sum_{k}^{link\ resources} \alpha_k lr_k + \sum_{nodes} \sum_{m}^{node\ resources} \beta_m nr_m,$$

where *lr* and *nr* are link and node resources, and α and β are normalizing weight coefficients. The function may not contain latency explicitly among the used node resources. Latency can be implicitly included in node resources as CPU, HD, memory, etc. used by an adapter; by favoring the reduction of the node resources, the number of adaptations and bandwidth use, the function tends to reduce latency.

The coefficients α_k and β_k are the coefficients of importance for the resource that indicate which link or node resource is more valuable in comparison with another. These coefficients can be different from network to network. If the planner is trying to build the plan on a network that consists of two partitions, where in one partition link resources are more valuable and in another partition node resources are more valuable, then the strategy of the optimization process must be different. Different values of the link and node resources can make partitioned planning (separate plans in separate partitions) necessary.

The evaluation function in the example above requires efficiency in resource distribution throughout the whole connection. The use of such a function produces an economic but very biased plan that can use the resources of nodes unfairly. The limit of how many resources of nodes can be used can be provided by the ONA accounting system.

The evaluation function is a powerful tool that can make the planning calculation adjustable to particular requirements.

5.1.5 Resource and temporal constraints

The plan calculation procedure must verify the resource constraints every time a local or global plan is created or modified. Unlike the evaluation function that observes overall resource usage, resource constraints must be verified on each link or node whose resources are used. Violation of a resource constraint may not lead directly to back-off because the merging procedure described earlier can make the plan fit the constraint. Back-off means to return to the last plan that was feasible, i.e., that fits the resource

constraint, and to resume the merging process in some alternative direction. But a plan that violates a resource constraint cannot be returned by the planner as a feasible plan. However, some resource violations can lead to back-off when it is clear that any modification other then back-off will not fix the violation.

The temporal constraint plays the role of a global limit on the plan-calculation process. If no feasible plan was calculated within the temporal limit, it means that the planning process failed. Otherwise, the best-found feasible plan must be returned no matter how much better it could be if the process proceeds. In the case of a very strict temporal constraint, the whole optimization process can be cancelled and the set of local plans returned as the only possible global plan. For a sufficiently long communication session, after a local plan is calculated and deployed, the time constraint can be made long enough to find a reasonably optimal global plan. If a local plan was not calculated successfully, the first feasible global plan should be deployed; the optimization process can continue in the background and the optimized global plan can be deployed later.

5.2 Example of a Plan Calculation

5.2.1 Peer-to-peer communication session

A peer-to-peer communication session is shown in Figure 5.7. Four nodes, A, B, C, and D, are connected with three links, AB, BC, and CD. The characteristics of these links are as follows:

1. AB requires compression because of limited link throughput
2. BC requires encryption

3. CD requires buffering on C and forward error correction (FEC).

The sender sends user data in format X at a rate of 10 Mbps. The receiver receives data in the same format and the same rate.

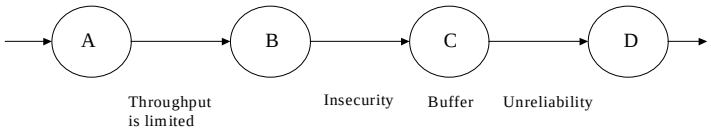

Figure 5.7: Peer-to peer communication through nodes A, B, C, D

5.2.2 Adapter selection

We assume that we have enough software adaptation packages to choose adapters for all our links. However, we found that our compressor adapter is not capable of reducing the data rate to the necessary level. So a color-dropping adapter is also chosen to contribute to the data-rate reduction in conjunction with the compression adapter. But the color-dropping adapter is designed for data format Y, which causes a conflict because the data is in format X. Therefore, the following adaptations are selected:

1. AB: color -> B/W for data format Y and compressor (color-dropping adapter is on node A)
2. BC: encryptor (encryptor is on node B, decryptor is on node C)
3. CD: buffer and FEC (buffer and FEC are on node C, DeFEC is on node D)

Note that there is a problem of conversion of the data format X into format Y. But at this step, the planner does not yet detect the problem.

5.2.3 Adapter ordering

In Figures 5.8 and 5.9 the local planning procedure for link AB is shown. The conflict between formats X and Y was resolved by adding the additional format converter X -> Y on node A and converter Y-> X on node D by the planner. In Figure 5.10, local planning procedures for links BC and CD are shown correspondingly.

In Figure 5.8 all adapters that are not needed for solving the problem are crossed out from the partial order plan. The ordered plan for node A is shown in Figure 5.9.

For simplicity, the UNDO parts are presumed but not shown in Figure 5.10. Thus, incoming data for link BC is compressed on node B and decompressed on node C, and for link CD the data is encrypted on node C and decrypted on node D.

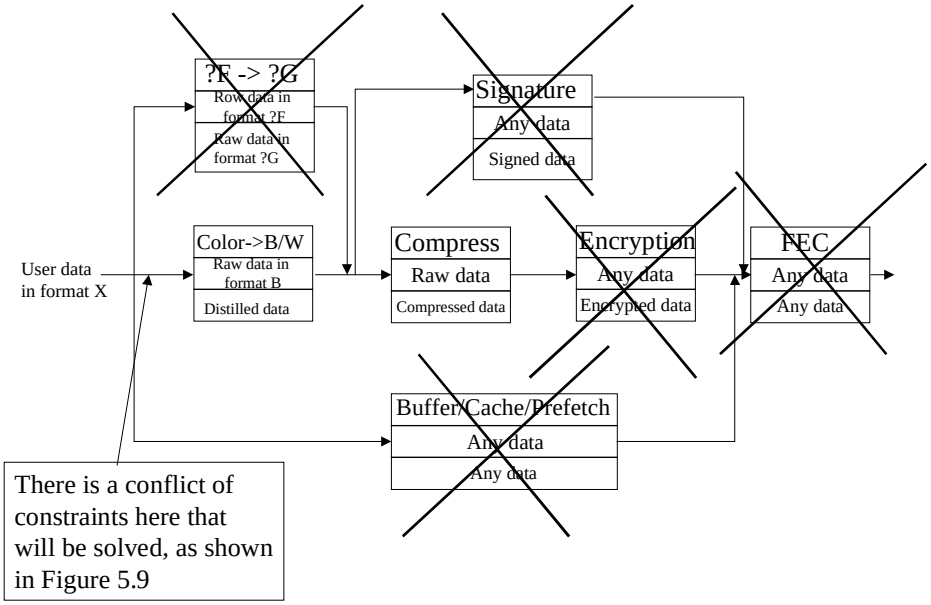

Figure 5.8: Adapting partial order plan for node A

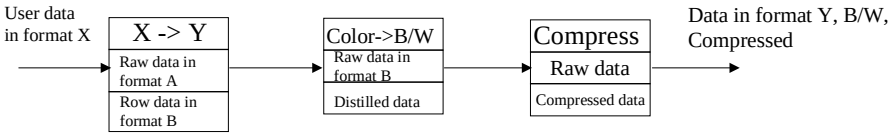

Figure 5.9: Local plan for node A

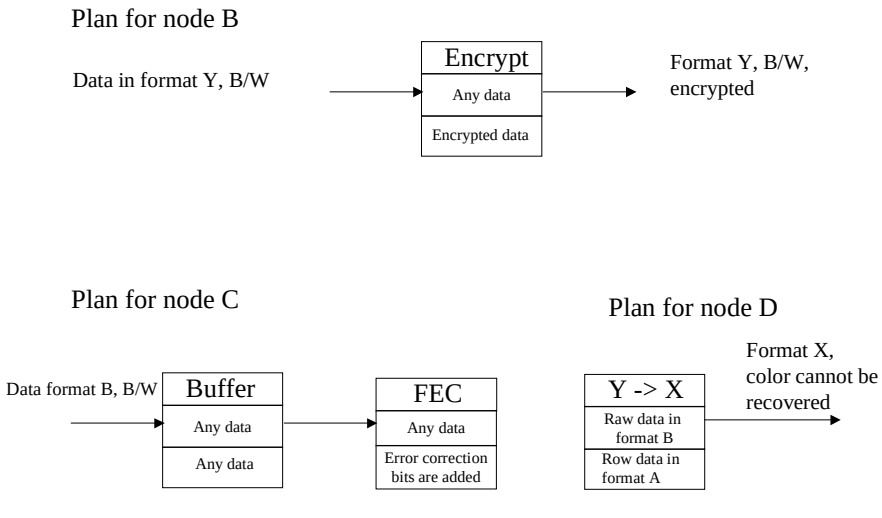

Figure 5.10: Local plan for links BC and CD. There was a constraint conflict here: data was expected in format X, and a data rate of 10 mbps. The format can be changed to X, but the data rate cannot be restored because color is not recoverable

5.2.4 Plan optimization

In Figure 5.11, the chain of local plans is presented. In Figure 5.12, merging of the AB and BC plans is shown. In Figure 5.13, plans ABC and CD are merged. The merging procedure merges the adapters from two neighbor plans using ordering heuristics formulated in the partial order library plan (Section 5.1.2). For example, adapters from nodes A and B must now be located on node A with the correct order defined in the partial order plan: the format-converter, the filter, the compressor, and the encryptor. The complexity of merging two sets of ordered adapters is MN, where M is the number of adapters in the first set, and N is the number of adapters in the second set.

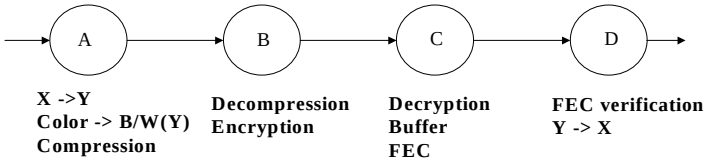

Figure 5.11: The chain of local plans on nodes A, B, C, D

In the ABC plan, encryption comes after format conversion and compression on A, B hosts no adapters, and C runs decryption and decompression.

In the ABCD plan, encryption in Figure 5.13 and compression were extended over the buffering adapters and FEC. The last adapter on D is format converter Y->X.

The evaluation function includes throughput and security in this example. Thus, compression and encryption were extended since they reduce data rate; FEC was not extended because it increases the amount of data sent.

After the evaluation of the candidate plan, it is also tested for feasibility. In this example, however, there was no node execution resource limitation, so the plan was feasible regardless of where adapters were located.

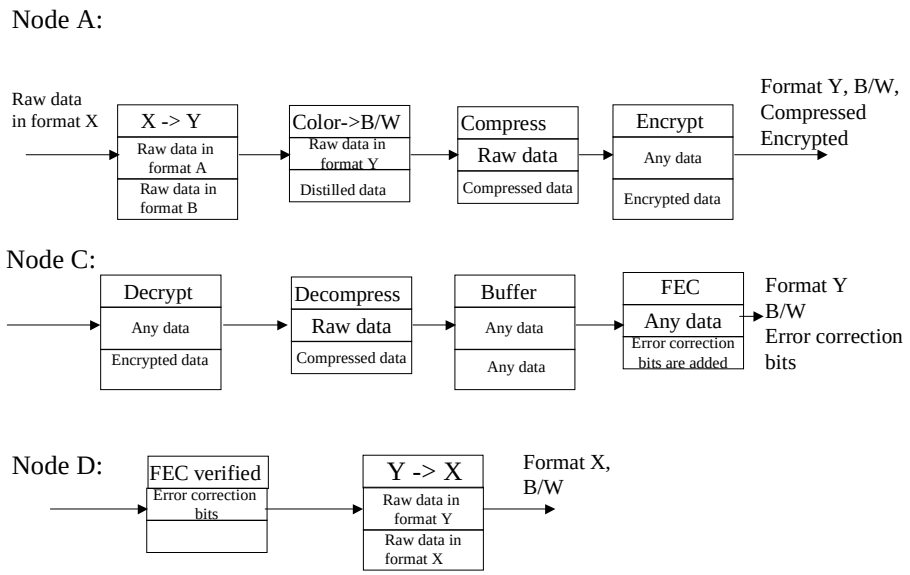

Figure 5.12: ABC plan after merging of AB and BC plans

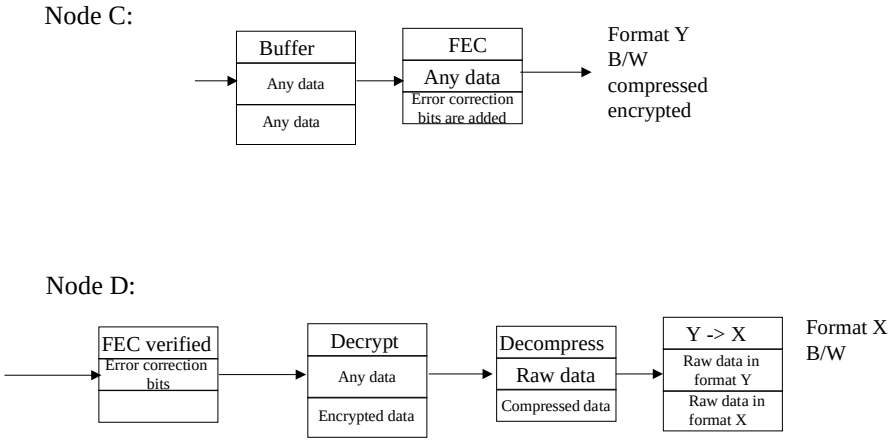

Figure 5.13: ABCD plan after merging of ABC and CD plans. Only changes on nodes C and D are shown to demonstrate the results of the last merge

5.3 Using an Unary Adapter Model for the Plan Calculation Algorithm

In the plan calculation algorithm, we used binary adaptation. However, we can consider a purely unary model where each part of a multiple-part adapter is assumed to be a separate adapter. The unary adapter model requires a slight revision of our model. The unary model looks promising because it may be the only way to build a plan for multicast transmissions, because the multiple second parts of binary adaptations can cause difficulties for the planner. The constraint that initiates the insertion of the UNDO part of a binary adaptation will have to be calculated through polling all leaves (users) that share a particular branch of the multicast tree. The multicast case will be considered in Chapter 8.

5.4 Summary

In this chapter we presented the problems of the planning algorithm. The planning algorithm is the key issue for ONA planning. Three steps of the planning algorithm were presented.

Heuristic search allows a fast calculation of the plan to address temporal constraints on the planning process. The procedures of adapter selection and adapter ordering address the problem of data and adapter consistency. Optimization of a plan addresses the problem of plan efficiency. The problem of planning system extensibility is handled by the encapsulation of adapter selection into adaptation packages with known to the planner interfaces between packages and the planner.

In Section 6 that follows, we present the measurement performance of an implementation of such a planner and an ONA system that uses it.