ATP: Autonomous Transport Protocol

Tamer Elsayed, Mohamed Hussein, Moustafa Youssef, Tamer Nadeem, Adel Youssef, Liviu Iftode
Department of Computer Science and UMIACS
University of Maryland
College Park, Maryland 20742
{telsayed,mhussein,moustafa,nadeem,adel,iftode}@cs.umd.edu
UMIACS-TR-2003-52 and CS-TR-4483
May 19, 2003

Abstract

In this report we present the design of the Autonomous Transport Protocol (ATP). The basic service provided by ATP is a reliable transport connection between two endpoints (identified by content identifiers) independent of their physical location. Autonomy allows dynamic endpoints relocation on ***different*** *end hosts without disrupting the transport connection between them. ATP depends on the existence of an underlying Instance-Based Network (IBN) to achieve its goals.*

An IBN provides the flexibility of having different instances of the same content. It is up to the user of the IBN network to define the relation between these instances. An IBN allows its user to to map a content to a particular node. Application endpoints can send messages to other content-identified endpoints. Routing in the IBN is instance-based; the IBN can route a message to a specific content instance or to the nearest instance, if no exact match is found for the destination content instance. Moreover, the IBN replicates the stored contents in order to provide fault tolerance and IBN nodes along the query path can cache a content to provide fast answers to future queries.

The ATP layer in the intermediate nodes between the source and destination endpoints can actively participate in the connection, for example, to buffer data for the destination endpoints during periods of unavailability. Data is transferred by a combination of active and passive operations, where the ATP layer of a node can decide whether to actively push the data to the destination or to passively wait for the destination endpoint to pull the data. The decision to whether to use the active or passive modes can be taken by a local policy on the node running the ATP protocol.

Keywords: *Content-based networks, instance-based networks, mobility management, transport protocols, ubiquitous computing*

1 Introduction

Advances in wireless networking, device miniaturization, and mobile computing have made ubiquitous computing possible more than ever before. The ubiquitous computing era will challenge our way of thinking and computing for more than the PC revolution did in the past. Users want to move everywhere and seamlessly access services and resources located in the environment. For this to happen, user applications need to migrate from one machine to another and the user network connections should be maintained during these migrations.

In a true ubiquitous environment, users should be allowed to change networks and hosts seamlessly and communication should continue even if the user is not available for a short period of time. Current Internet architecture offers a limited support for ubiquitous communication. In traditional TCP/IP, the connection is identified by the IP address of the end hosts and the user is bound to the same host during the connection lifetime. Although the mobile IP protocol [5] provides a solution to the host mobility between different networks, a user is bound to a single host during the lifetime of a connection. As ubiquitous computing emerges, the user should become the focus of communication and not the end hosts.

To achieve this goal, connections should be carried between users, independent from the host on which the user is located. Peer-to-peer lookup services provide a mechanism to map a content into a specific host and to query the location of this content in a peer-to-peer (P2P) network. Considering a user's endpoint as a content, a transport layer protocol over a P2P network uses the lookup service mechanism to locate the end hosts. The challenge is how to maintain a reliable connection between the users' endpoints as they roam in the environment moving from one host to another leading to a dynamic change of the user ID (content) to host mapping. In such an environment, data for the same connection can be produced at different hosts and consumed at multiple locations depending on the location of the user endpoint. In TCP , this mapping is static during the lifetime of the connection.

In this report, we introduce the Autonomous Transport Protocol (ATP). In ATP, autonomy allows dynamic endpoint relocation on different end hosts without disrupting the transport connection between them. The ATP has the following features:

- It does not enforce any naming scheme on the user application. The application is responsible for uniquely identifying the endpoint.

- The endpoints of a transport connection are defined as contents in the P2P network. This decouples the connection from the physical host where the user endpoint is located, and hence ensures autonomy.

- Mobility of the endpoints is handled via the P2P network by dynamically changing the mapping between the endpoint and the host. The ATP layer is responsible for moving segments to the destination and the acknowledgment to the source regardless of their current mapping in the P2P network.

- Since a P2P network is built as an overlay network, the ATP layer in the intermediate nodes between the source and destination endpoints can actively participate in the connection, for example, to buffer data for the destination endpoints during periods of unavailability.

- Data is transfered by a combination of *active* and *passive* operations, where the ATP layer of a node can decide whether to actively push the data to the destination or to passively wait for the destination endpoint to pull the data. The decision to whether to use the active or passive modes can be taken by a local policy on the node running the ATP protocol.

This report is organized as follows. In Section 2 we introduce an example scenario to illustrate the problem we are trying to solve and motivate the ATP protocol. Section 3 describes the system architecture. We introduce an extension to the current P2P lookup services, the Instance-Based Network (IBN), in Section 4. Section 5 introduces the ATP protocol. In Section 6, we present different applications that can benefit from the ATP protocol. We survey related work in Section 7. Section 8 gives a discussion about the ATP design. Finally, Section 9 presents the conclusions and the extension to the current work.

2 Example

Figure 1 shows an application scenario for the ATP. A set of wireless nodes, represented by small black circles, are distributed over an area. An observer *Src* generates statistical data about the environment as it moves around. This data is transmitted reliably to an analyzer *Dst*.

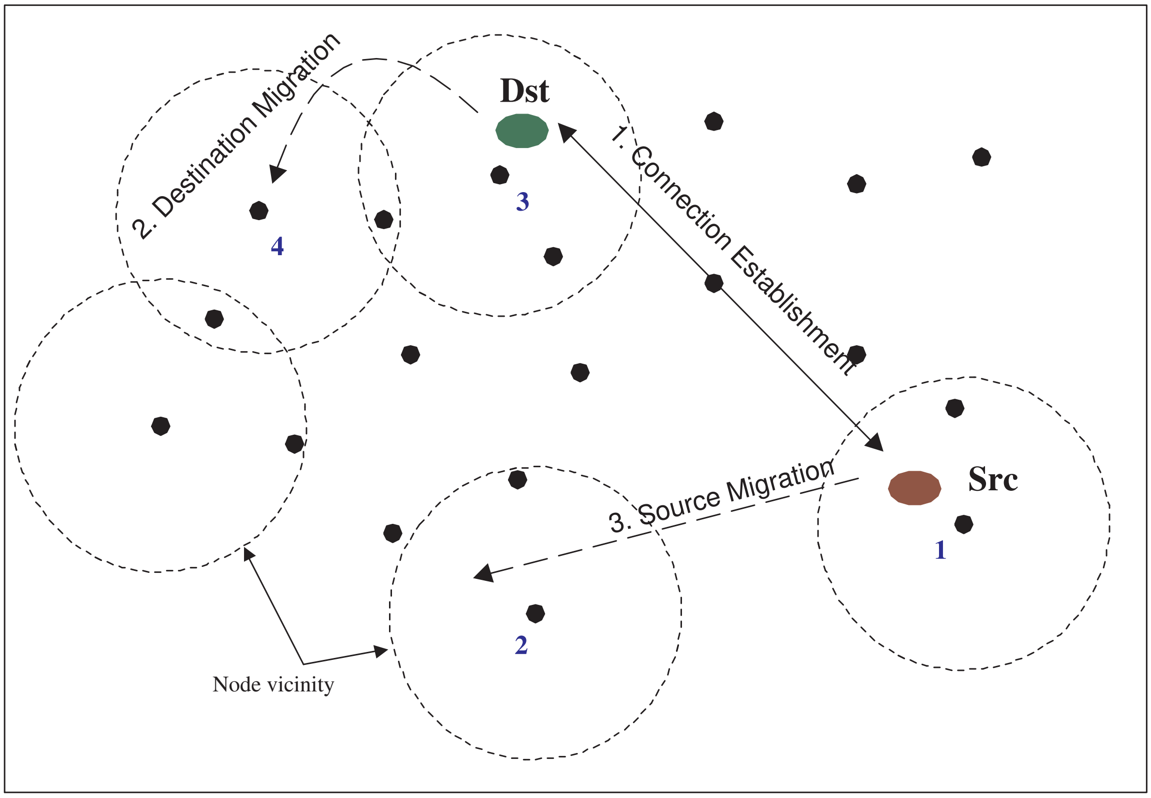

Figure 1. A typical ATP application scenario

The observer triggers the tracking application of the nearest node to become active (Node 1 in Figure 1) in which it records and transmits the collected data to the analyzer (Node 3) using the ATP layer. During the observer movement, it can become out of the vicinity of any tracking node. For example, while the observer moves from node 1 to node 2 (*source migration*) as in Figure 1, temporally it does not come in the vicinity of any node. Once the observer becomes out the vicinity of Node 1, the tracking application stops tracking and the ATP layer on node 1 handles the established connection to continue reliably transmitting the packets it has to the analyzer. When the observer approaches node 2, it triggers the tracking application on that node to collect the data and to transmit them to the analyzer using the ATP layer of node 2. Meanwhile, ATP layer at Node 1 may still be transmitting the rest of the data it has to the observer.

Similarly, The analyzer changes the monitoring node from Node 3 to Node 4 (*destination migration*) as in the figure. As analyzer moves from node 3 to node 4, the ATP layer at node 3 continues the connection on behalf of him. When the analyzer connects to node 4, the connection is updated to forward the data to the new physical location of the analyzer. ATP layers at node 3 and 4 are responsible for transferring the data from node 3, which is not shown yet to the analyzer, to node 4. Due to the spatial locality of nodes 3 and 4, transferring the data between those two nodes has lower overhead than transferring the data from the observer. In this scenario, the observer and the analyzer do not need to be aware of the physical location of each other and the roaming handling should be transparent to both.

3 System Architecture

Figure 2 shows the system architecture for an ATP environment. The ATP protocol stack consists of four layers:

1. The underlying-network layer
2. The IBN layer
3. The ATP layer
4. The application layer

In the following subsections, we introduce each layer.

3.1 Underlying Network Layer

This layer presents the communication infrastructure for our system. This can be an IP infrastructure (such as the case in the Internet), an ad-hoc network (such as the case in MANET [1]), or any other underlying network.

3.2 Instance-Based Network (IBN) Layer

We define content-based network (CBN) as a network of endpoint entities called *contents* where each content is addressed or located by its name, properties or attributes, independent of its physical location. The content could be a user, an application service, a document, a network node, a network connection or any other object. Unlike IP networks where the IP address is not just a unique ID but also a locator, CBN addressing is decoupled from the location of contents. Contents can actively communicate with each other by sending or receiving messages, or performing a lookup for other contents. Other content types, such as a document, can be passively stored in the network.

The CBN layer extends the functionality provided by the current peer-to-peer lookup services (such as CAN [6], Chord [10], Pastry [7], and Tapestry [14]). Peer-to-peer lookup services provide a mechanism to map a key to some

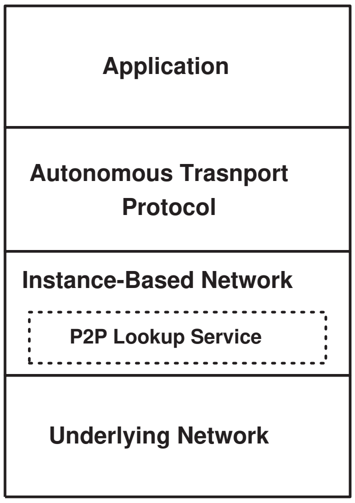

Figure 2. System architecture

node in network specified by the lookup service, and allows the users to query for these keys. The CBN, however, maps a content to a *specific* node in the network and routes messages to this node.

We developed an enhanced content-based network named as the *instant-based network (IBN)*. The IBN has the unique feature of allowing different instances of the same content to be stored in the network (and hence the name IBN). Details of the IBN layer are discussed in Section 4.

3.3 Autonomous Transport Protocol (ATP) Layer

The main goal of the Autonomous Transport Protocol is to offer a reliable transport protocol over the IBN. Migration of endpoints should be transparent to each other; that is, the protocol should be able to maintain the connection during the migration of endpoints. The protocol should maintain established connections seamlessly and independent from intermediate node availability. In other words, our proposed protocol should have the same functionality of TCP in static networks while confronting the dynamics of the IBN environment. The details of the ATP protocol are discussed in Section 5.

3.4 Application Layer

ATP-aware applications communicate with the ATP layer to perform migration of the endpoints. The ATP layer is responsible for migrating the open connection state to the new node, while the application is responsible for migrating the application state. To minimize the required changes of the existing applications to be ATP-aware, the interface between the ATP layer and the application layer is similar to the TCP socket interface with the addition of two functions: *migrate* and *land*. An application that wants to migrate calls the "*migrate*" API function to inform the ATP layer of its intention to migrate. When the application migration is done, the application needs to call the "*land*" API function to notify the ATP layer on the new node to restore the open connections. The ATP-Application layer interface is described in Appendix A.

4 The Instance-Based Network

In this section, we propose a new Instance-Based Network (IBN). The proposed network provides the flexibility of having different instances of the same content. It is up to the user of the IBN network to define the relation between these instances. For example, a file archiving system can use the new IBN to store different versions of the same file. Here, the file name represents the common content and the different versions represent the different instances of the same file. A user of this file archiving system may want to query for a particular version of the file (instance) or ask for the latest version of the file. This feature is unique in the proposed IBN compared to any P2P lookup service.

In the balance of this section, we introduce the features of the proposed IBN and then present its addressing and routing schemes required to achieve these features.

4.1 Features/Goals

The proposed IBN enhances the content-based network that extends the functionality of the current peer-to-peer lookup services [6, 10, 7, 14], which provides a mechanism to map a content identifier to a specific node. IBN support the following functionalities:

- *Content-node mapping*: The IBN user can ask the IBN to map a content to a particular node (through a publish operation). All mapping are leased, that is, if the user does not refresh the lease before it expires, the content is removed from the IBN.
- *Content communication*: Application endpoints, defined by contents, can send messages to other content-identified endpoints.
- *Instance-based routing*: The IBN can route a message to a specific content instance or to the nearest instance (the neighborhood metric is defined below) if no exact match is found for the destination content instance.
- *Replication*: The IBN replicates the stored contents in order to provide fault tolerance.
- *Caching*: Nodes along the query path can cache a content to provide fast answers to future queries.

Figure 3 illustartes the architecture of an IBN node model showing its main components.

4.2 Addressing

A content of the new IBN is addressed using a name X and an instance identifier $(i_1, i_2, ..., i_n)$, where $i_1, ..., i_n$ are n integer numbers (Figure 4). We use the notation $(X : i_1, ..., i_n)$ to refer to an instance of a content X. The semantics and dimensionality (n) of the instant identifier tuple is assigned by the user of the IBN network. These semantics include the ordering relation between different instances. For example, in a file archiving system, a file name can be represented as $(\text{logfile} : 1, 0, 1)$ to represent the version 1.01 of the file *logfile*. These semantics are assigned by the file archiving system.

The user of the IBN network maps a content instance to a particular node. The next subsection describes how the routing is performed in the proposed IBN.

4.3 Routing

The routing in the proposed IBN network is instance-based. A message destined to content $(X : i_1, ..., i_n)$ is routed to the nearest published instance of the same content X to the destination instance. The routing algorithm of the IBN is illustrated in Algorithm 1. The function *Closest* in the algorithm represents one possible ordering relation between different relations. This is the comparison function used by the ATP protocol.

The proposed IBN extends the routing techniques of the current P2P lookup services. Appendix B shows the API that the CBN network exports to upper layers. Appendix C presents a possible implementation for the proposed CBN.

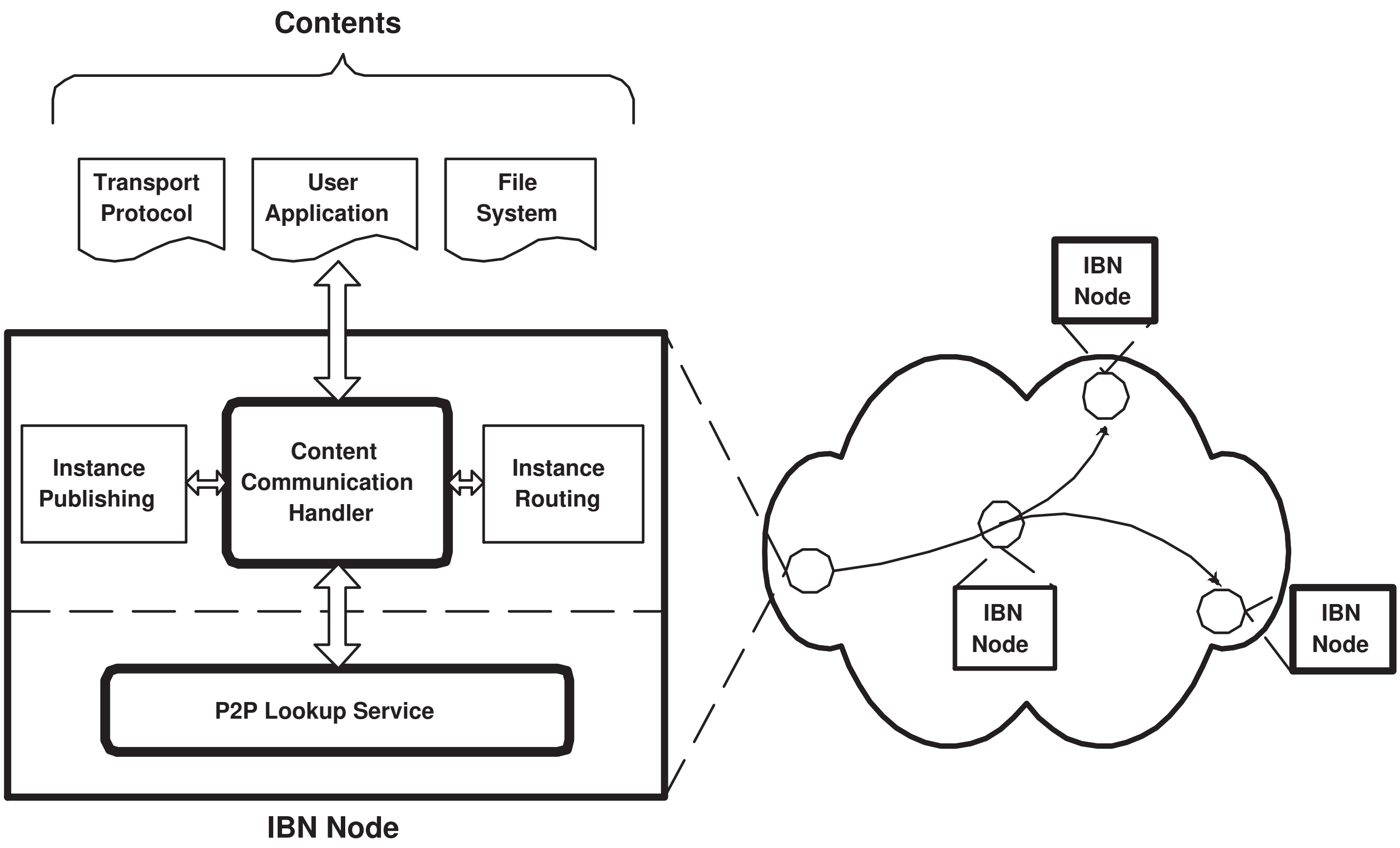

Figure 3. IBN node architecture

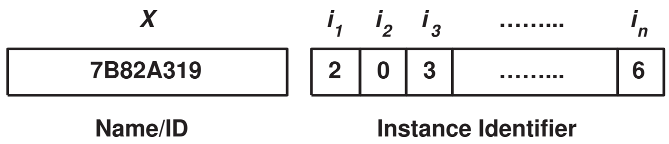

Figure 4. Instance-Based Naming

Alg. 1. IBN_Route (*Msg*, ($X : i_1, ..., i_n$))

if Content is published in the IBN network **then**
  if $(X : i_1, ..., i_n)$ is published in the network on node N **then**
    Route *Msg* to node N.
  else
    Msg is routed to $(j_1, ..., j_n)$= Closest $(i_1, ..., i_n)$ where *Closest* is defined as:

$$\text{Closest}\,(i_1, ..., i_n) = \begin{cases} (j_1, ..., j_n) & : \; n > 1 \text{and} (j_1, ..., j_{n-1}) = \text{Closest}\,(i_1, ..., i_{n-1}) \text{ and} \\ & : \; (j_n \text{ is the maximum number } \leq i_n \text{ such that} \\ & : \; (j_1, ..., j_n) \text{ is a prefix of the indices of an existing instance)} \\ a & : \; n = 1 \text{ and } a \text{ is the maximum number } \leq i_1 \text{such that} \\ & : \; j_1 \text{ appears as a first index of an existing instance)} \end{cases}$$

  end if
else
  Msg is dropped.
end if

5 Autonomous Transport Protocol

The main goal of ATP protocol is to decouple the operation of the endpoints and maintain a reliable communication between them. In this section, we introduce the details of the ATP protocol.

5.1 Overview

A connection in the ATP protocol is established between two endpoints which are identified by *content ID*'s. Endpoints could migrate or temporarily disappear from the network, and the data *segments* and *acknowledgments* should continue to flow between them. For the sake of simplicity of presentation, we assume that all connections are simplex. Extension to the full-duplex case is straight forward.

A typical communication scenario is shown in Figure 1[1]. A source endpoint (*Src*), with content ID S, establishes a connection with a destination endpoint (*Dst*), with content ID D (Step 1). When the destination endpoint migrates to a new node (Step 2), the ATP layer on the old node spawns an *agent* that acts, at the ATP layer, on behalf of the destination to buffer any received data and to send acknowledgments. The ATP layer on the new and old nodes cooperate to make the migration transparent to the source endpoint. Similarly, when the source endpoint migrates (Step 3), the ATP layer on the old node spawns an agent to take care of sending any data in the buffer and to receive acknowledgments. The ATP layers on the new and old nodes cooperate to make the migration transparent to the destination endpoint. The migration step can be performed multiple times and there can be multiple agents working for the same endpoint at any time.

The ATP layer of the source endpoint, whether it is in the original node or any other agent, can take the decision to participate actively in the connection or to wait passively for the destination to pull the data. In the former case, the node publishes itself as *AS* while in the latter the node publishes itself as *PS*. Since the operations of the source and destination are decoupled, the default mode of the agents acting on behalf of the source is to be in the active mode. Agents acting on behalf of the destination in receiving data should buffer this data until the destination appears on the new node. Therefore, these agents should be in the passive mode until the destination reappears

[1]For simplicity of illustration, we assume that an application message fits an ATP segment. Moreover, we assume that the ATP acknowledges a segment number whereas in the actual implementation it acknowledges an octet number.

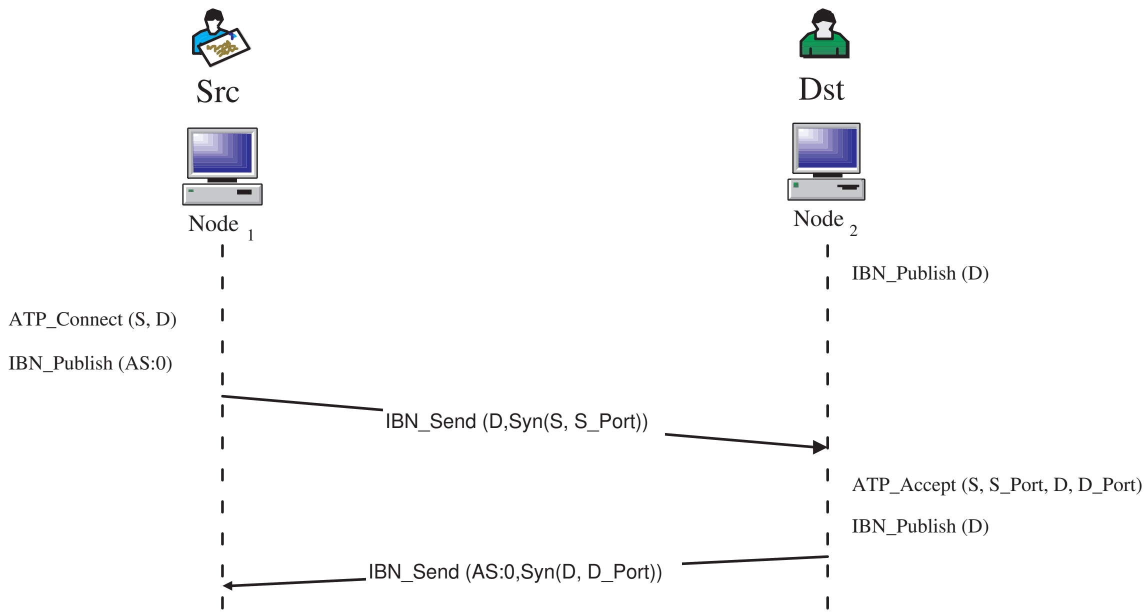

Figure 5. ATP connection establishment phase

and request the buffered data. A sender in the passive mode can arbitrarily choose the length of the data to be sent before waiting for another pull or the requester can determine the desired length (or its limit) in the pull message.

Each agent has a unique name composed of the original content ID plus the starting sequence number of the data it is responsible for. For example, *AS:4* denotes an agent for the source endpoint in the active mode responsible for the segment starting with sequence number 4. This naming of the agents is supported by the underlying IBN where different instances (agents) of the same content S are identified by the sequence number of the starting segment the agent is responsible for. The following subsection describes the operations of the ATP protocol in more details.

5.2 Connection Establishment

Figure 5 shows the steps involved in the connection establishment phase of the ATP protocol. We assume that *Dst* has already registered itself with the IBN by calling "*IBN_PUBLISH(D, [], Node*$_2$*)*", otherwise, the connection establishment will fail returning "*Destination Unreachable*". The connection establishment phase starts by *Src* calling "*ATP_Connect(S, S_Port, D)*" where *S_Port* is a unique port number for each connection in which S is involved [2]. The ATP layer spawns an agent (*ATP_Agent*$_1$) which becomes responsible for the connection that is being established. The agent calls "*IBN_PUBLISH(AS:0*[3]*,[], Node*$_1$*)*" indicating its presence on Node$_1$ and that it will be in the active mode starting from sequence number 0 [4]. Following that, *ATP_Agent*$_1$ sends a connection establishment message to *Dst* by calling "*IBN_SEND(D, Syn(S, S_Port))*". The IBN routes this message to Node$_2$ where Dst is currently located. When *Dst* receives this message, it extracts the S and *S_Port* fields and calls "*ATP_ACCEPT(S, S_Port, D, D_Port)*". This function make the ATP layer at Node$_2$ spawn an agent (*ATP_Agent*$_2$). This agent publishes *D* by calling "*IBN_PUBLISH(D,[],Node*$_2$*)*". Following that, *ATP_Agent*$_2$ calls

[2] This can be implemented as a counter maintained by the ATP layer which is incremented for each connection that S is involved in.
[3] The published identifier should be AS:S_Port:0, but we drop the port number part here for simplicity of illustration.
[4] the starting sequence number can be chosen arbitrary

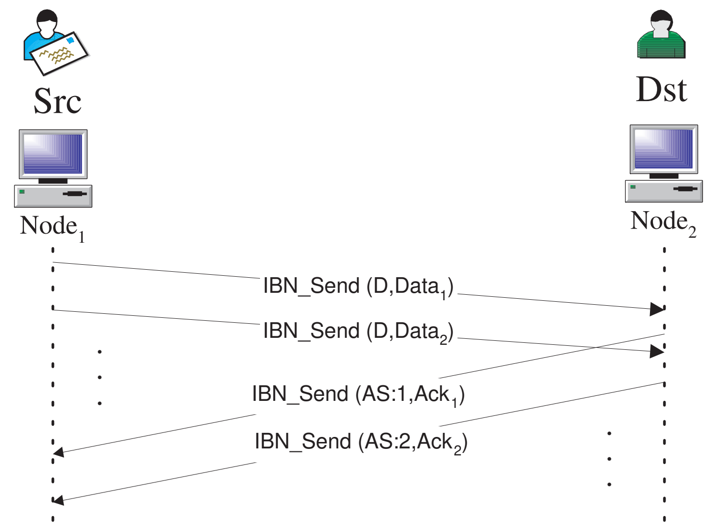

Figure 6. ATP basic mode

"*IBN_SEND(AS:0, Syn(D, D_Port))*" so that *Src* can know about the port that *Dst* is using for this connection. *Src*, *Dst*, *ATP_Agent*$_1$, and *ATP_Agent*$_2$ store the connection identifier "*<S, S_Port, D, D_Port>*" that uniquely identifies the connection for later use[5].

5.3 Basic Mode

Figure 6 shows the operation of the ATP protocol in the basic mode. In this mode, neither *Src* nor *Dst* migrates. Operation in this mode is very similar to the operation of the TCP protocol over the IP networks. Segments generated by *Src* are sent by *ATP_Agent*$_1$ with destination address *D*. An acknowledgment from *ATP_Agent*$_2$ for sequence number k is sent to *AS:k*. The ATP layer uses the underlying IBN to route the acknowledgment message to the node responsible for the nearest instance to *AS:k*. In Figure 6, this node is *Node*$_1$ which is responsible for AS:0.

5.4 Source Migration

Figure 7 shows the operation of the ATP protocol when the source migrates from *Node*$_1$ to *Node*$_3$.

Before migration: *Src* informs *ATP_Agent*$_1$ of its intention to migrate by invoking the ATP function "*ATP_MIGRATE()*". Upon the invocation of this function, *ATP_Agent*$_1$ stores the current status of all connections, involving *Src* as one of the endpoints, in the IBN.[6]. The status contains the last sequence number buffered in the ATP layer by *Dst* on *Node*$_1$.

During migration: The content *AS:0* is still published in the IBN and attached to *ATP_Agent*$_1$ at *Node*$_1$. *ATP_Agent*$_1$ is responsible for transmitting the remaining data in the ATP sending buffer, accepting acknowledgments from the destination, managing the transmission window and retransmitting segments if necessary.

Upon landing: When SRC lands on *Node*$_3$, it invokes the ATP function *ATP_LAND(S)*. A new ATP agent is spawned to be responsible for the connection on the new node. Let's call the newly spawned ATP agent

---

[5]The connection establishment uses a two-way handshake as it provides a simplex connection. For the case of two-way communication, three-way handshaking should be used.

[6]What is the ID of the stored status? One proposed ID is *AS:-1* and when the new ATP agent is spawned, it will get the stored connections status, unpublish *AS:-1* and publishes it again.

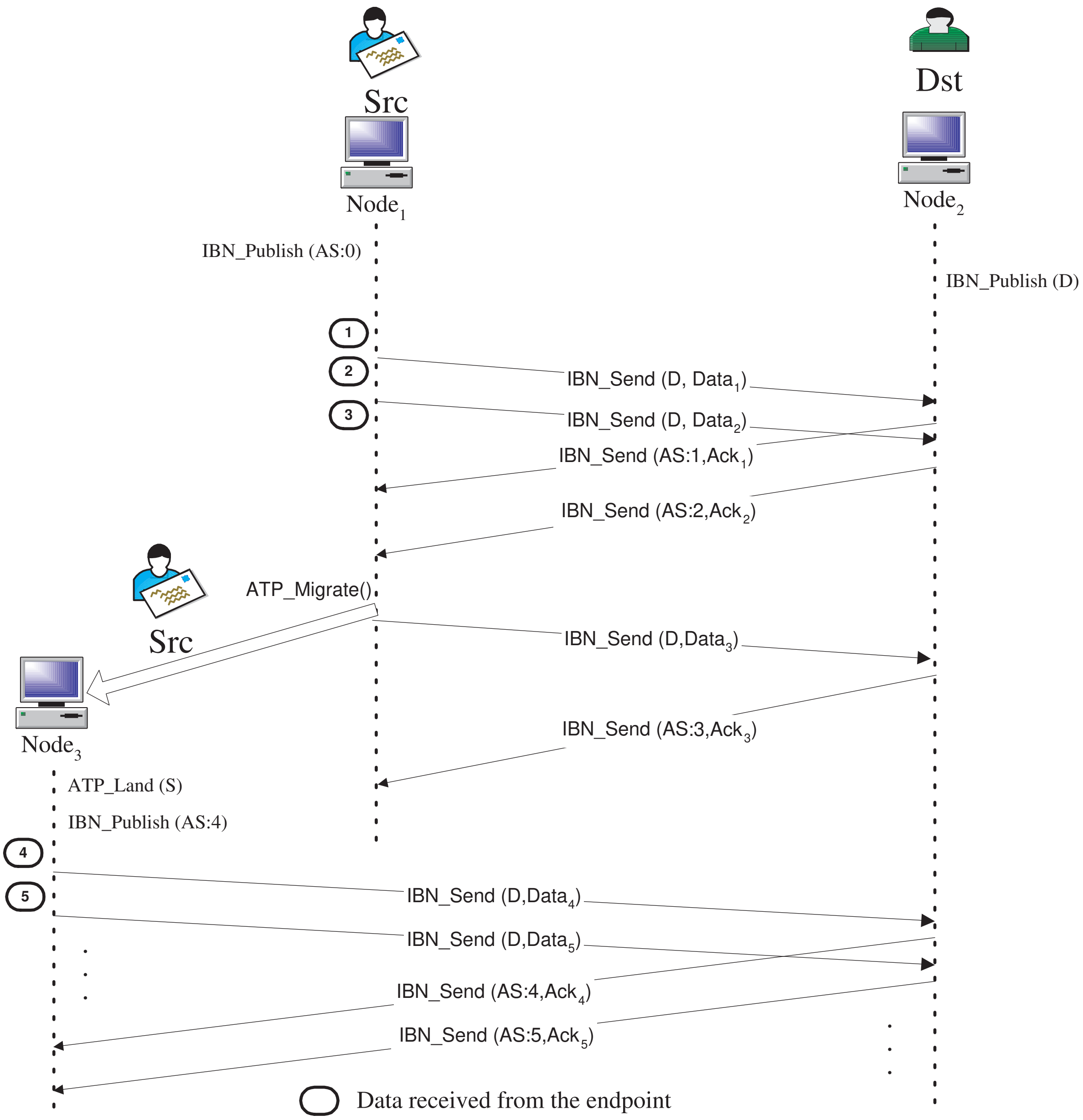

Figure 7. ATP source migration

ATP_Agent$_3$. *ATP_Agent*$_3$ starts by restoring the status of all connections that *Src* is involved in (only one connection in Figure 7 for simplicity) from the IBN. At that point, knowing its starting sequence number, *ATP_Agent*$_3$ publishes a new content in the IBN with ID *AS:j*, where j is the sequence number obtained from the stored status ($j = 4$ in Figure 7), and attach this content to *Node*$_3$ by calling "*IBN_PUBLISH(AS:j, [], Node*$_3$*)*".

After landing: *Src* starts sending data. A cumulative acknowledgment[7] from *ATP_Agent*$_2$ for segment k is sent with destination address *AS:k*. If k is less than j, the acknowledgment is routed to *ATP_Agent*$_1$ (using the instance-based routing feature of the underlying IBN). Similarly, if k is greater than or equal to j, the acknowledgment is routed to *ATP_Agent*$_3$.

5.4.1 Multiple migrations

Figure 8 shows the operation of the ATP protocol when the source makes multiple migrations from *Node*$_1$ to *Node*$_3$ and finally to *Node*$_4$. This is a direct extension to the case described above.

5.5 Acknowledgment

The cumulative acknowledgment used in TCP is adequate for the communication model assumed in IP networks in which we have a single source and a single destination for each connection. Due to the flexibility of endpoints migration in the ATP protocol, we might end up having interaction between many source and destination agents, as explained in the next subsections. For this model of communication, the cumulative acknowledgment is no longer suitable. Figure 9 shows an example of why cumulative acknowledgment alone is not adequate. In the example, the source sends segment 1 and then migrates to *Node*$_2$ where it sends segment 2. The acknowledgment for segment 1 is lost and even if the agent at *Node*$_1$ retransmits segment 1, all acknowledgments will be sent to *Node*$_2$. The agent at *Node*$_1$ will retransmit until its lifetime[8] is over, even though the segment reached the destination successfully leading to a significant loss of bandwidth.

ATP solves this problem by requiring that any duplicate segment to be acknowledged separately. In this case, the acknowledgment reaches the correct node. If an acknowledgment for segment k is lost and the original segment is retransmitted, a cumulative acknowledgment of the **last** segment received is with destination address *AS:k*. This is consistent with TCP behavior except that the acknowledgment is sent to the node responsible for the duplicate segment and not to the node responsible for the last sequence number.

Therefore, sending of acknowledgments is triggered by either:

- expiration of a timer (or alternatively, reception of a certain amount of data): in this case, the ATP protocol sends a cumulative acknowledgment for the segments it received.
- reception of a duplicate segment: in this case, the ATP protocol sends a cumulative acknowledgment for the node responsible for the duplicate segment.

For the remaining of the report, we use the acknowledgment policy described in this section. Other acknowledgment policies, mainly for range acknowledgments, are discussed in Appendix D.

5.6 Destination Migration

Figure 10 shows the operation of the ATP protocol when the destination migrates from *Node*$_2$ to *Node*$_3$. The ATP protocol works as follows:

Before migration: *Dst* informs *ATP_Agent*$_2$ of its intention to migrate by calling *ATP_MIGRATE()*. In response, *ATP_Agent*$_2$ does two things:

[7] ATP acknowledgments are discussed in Section 5.5

[8] We discuss agents' lifetime in Section 5.7.

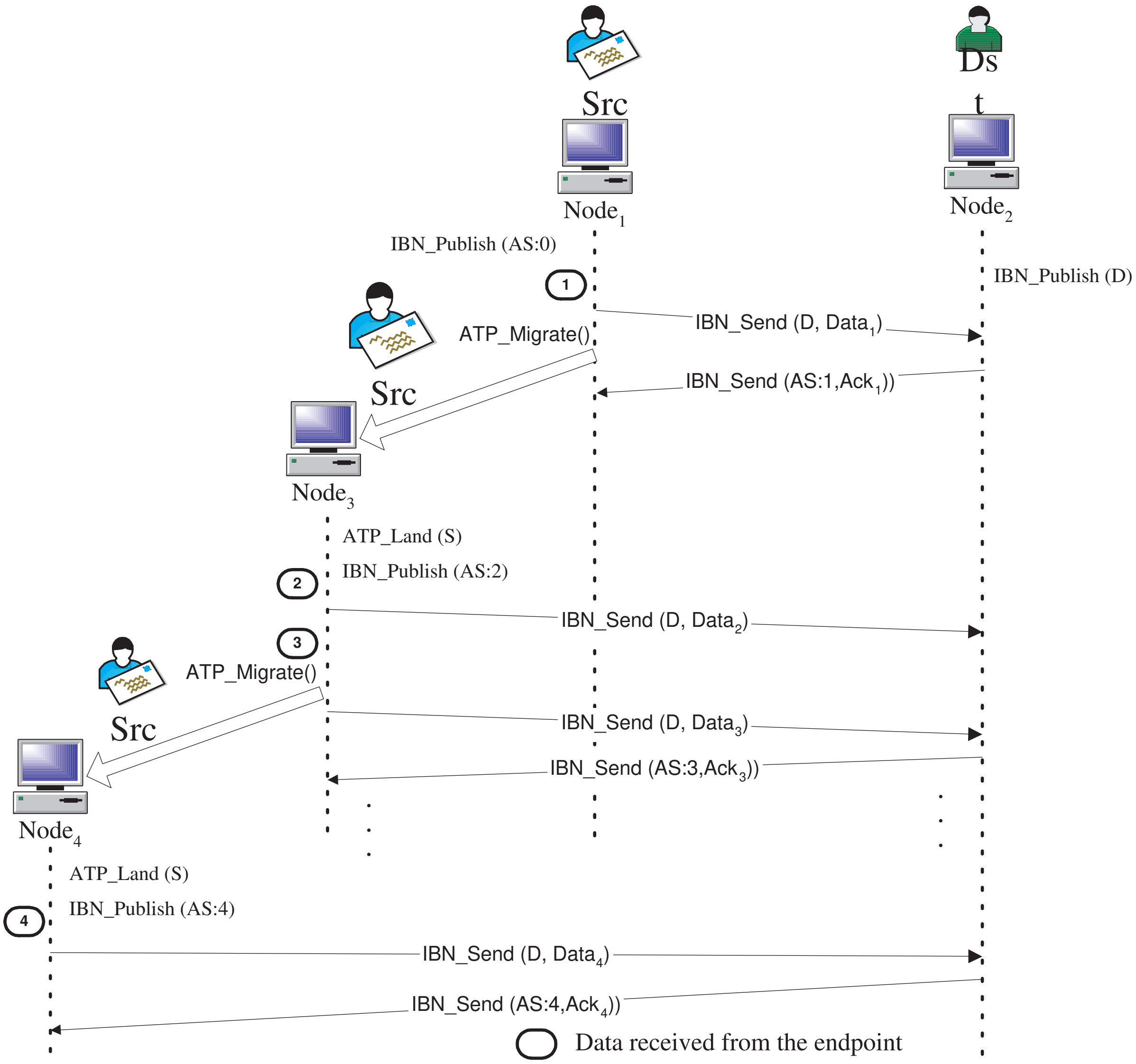

Figure 8. Multiple source migrations

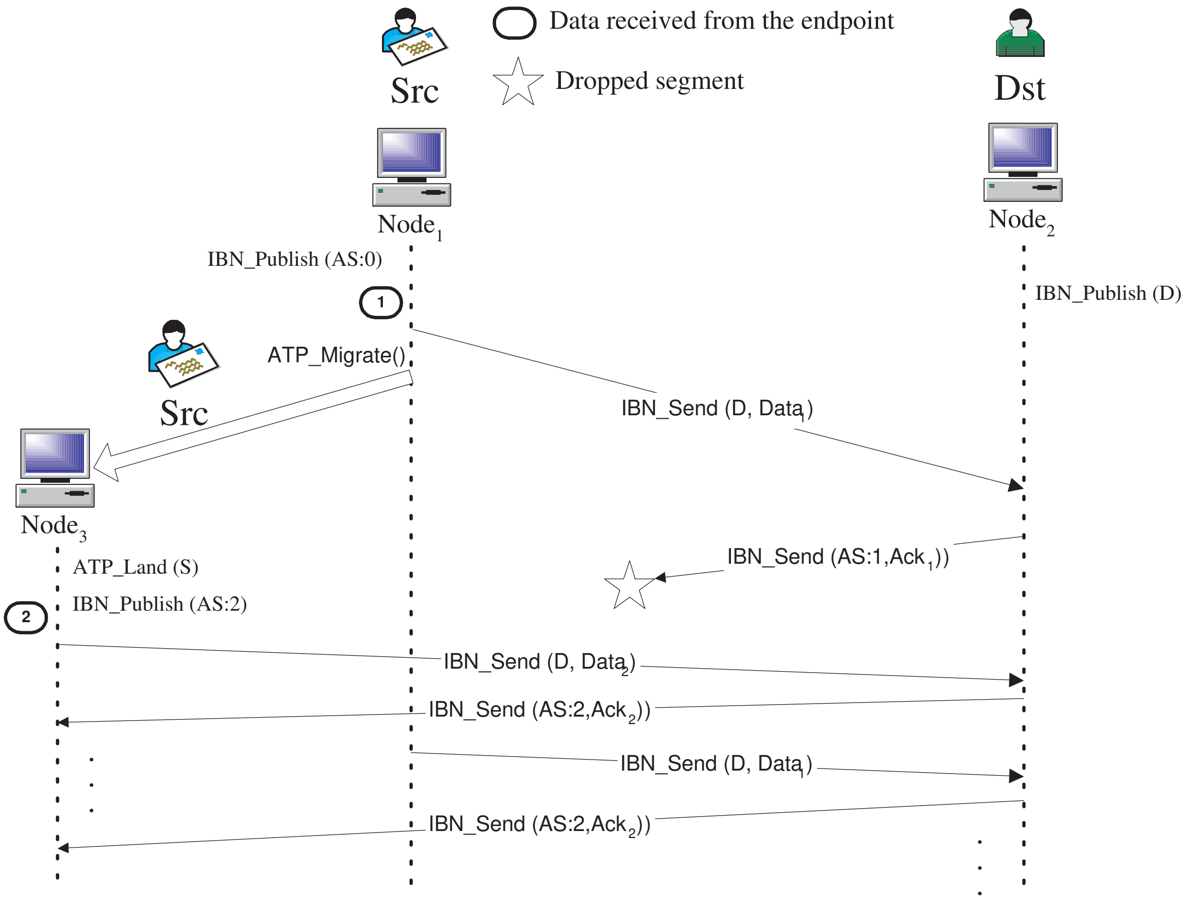

Figure 9. Inadequacy of cumulative acknowledgment in ATP

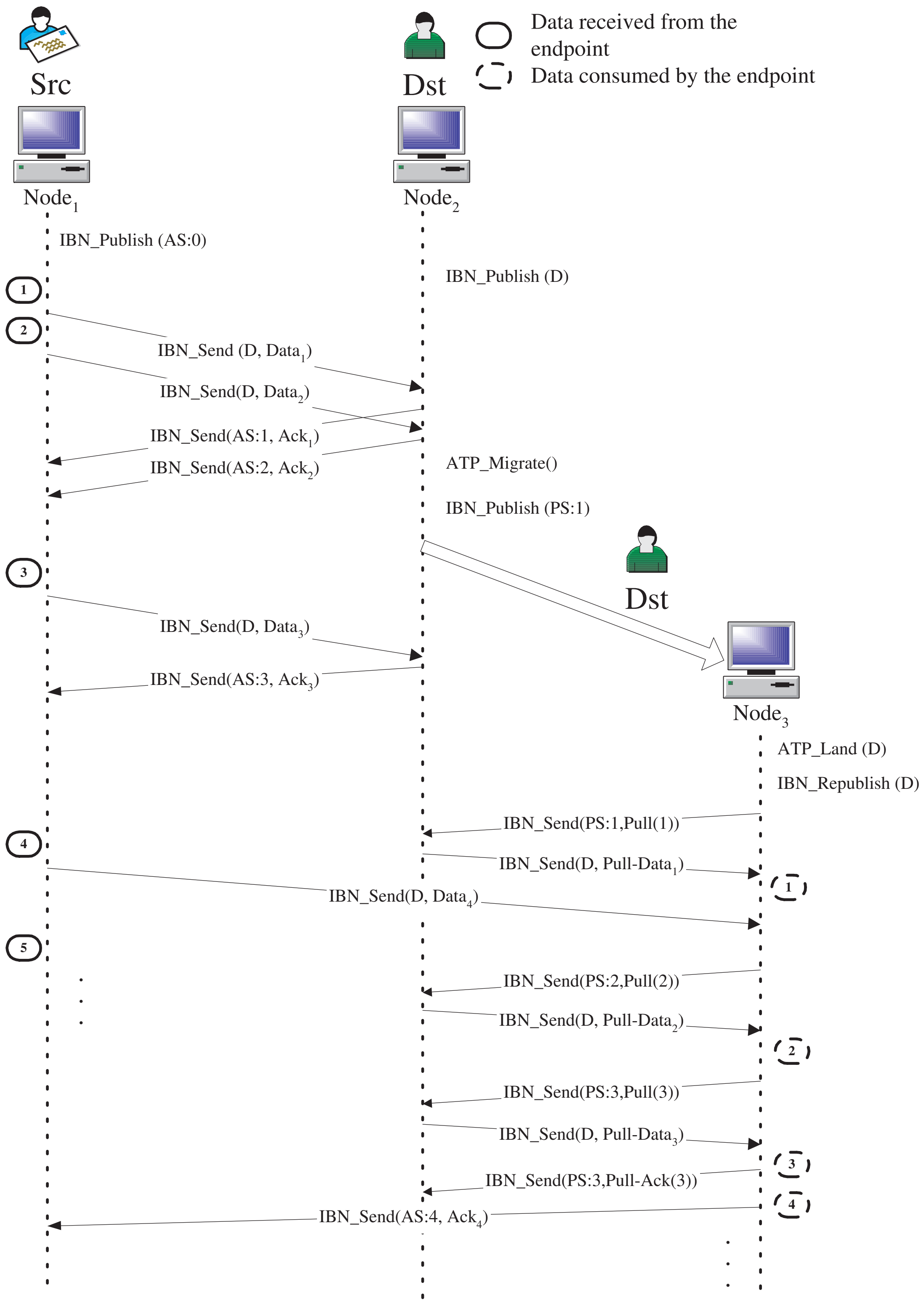

Figure 10. ATP destination migration

1. It stores the current status of all connections, that *Dst* is involved in, in the IBN. The status contains the last sequence number consumed by *Dst* on $Node_2$, last sequence number acknowledged[9], and other information needed to restore the connection on the new node.

2. It publishes a new content in the IBN identifying the agent responsible for maintaining the connection on $Node_2$. The content published by ATP_Agent_2 has an ID of *PS:j*, where j is the first sequence number not consumed by the application ($j = 1$ in Figure 10) attaching it to $Node_2$. This content identifier will be used by the ATP agent on $Node_3$ to pull the segments received at $Node_2$ while *Dst* is in migration. Note also that this agent will be in the passive mode since there is no benefit of actively sending segments to the destination when it is in migration.

During migration: Since the content D is still published in the IBN and is attached to $Node_2$, data segments sent from ATP_Agent_1 is routed to ATP_Agent_2 on *Node_2*. ATP_Agent_2 is responsible for buffering the new segments and acknowledging them.

Upon landing: When *Dst* lands on $Node_3$, it informs the ATP layer of its arrival using the ATP function call *ATP_LAND(D)*. In response, ATP spawns an agent to be responsible for this connection. Let's call that new agent ATP_Agent_3. ATP_Agent_3 starts its operation by performing two tasks:

1. Restoring the status of all connections, in which D is an endpoint, from the IBN;

2. Republishing the content D and attaching it to $Node_3$.

After landing: From now on, data segments sent by ATP_Agent_1 are routed to $Node_3$. Pulling of data is initiated by the application layer requesting data from the ATP layer[10]. When *Dst* asks for data by calling *ATP_RECEIVE()*, ATP_Agent_3 starts pulling data segments. A pull for segment with sequence number k is sent with destination address *PS:k*. The pull segment is routed, by the IBN instance-based feature, to the node that has the closest sequence number to the requested sequence number. For example, in Figure 10, a pull for segment 1 is routed to $Node_2$. In response to a pull segment, ATP_Agent_2 replies by sending the segment with the pulled sequence number. This reply should be in a special segment type, pull-reply. ATP_Agent_3, also, receives normal segments from ATP_Agent_1, which is still in the active mode.

5.6.1 Pulling mechanism

Pulling of segments is triggered by the application layer (*Dst*) when it asks for new data and the requested data in not in the ATP buffer. In order to avoid aggressive pulling, a certain amount of time should elapse between successive pulls. Pulling stops as soon as a data segment is received from an active source whose sequence number follows the sequence number of the last pulled segment. An ATP agent in the passive mode recognizes a pull for sequence number $k+1$ as an acknowledgment for sequence number k. If this ATP agent is not responsible for sequence number k, it needs to send an explicit acknowledgment for this sequence number with destination *PS:k*. This is needed because different agents may store different segments and two adjacent sequence numbers may reside on two different nodes[11]. The destination ATP needs to send an acknowledgment for a pulled data segment if this was the last pull segment (if it receives a data segment from an active source whose sequence number follows the last pulled data). The time sending the next pull (or the pull acknowledgment) can be either:

- After the application layer consumes the pulled segment. In this case, if *Dst* migrates before consuming the segment, ATP_Agent_3 can drop the segment. There will be only one version of the segment in the entire network.

[9] This is needed to optimize the cumulative acknowledgment process on the new node.

[10] This is to avoid pulling data that may not be consumed if the destination migrates again.

[11] Even if this acknowledgment is lost, the correctness of the protocol is not affected as each agent has a lifetime when it is spawned after which it is killed and its resources are released.

- As soon as the ATP layer receives the pulled data (before *Dst* consumes it). In this case, if *Dst* migrates before consuming the segment, ATP_Agent_3 has to maintain it in its buffer since there is no guarantee that ATP_Agent_2 would maintain it after receiving an acknowledgment for the last pull). There may be different copies of the same segment in the network since acknowledgments can be lost in the network.

5.6.2 Multiple Migrations

Figure 11 shows the operation of the ATP protocol when the destination makes multiple migrations from $Node_2$ to $Node_3$ and finally to $Node_4$. This is a direct extension to the case described in the previous section. It is important to notice two things here:

1. The ATP layer in $Node_3$, when it received the pull for segment 2, created a pull acknowledgment for segment 1. This is consistent with our pull policy, as described in Section 5.6.1, $Node_3$ is not responsible for segment 1.

2. The ATP layer in $Node_4$, when it received the pulled segment 2, created a pull acknowledgment for it. Again, this is consistent with our active mode policy as segment 3 was received using the active mode.

5.7 Reclaiming Network Resources

Since the ATP protocol uses the IBN to map content ID's to nodes and to store the status of the connection during migration, it becomes important to reclaim the network resources used by ATP when they are not needed. The leasing feature of the IBN means that if an endpoint of the connection or an agent is crashed (or does not appear for an extended period of time) all its resources will be deleted from the IBN as they will not be refreshed.

ATP also need to free resources that are used by agents in the nodes (e.g., the sending buffers). When an agent receives acknowledgement for all of its buffered segments, it can release the maintained buffer and die safely. Also, each agent has a lifetime assigned to it when it is created. If the lifetime of an agent elapses before receiving acknowledgment for all the segments in its buffer, the agent dies immediately.

6 Applications

The ATP protocol is essential to applications requiring reliable communication with endpoints migration. Examples include surveillance applications in sensor networks where a target is tracked using cameras on energy constrained sensors. To conserve energy, it is important to track the target using only one camera and to change the active camera as the target moves. It is also critical not to lose any data as the active camera changes. ATP becomes handy in such an environment. An ATP-aware application can be run on the node controlling the active camera. This application is responsible for sending the captured video frames to the command center. When the active camera changes, the ATP-aware application migrates to the new node. No data is lost during migration as the ATP provides a reliable connection between the source endpoint (sensor node) and the destination endpoint (command center). In such an

Another possible application is parallel download of large files. A destination endpoint representing the file can buffer a part of the file on one node and migrates to another node where it buffers another part. This buffering-migration cycles continues until the entire file is buffered at different nodes. The ATP agent at each node is responsible for transferring the part of the file in its buffer.

Future applications, e.g. the CarNet [] project can also benefit from the service provided by the ATP layer. (**Dr. Liviu, please complete this part and add any other relevant application**).

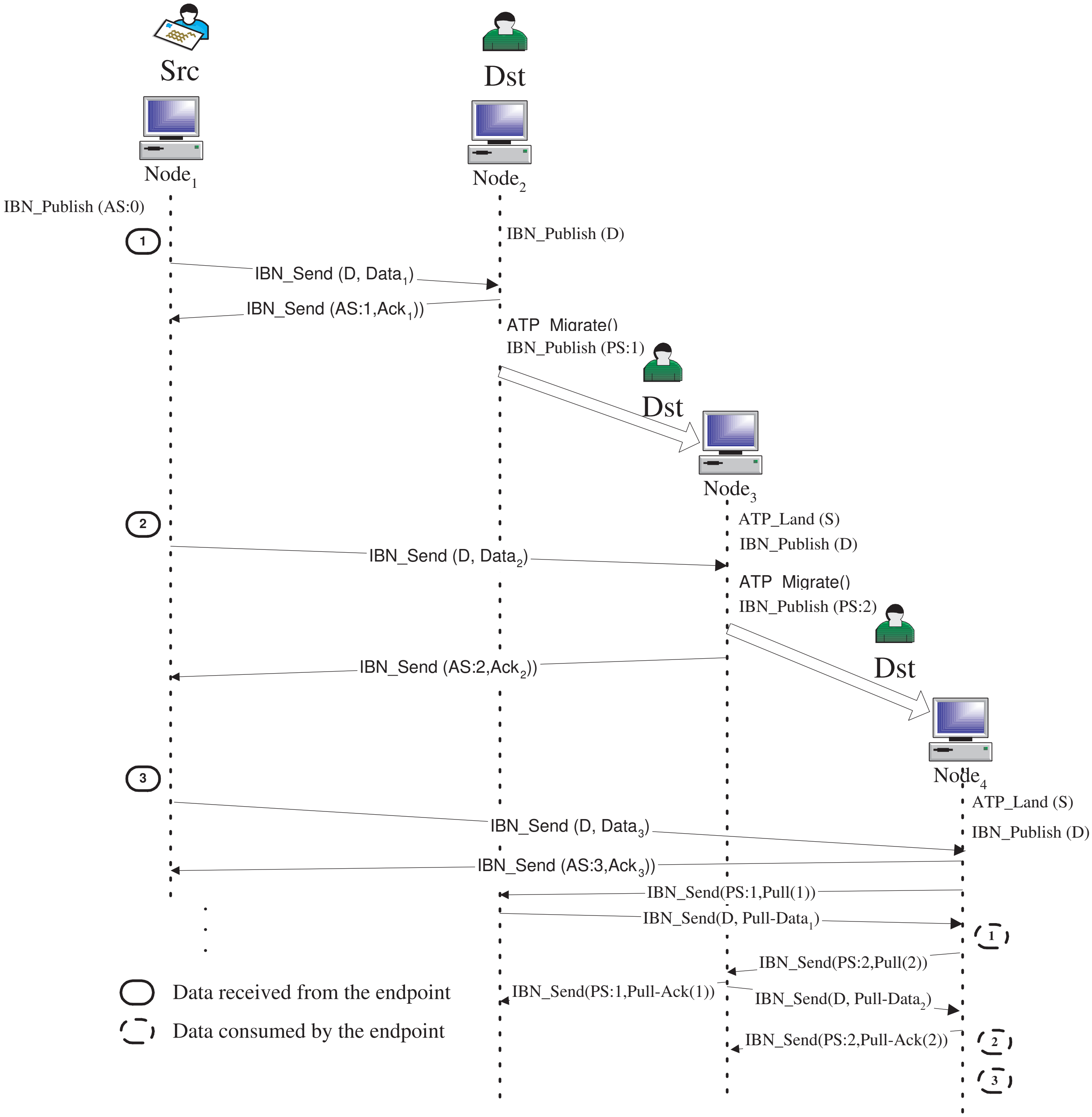

Figure 11. Multiple destination migrations

7 Related Work

Many systems over the year tried to address similar problems to the one we are trying to solve. In this section, we describe the related work to the IBN and to the ATP protocol.

Peer-to-peer lookup services (such as CAN [6], Chord [10], Pastry [7], and Tapestry [14]) provide a mechanism to map a key to some node in network specified by the lookup service, and allows the users to query for these keys. The IBN extends these functionality to allow mapping of a content to a *specific* node in the network and to route messages to this node. Moreover, the IBN has the unique feature of allowing its user to define different instances of the same content and route a message to the nearest instance of some particular instance.

The nearest work to the proposed IBN is the Internet Indirection Infrastructure (*I3*) [9] which builds over the Chord peer-to-peer system an indirection layer that allows a key to be mapped to a specific node in the network. Our IBN is unique in allowing different instances of the same content and in using instance-based routing.

There have been different solutions to the mobility problem over the years. Although the mobile IP protocol [5] provides a solution to the host mobility between different networks, a user is bound to a single host during the lifetime of a connection. The ATP provides the endpoints the flexibility of changing the hosts during the connection lifetime.

In [12, 8] they provide a mobility solution at the TCP level that allows the connection to remain open as the host moves between different networks. The solution in [12] allows the user to use a global 32-bit identifier for the connection, while the solution in [8] allows a connection endpoint to have different IP addresses and ports. However, both solutions does not allow the endpoint to change hosts.

The authors in [15] proposes a mobility infrastructure, based on *I3*, that offers a rendezvous-based communication abstraction. Instead of explicitly sending packets to a destination address, packets are sent to an identifier. A receiver who wishes to receive those packets uses the indirection infrastructure to associate its address with the identifier. Although their system is similar to ours in naming endpoints by identifiers and routing based on these identifiers, there are significant differences between them. Since their system relies on TCP, it inherits its problems regarding mobility. TCP requires that both source and destination coexist simultaneously for the communication to continue while our proposed protocol decouples each of them from the other. The *I3*-based system explicitly on the IP routing between the rendezvous point and the target while our proposed protocol does not assume a specific underlying infrastructure. Finally, the function of the switch points (*I3* servers) is to redirect data if it found matched triggers, while our proposed protocol requires intermediate nodes to collaborate with each other to maintain the connection even in the destination migration periods.

The *Mobile Tapestry* system [13] offers a similar system to the *I3*-based system. The main difference between a mobile system built on *I3* and *Mobile Tapestry* is that, while the former provides a single level of indirection as a rendezvous point and for redirecting packets, Mobile Tapestry provides multiple points of indirection. This system has the same shortcomings as the *I3*-based system.

The system in [11] provides a seamless service platform in which endpoints can migrate between different nodes. The system is based on seamless-proxies that forms a network between themselves and works as a middleware between the application and the transport layer. Our system differs from this system in that their system is IP-based while ATP uses the IBN to decouple itself from the underlying network. The endpoint cannot have multiple migrations standing thus limiting the flexibility of endpoint migration. For each possible application their system requires an application-specific module to handle endpoint migration. Moreover, endpoint migration is handled by closing existing connection on the old node and creating new connections on the new node and trying to maintain the state of the old connection.

8 Discussion

In this section, we investigate some aspects of the design of the ATP protocol.

8.1 Fault Tolerance

In a dynamic network environment, maintaining a reliable communication is a great challenge. We have two types of faults: node failure and link failure.

ATP can rely on the IBN route discovery service to alleviate the link-failure problem. The node-failure problem manifests itself in three situations. First, the failed node could be responsible for a set of mapped contents. Second, the node could be the host node for an agent responsible for some segments. Third, that node could be the destination endpoint for a stream of network segments.

To solve the first subproblem, ATP can rely on the underlying IBN replication mechanisms to transparently provide backup nodes for each mapped content. To solve the second subproblem, the ATP should rely on its own replication mechanism. This issue is still open. One possible solution is to make the ATP state (including buffers) stored in the IBN network and depend on the IBN replication mechanism. The third subproblem is a catastrophic case as it may lead to the destruction of the application state. If the node failure affects only the networking stack and not the application layer, then we can apply the solution of subproblem 2 to this subproblem. The IBN may need to do a remapping of all content ID's that were published at the failed node to a new default node so that segments sent during the recovery period are buffered and the source can continue its operations without interruption of service.

8.2 Active/Passive Policy

Any sending ATP agent can take the decision to dynamically switch between the active and passive modes according to a certain policy. One possible policy is to consider the current state of network resources (e.g. buffer length, CPU load, available bandwidth, or remaining energy) locally or even globally (which requires the neighbor nodes to cooperate is someway). The decision could also be left to the application and can be negotiated during the connection establishment. The challenge is to determine who, when, and how to take the switching decision. This needs to be tested experimentally.

It is natural for the source agents to be in the active mode, while it is more appropriate for the destination agents that buffer data from the source to be in the passive mode waiting for data to be pulled from them as discussed before.

Note that when we allow destination agents to be in the active mode, there may be more than an active source responsible for the same data segments. This may lead to deleting a data segment from the network without being consumed by the destination. This is because acknowledgments from the past may be delayed for any arbitrary period in the network and arrive at a wrong destination. Figure 12 shows an example for such a scenario. We suggest to different solutions to this problem:

1. send the acknowledgment to the source that generated the segment. This solution has the disadvantage of making the destination keep track of the source of each segment, reducing the decoupling between the source and the destination.
2. use epoch numbers protect from aging acknowledgments.

8.3 Security

Allowing ATP to act on behalf of the source or the destination opens many interesting security issues. We plan to investigate these issues thoroughly after setting up the initial design.

Security is an important issue is distributed systems like ATP. The security problem is three-fold: privacy, authenticity and trust. One solution to privacy is to apply public-key cryptography, such as the Diffe-Hellman key-exchange technology [3], to communications between endpoints. Moreover, the platform's security is no weaker than that of any other distributed platform.

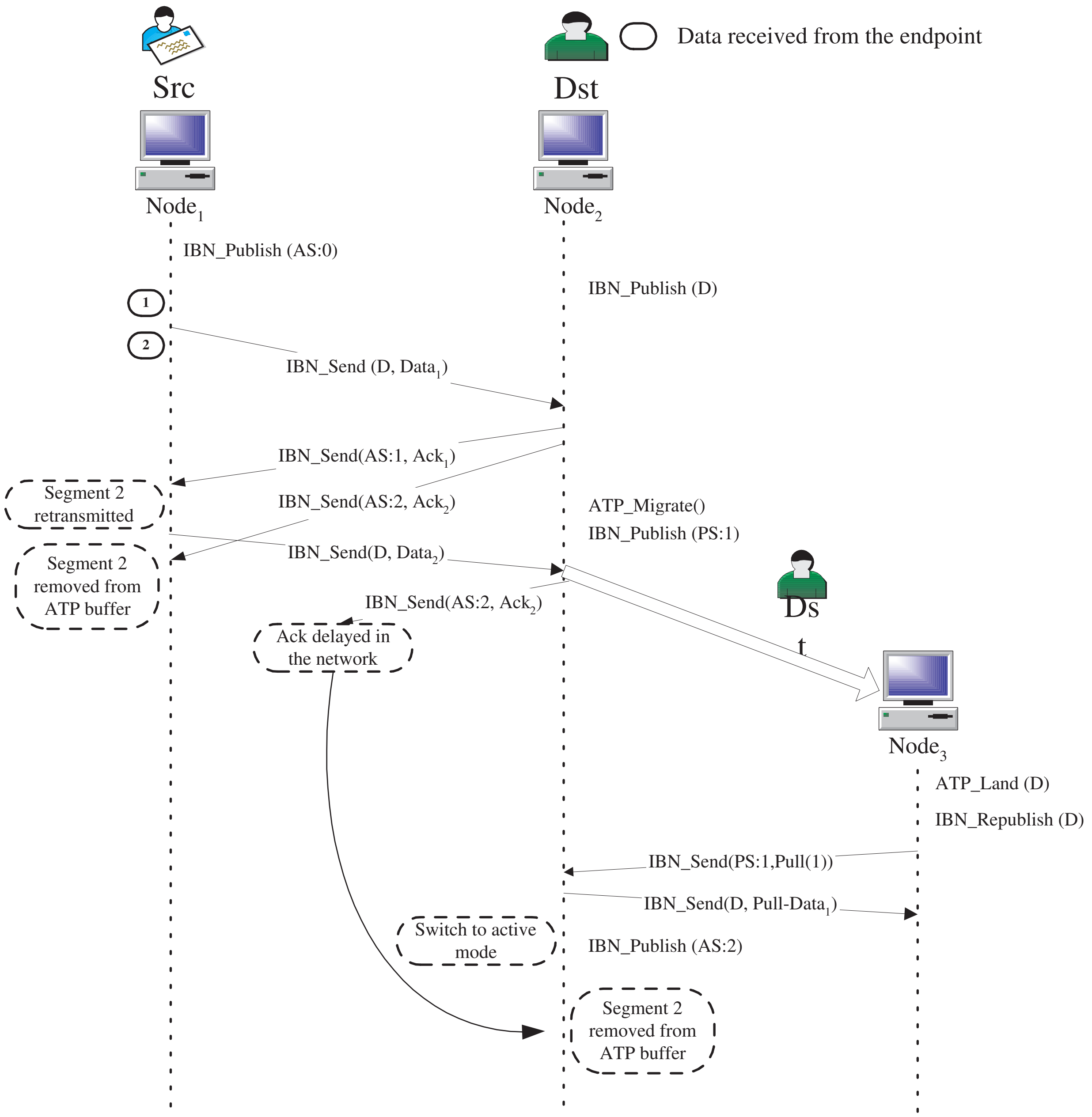

Figure 12. A delayed acknowledgment in the network can lead deleting a data segment from the network without being consumed by the endpoint

Authentication is difficult in a dynamic network environment because the presence of reliable and always available certificate authorities (CAs) cannot be assumed. Although there is a tradeoff between the decentralization of a CA [2] and the CA's effectiveness, this does place a limit on the level of security possible with our system. However, there are some easy solutions to the authentication of nodes on the user side; e.g., by inserting secure IC cards or communicating directly with each other through IrDA ports, nodes can be identified with certainty and keys can then be safely exchanged.

There have been some initial proposals for distributed trust systems [4]. These solutions can be directly incorporated into our system providing a solution to the trust issue.

8.4 End-to-End Semantics

In ATP, agents act on behalf of both source and destination endpoints to guarantee the delivery of the segments including control segments, e.g. close connection request. In traditional reliable transport protocols, e.g. TCP, the source has to wait until it receives an acknowledgment for the close connection request after successfully transmitting all the data segments. In addition to coping with this traditional end-to-end semantic, ATP could shift the burden of waiting for such acknowledgment from the source to the ATP agents. Eliminating such waiting time, which is signified by the migration of the endpoints, allows the source to terminate earlier while the destination is still in the middle of the connection. Although this feature violates the traditional end-to-end semantics of the reliable connection, it could be very useful for certain application types.

9 Conclusions

In this report, we presented the design of the *Autonomous Transport Protocol*. ATP provides a reliable stream between two endpoints regardless of their physical location in the network. We also introduced the *Instance-Based Networking* as an extension to the current *peer-to-peer* lookup services.

In an instance-based network, the network provides the flexibility of having different instances of the same content. It is up to the user of the IBN network to define the relation between these instances. An IBN provides its user with the ability to map a content to a particular node. Application endpoints, defined by contents, can send messages to other content-identified endpoints. Routing in the IBN is instance-based; the IBN can route a message to a specific content instance or to the nearest instance, if no exact match is found for the destination content instance. The semantics of different instances and the relation between them are assigned by the application using the IBN. Moreover, the IBN replicates the stored contents in order to provide fault tolerance and IBN nodes along the query path can cache a content to provide fast answers to future queries.

The ATP protocol allows the endpoints to migrate between different hosts. At each node that an endpoint has data, the ATP spawns an agent that becomes responsible for the connection on this node. Each instance is treated as an instance of the same content endpoint. ATP uses the underlying IBN to route the data and acknowledgment to the correct agent.

Data is transfered by a combination of *active* and *passive* operations, where the ATP layer of a node can decide whether to actively push the data to the destination or to passively wait for the destination endpoint to pull the data. The decision to whether to use the active or passive modes can be taken by a local policy on the node running the ATP protocol. This decision may be based on the current state of network resources (e.g. buffer length, CPU load, available bandwidth, or remaining energy). The decision could also be left to the application and can be negotiated during the connection establishment.

The ATP protocol is essential to applications requiring reliable communication with endpoints migration. Current applications can be made ATP-aware with minor modification to their code. We believe that the ATP protocol is a corner stone for ubiquitous computing to become a reality.

References

[1] Mobile Ad Hoc Networking (MANET): Routing Protocol Performance Issues and Evaluation Considerations. *RFC 2501*, January 1999.

[2] R. Canetti, S. Halevi, and A. Herzberg. Maintaining authenticated communication in the presence of break-ins. *Journal of Cryptology*, 13:61–105, 2000.

[3] W. Diffe and M. Hellman. New Direction in Cryptography. *IEEE Transaction on Information Theory*, IT-22(6):644–654, 1976.

[4] S. Lee, R. Sherwood, and B. Bhattacharje. Cooperative Peer Groups in NICE. In *Proceedings of IEEE Infocom 2003*, 2003.

[5] C. Perkins. IP Mobility Support. *RFC 2002*, October 1996.

[6] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A Scalable Content-Addressable Network. In *Proceedings of ACM SIGCOMM 2001*, 2001.

[7] A. Rowstron and P. Druschel. Pastry: Scalable, distributed Object Location and Routing for Large-Scale Peer-to-Peer Systems. In *Proceedings of IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, pages 329–350, Heidelberg, Germany, November 2001.

[8] A. C. Snoeren and H. Balakrishnan. TCP Connection Migration. *Internet Draft*, November 2000.

[9] I. Stoica, D. Adkins, S. Zhaung, S. Shenker, and S. Surana. Internet Indirection Infrastructure. In *Proceedings of ACM SIGCOMM 2002*, pages 73–86, Pittsburgh, PA, August 2002.

[10] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications. In *Proceedings of ACM SIGCOMM 2001*, San Diego, September 2001.

[11] K. Takasugi, M. Nakamura, S. Tanaka, and M. Kubota. Seamless Service Platform for Following a User s Movement in a Dynamic Network Environment. In *Proceedings of the First IEEE International Conference on Pervasive Computing and Communications (PerCom 03)*, March 2003.

[12] B. Zhang, B. Zhang, and I. Wu. ETCP: Extended TCP for Mobile IP Support. *Internet Draft*, 1998.

[13] B. Y. Zhao, A. D. Joseph, and J. D. Kubiatowicz. Supporting rapid mobility via locality in an overlay network. Technical Report UCB/CSD-02-1216, UC Berkeley, November 2002.

[14] B. Y. Zhao, J. D. Kubiatowicz, and A. D. Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Technical Report UCB/CSD-01-1141, UC Berkeley, April 2001.

[15] S. Zhuang, K. Lai, I. Stoica, R. Katz, and S. Shenker. Host Mobility Using an Internet Indirection Infrastructure. In *Proceedings of MobiSys 2003*, 2003.

Appendix A: ATP-Application Interface

The ATP interface to the application layer adds two function calls to the TCP socket-programming interface ("*Migrate*" and "*Land*") besides minor changes to the other calls. Here, we briefly review those calls. For the following functions, the "Address" structure has two fields: *content* and *port*, both are given and arbitrarily selected by the application. The *content* is equivalent to the IP address in the current IP networks and the port is equivalent to the current port.

- *SOCKET(Type)*: Creates a new socket and returns the socket file descriptor. The argument *Type* specifies the socket type; which could be one of two: streams, and datagrams. In our initial design, we are concerned only with streams.

- *BIND(Socket, Addr)*: Binds *Socket* to the port specified in the address structure *Addr*.

- *CLOSE(Socket)*: Shutdowns the communication from the caller side specified by *Socket*.

- *CONNECT(Socket, Dst_Addr)*: Connects the *Socket* to the remote socket bound to the address *Dst_Addr*.

- *SEND(Socket, Msg)*: Sends the message *Msg* through the connection specified by *Socket*.

- *RECEIVE(Socket, Buffer, Length)*: Requests ATP to deliver data of length at most *Length* sent to *Socket*.

- *LISTEN(Socket)*: Causes the incoming ATP connection request to socket *Socket* to be handled and then queued for acceptance by the program.
- *ACCEPT(Socket, Src_Addr)*: Dequeues the next connection on the queue for *Socket*.
- *MIGRATE()*: Informs ATP that the application decided to migrate requiring ATP to maintain all associated socket connections.
- *LAND(Content)*: Restores all sockets that were bound to *Content* name before migration. This function is called when the migrating application land on a new physical node.

Appendix B: IBN Interface with Upper Layers

An IBN should provide the following API's[12]:

- *PUBLISH(ContentID, ContentData, NodeID)*: Publishes (or announces) a content to the IBN network. *ContentID* is the unique identifier of the content to be published. *ContentData* is the data, corresponding to that content, which will be stored in the network. *NodeID* is the identifier of a particular node in the IBN to hold the content. NodeID can be omitted; in that case, the IBN is free to store the content in any node (this is the function provided by the current peer-to-peer lookup services).
- *REPUBLISH(ContentID, NodeID)*: Republishes the content whose *ID* is *ContentID* by changing its location in the IBN to the node whose *ID* is *NodeID*.
- *REMOVE(ContentID)*: Removes a content that the user has previously published in the IBN.
- *SEND(Message, ContentID)*: Informs the IBN to send a segment *Message* to a content *ContentID* published to the IBN.
- *RETRIEVE(ContentID)*: Retrieves a copy of the data corresponding to a previously published content whose ID is ContentID.
- *DELIVER (SrcContentID, DstContentID, Message)*: Acts as an up call from the IBN to the upper layer to notify the user of the arrival of a message *Message* destined to it. *SrcContentID* and *DstCOntentID* are the *ID* of source and destination contents respectively.

Appendix C: IBN Implementation

We propose an implementation that extends the current peer-to-peer lookup services. When the user asks the IBN to publish the content instance $(X : i_1, ..., i_n)$ on a certain node A, the IBN uses the underlying peer-to-peer lookup service to find the node B where the content X should be mapped to. The IBN layer uses the node B to store a mapping between the instance $(X : i_1, ..., i_n)$ and the node A in a data structure $\mathbb{D}$. Note that node B will be responsible for all instances of the same content X.

If an application wants to send a message to an instance $(X : j_1, ..., j_n)$, the IBN routes the message using the underlying peer-to-peer network to the node B. The IBN layer on node B checks the data structure $\mathbb{D}$ to find the nearest instance to the destination instance identifier $(j_1, ..., j_n)$ and the node C where this instance is stored (using the routing algorithm is Section 4. The IBN finally forwards the message to node C. Other API functions described in Appendix B can be obtained in a way similar to the above two operations.

The proposed implementation has the following features:

[12] We use *ContentID* here to refer to the full name of the content instance

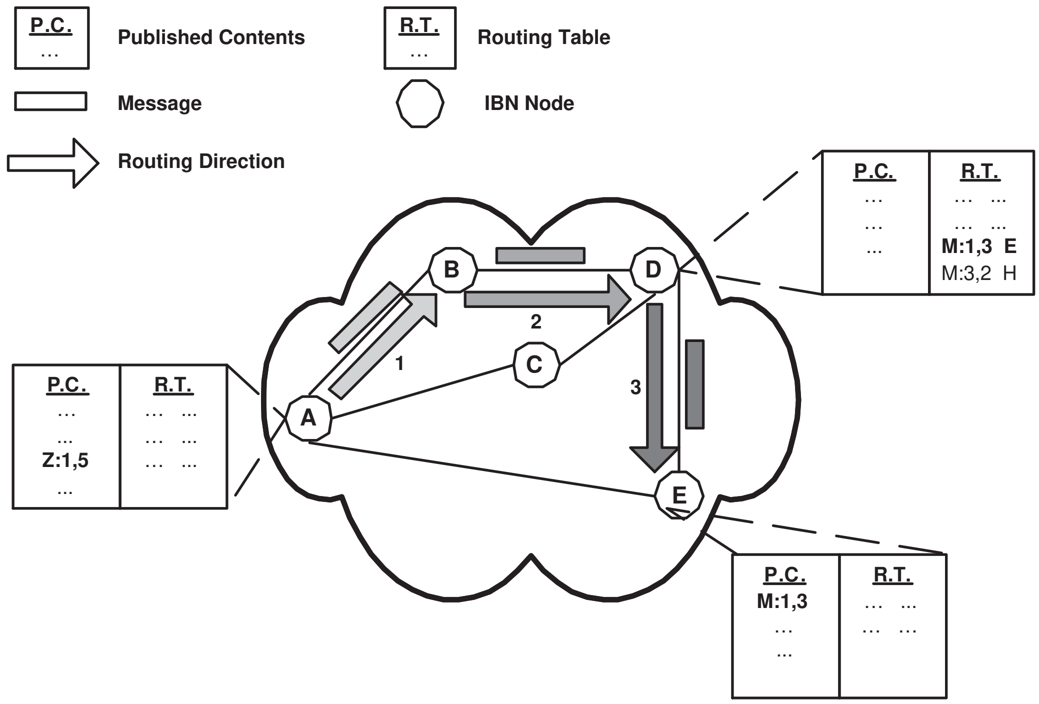

Figure 13. IBN routing: a message from $Z:1,5$ destined to $M:2,3$ is routed to $M:1,3$

- A single node is responsible for all instances of a particular content. This allows for efficient algorithms for finding the nearest instance to a given instance while routing a message.
- Different contents are mapped to different nodes based on the algorithm used by the underlying peer-to-peer lookup service. In general, peer-to-peer lookup services assigns keys (content IDs) to nodes uniformly so that no node becomes overloaded and there are techniques to differ the number of keys on each node based on its actual load.
- The proposed implementation is independent of the underlying peer-to-peer lookup services. The implementer can choose from different peer-to-peer lookup services, for example, based on the efficiency of the lookup operation (some peer-to-peer lookup services, such as Pastry [7], takes network locality into their routing algorithm.)

Figure 13 shows an example for the routing in the IBN. A message from $Z:1,5$ destined to $M:2,3$ is routed to $M:1,3$. The message is first routed to D (the node responsible for the mapping of the instances of M) using the underlying peer-to-peer routing mechanism (steps 1 and 2). When the messages reaches node D, it checks its forwarding table and redirects it to the node responsible for the closest instance (Node E).

Appendix D: Range Acknowledgment

In this appendix, we describe a technique for range acknowledgment that can enhance the performance of the ATP protocol. Performance evaluation experiments need to be conducted to assess the benefit of this technique.

D.1 Overview

A range acknowledgment acknowledges a range of sequence numbers. It is identified by two sequence numbers, the starting sequence number of the acknowledged range and the ending sequence number of the acknowledged range.

Let *ACK(i:j)* be a range acknowledgment from i to j. *ACK(i:j)* is sent with a destination address *AS:j*. If *ATP_Agent*$_2$ receives all data segments in sequence, it sends back *ACK(1:k)*, where 1 is the first sequence number received and k is the last sequence number received. If *ATP_Agent*$_2$ receives segments out of sequence, we can have different acknowledgment policies:

1. Sending multiple acknowledgment for the multiple ranges. For example, if *ATP_Agent*$_2$ has sequence number ranges i_1 to i_2, i_3 to i_4, i_5 to i_6, it sends *ACK(*$i_1 : i_2$*)*, *ACK(*$i_3 : i_4$*)*, and *ACK(*$i_5 : i_6$*)*. Each of these acknowledgments is sent to a different destination address depending on its range: *ACK(*$i_1 : i_2$*)* is sent to *AS:*i_2, *ACK(*$i_3 : i_4$*)* is sent to *AS:*i_4, while *ACK(*$i_5 : i_6$*)* is sent to *AS:*i_6.

2. Sending acknowledgment for the first range only with a possibility of retransmission of the segments we already received. This is equivalent to the cumulative acknowledgment mechanism.

3. Sending acknowledgment for each segment we receive. In this case we do not need the range (or cumulative) acknowledgment mechanism.

4. Sending acknowledgment for the range that the last received segment falls in.

5. Sending two range acknowledgments. One for the first range (corresponding to cumulative acknowledgment) and the second is for the range in which the last segment was received. The first range acknowledgment is sent to trigger the fast retransmission mechanism (imported from TCP).

These different policies do not affect the correctness of the ATP protocol. In all cases, the source will time out, retransmit the segment, and dies if it does not receive any acknowledgment for a certain period of time. The main factors affecting the choice of the acknowledgment policy are:

- avoiding retransmission of already received segments.

- avoiding sending unnecessary acknowledgments.

Sending of acknowledgments is triggered by either:

- expiration of a timer (or alternatively, reception of a certain amount of data): in this case, the ATP protocol sends a range acknowledgment for the first consecutive range that it is responsible for (similar to cumulative acknowledgment).

- reception of a segment out of order: in this case, the ATP protocol can choose one of the above policies. For the rest of this appendix, we will stick to option 4 (acknowledging the range that the last received segment falls in).

- reception of a duplicate segment: in this case, the ATP protocol sends an acknowledgment for this duplicate segment.

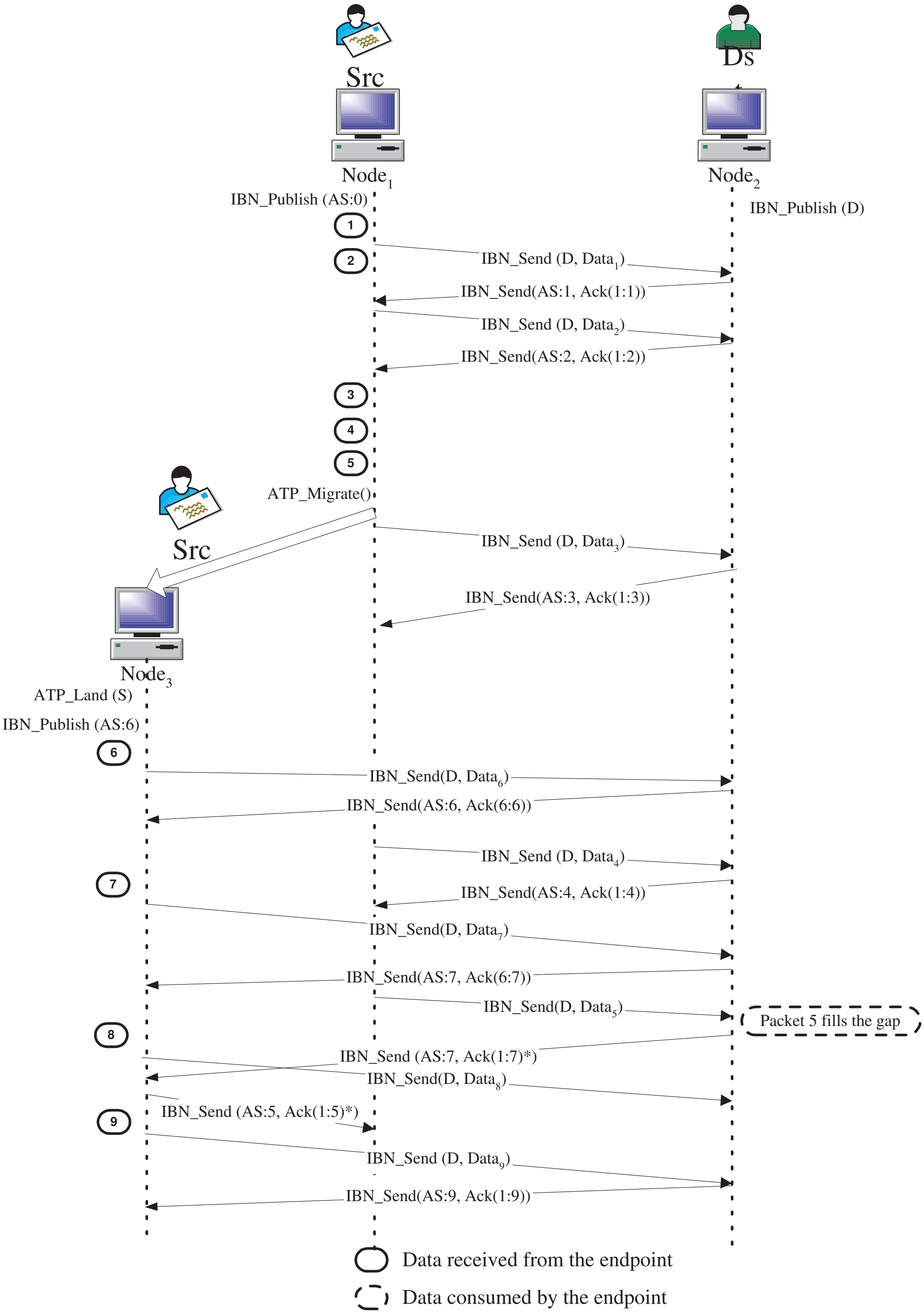

Figure 14. ATP range acknowledgment

D.2 Forward Acknowledgment Flag

Since we may have different agents acting on behalf of the source, a range acknowledgment may cover different nodes. However, a single acknowledgment will be routed to a single node. Therefore, we need a mechanism by which a single range acknowledgment sent by the destination could reach all the nodes covered by it. We propose the *Forward Acknowledgment Flag* (*FAF*) as a solution to this problem. Figure 14 shows an example of the technique (*FAF* indicated by * in the acknowledgment). When the destination receives a segment with a sequence number that fills the gap between two ranges (segment 5 in the figure), it sets the *FAF* in the corresponding range acknowledgment. When a sending agent receives an acknowledgment with the *FAF* set, it checks whether a new range acknowledgment should be generated with a new range that is not covered by the current agent. If a new acknowledgment is generated, its *FAF* is set and the process is repeated.

D.3 Multiple Destination Migration

In this section, we discuss the multiple destination migration case when we have the range acknowledgment mechanism. The problem of destination migration becomes more difficult when *Dst* migrates another time before finishing the pulling process for the first migration. In this case, *Dst* may leave behind some segments in the receiving buffer of ATP_Agent_2 and some other segments in ATP_Agent_3. Because of the time of instability in the IBN, after republishing an old content or publishing a new content, the segments buffered in both agents may not be in sequence.

For example, consider the scenario depicted in Figure 15. Let the ATP agents responsible for this connection on $Node_2$, $Node_3$, and $Node_4$ be ATP_Agent_2, ATP_Agent_3, and ATP_Agent_4 respectively. Because of the loss of segment 3 (step A), it does not reach ATP_Agent_2. When ATP_Agent_1 retransmits segment 3 (step B), ATP_Agent_3 has republished the content *D* and attached it to $Node_3$ (step C), therefore segment 3 is routed to ATP_Agent_3 instead of ATP_Agent_2[13]. On the other hand, segment 4 reaches ATP_Agent_2 successfully and is acknowledged (step D), therefore, it is not retransmitted by ATP_Agent_1. ATP_Agent_3 starts pulling segments from ATP_Agent_2. It pulls segments 1 and 2 successfully (steps E and F). When *Dst* migrates from $Node_3$ to $Node_4$ (step G) after consuming segment 1 but before consuming segment 2. Therefore, ATP_Agent_3 acknowledges segment 1 and drops segment 2. To this point, ATP_Agent_2 is buffering segments 1, 2, and 4; and ATP_Agent_{22} is buffering segments 3 and 5. Upon the second migration of *Dst*, ATP_Agent_{22} publishes *PS:3* (step H).

When ATP_Agent_4 restores the connection status, it knows that the next sequence number to be consumed by *Dst* is 2. Therefore, upon *Dst*'s request of new data, ATP_Agent_4 pulls segment 2 (step I). This pull segment reaches ATP_Agent_2, which responds by sending a pull-reply with the required segment (step J). Similarly, segment 3 is pulled. The difference is that the pull segment in this case reaches ATP_Agent_3 instead of ATP_Agent_2 since ATP_Agent_3 published *PS:3* (step K). The problem occurs when ATP_Agent_4 pulls segment 4. The pull segment for segment 4 has *PS:4* as its destination address, and hence reaches ATP_Agent_3 (step L) , which does not have it! Actually, a pull segment for segment 4 will never reach ATP_Agent_2.

As a solution to this problem, we propose that ATP_Agent_3 when receiving a pull for segment 4, it removes the content *PS:3* from the IBN and publishes, instead, content *PS:5*, where 5 is the next sequence number it has (step M). ATP_Agent_3 should also forward the pull segment. Doing so, ATP_Agent_4 will be able to pull segments 4 and 5 correctly. Publishing content *PS:5* instead of *PS:3* is not performed before receiving pull-acknowledgment for segment 3.

Another problem occurs if segment 4 arrives at ATP_Agent_3 after publishing *PS:5* instead of *PS:3*? That can happen since segment 4 might have been in its route to ATP_Agent_3 when the latter received the pull for it. Because of the high dynamics in our environment, we have to take precautions for such intricate situations. In this situation,

[13]Note that this cannot happen in the case of cumulative acknowledgment as segments will always be in sequence

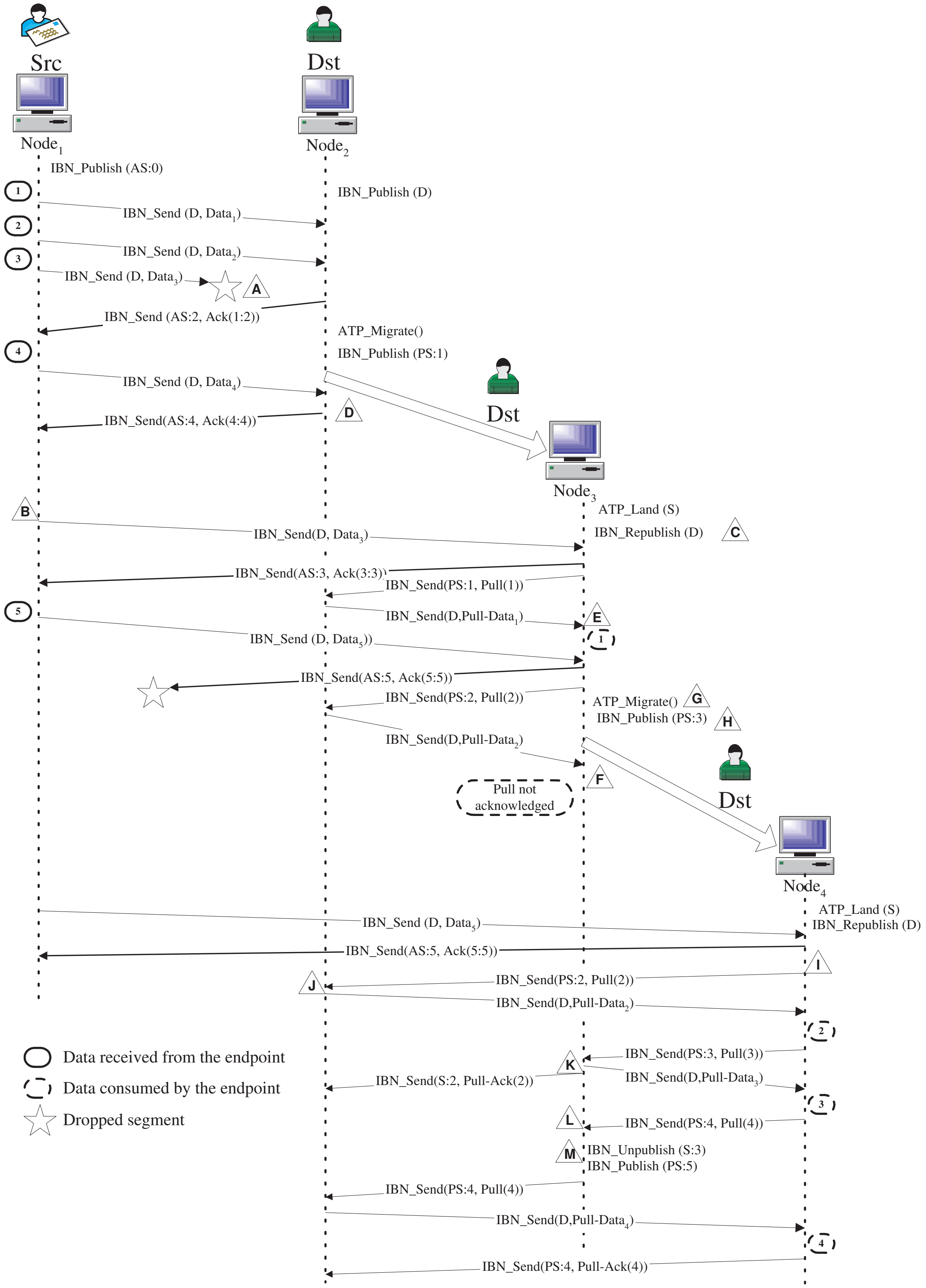

Figure 15. Multiple migration with range acknowledgment

we propose that ATP_Agent_3 has to remove the content *PS:5* and publish *PS:4*, instead. If another agent is already responsible for segment 4 (i.e if the IBN rejects the publishing operation) then segment 4 can be safely ignored.

To summarize, an agent receiving a pull segment for a sequence number not in its buffer should unpublish its current ID and republishes an ID containing the next sequence number, in the agent buffer, after the one in the pull segment. It should also retransmit the received pull segment. If an agent receives a segment with sequence number before the sequence number in its current published ID, it should try to change its published ID to contain the sequence number in the received segment. If the publishing operation fails, the agent can safely drop the received segment.

D.4 Closing Remarks

The range acknowledgment mechanism has the advantage of reducing the retransmission of already received segments. However, it may produce unnecessary acknowledgments. Another problem is that when a range acknowledgment with the *FAF* is lost, this acknowledgment will not be reproduced again (as its range has already been filled). Moreover, handling the case of multiple destination migrations may involve publishing and unpublishing operations. We believe that range acknowledgment is still an effective mechanism to enhance performance as acknowledgment are smaller in size than data segments and the problematic cases occurs with low probability. However, we need to verify that by experimentation.