Computer Security, PART III: Conference Protocols

Timo Karvi

11.2010

Introduction to Conference Protocols I

Conference protocols establish keys for groups of principals. There is a great variety of different practical requirements that may be appropriate in different applications.
The following four factors influence the requirements:

- Application type: A fundamental feature is how many of the parties must be able to send information. Corporate teleconference: all wish to send. Satellite broadcast: only one sender.
- Group size and dynamics: If groups are small, all can take part in interactive key establishment. If groups are very large, this is impractical. If members can be added or deleted, it must happen without too heavy a burden.
- Scalability: The efficiency of protocols may vary as the size of the group of users changes.
- Trust model: It is important to define which principals are trusted to generate and authenticate keys.

Introduction to Conference Protocols II

More concepts:

- Forward secrecy (for multiparty protocols): An adversary who knows a set of groups keys cannot derive any subsequent group key.
- Backward secrecy: An adversary who knows a set of groups keys cannot find any earlier group key.
- Key independence: An adversary who knows a set of groups keys cannot find any other group key.

GDH-Protocols I

Steiner, Tsudik and Waidner proposed three protocols which generalize Diffie-Hellman key agreement. Consider the first version GDH1:

As in ordinary Diffie-Hellman, we have a public group Z_p and its public generator g. There are m users $U_1, U_2, \cdots, U_m$. The protocol assumes that every user knows its neighbours, i.e. U_i knows U_{i-1} and U_{i+1}. Every user U_i chooses its secret number r_i randomly.

Phase 1	$U_{i-1} \longrightarrow U_i$:	$g^{r_1}, g^{r_1 r_2}, \cdots, g^{r_1 r_2, \cdots, r_{i-1}}$
	$U_i \longrightarrow U_{i+1}$:	$g^{r_1}, g^{r_1 r_2}, \cdots, g^{r_1 r_2, \cdots, r_i}$
Phase 2	$U_{i+1} \longrightarrow U_i$:	$h_{i+1}, h_{i+1}^{r_1}, h_{i+1}^{r_1 r_2}, \cdots, h_{i+1}^{r_1 r_2 \cdots r_{i-1}}$, where $h_{i+1} = g^{r_{i+1} r_{i+2} \cdots r_m}$
	$U_i \longrightarrow U_{i-1}$:	$h_i, h_i^{r_1}, h_i^{r_1 r_2}, \cdots, h_i^{r_1 r_2 \cdots r_{i-2}}$, where $h_i = g^{r_i r_{i+1} \cdots r_m}$

After a user U_i has received the message in the second phase, it can compute the session key as $K = g^{r_1 r_2 \cdots r_m}$ by raising the last part of the message to power r_i.

GDH2

Version 2 is more efficient.

Phase 1	$U_{i-1} \longrightarrow U_i$:	$p_{i-1}, p_{i-1}^{r_1^{-1}}, p_{i-1}^{r_2^{-1}}, \cdots, p_{i-1}^{r_{i-1}^{-1}}$ where $p_{i-1} = g^{r_1 r_2 \cdots r_{i-1}}$
	$U_i \longrightarrow U_{i+1}$:	$p_i, p_i^{r_1^{-1}}, p_i^{r_2^{-1}}, \cdots, p_i^{r_i^{-1}}$ where $p_i = g^{r_1 r_2 \cdots r_i}$
Phase 2	U_m broadcasts:	$K^{r_1^{-1}}, K^{r_2^{-1}}, \cdots K^{r_m^{-1}}$ where $K = g^{r_1 r_2 \cdots r_m}$

GDH3 I

Version 3 minimizes the average computation of each principal. It has four phases:

- Phase 1: Partial information is generated by the first $m-1$ principals.
- Phase 2: Principal U_{m-1} broadcasts $g^{r_1 r_2 \cdots r_{m-1}} = K^{r_m^{-1}}$.
- Phase 3: Each of the principals $U_1, \cdots, U_{m-1}$ removes its exponent from the broadcast information and sends the result to principal U_m to add the final exponent to these partial values.
- Phase 4: Principal U_m applies its exponent r_m to all the received partial calculations and broadcasts the results. This allows each principal to find K by applying its exponent to the correct partial value.

GDH3 II

Phase 1:	$U_{i-1} \longrightarrow U_i$:	$g^{r_1 r_2 \cdots r_{i-1}}$
	$U_i \longrightarrow U_{i+1}$:	$g^{r_1 r_2 \cdots r_i}$
Phase 2:		U_{m-1} broadcasts $g^{r_1 r_2 \cdots r_{m-1}}$
Phase 3:	$U_i \longrightarrow U_m$:	$(g^{r_1 r_2 \cdots r_{m-1}})^{r_i^{-1}}$
Phase 4:		U_m broadcasts $K^{r_1^{-1}}, \cdots, K^{r_{m-1}^{-1}}$
		U_i calculates $(K^{r_i^{-1}})^{r_i} = K$

Addition and deletion of members is basically easy and we return to this topic in the exercises.

Burmester-Desmedt Protocol I

This is an efficient protocol, both in the number of messages sent per user and in the amount of computation required.
Protocol principals are arranged in a ring so that $U_1 = U_{m+1}$. The protocol is simplest to understand in the version that allows broadcast communications:

Phase 1	$U_i \longrightarrow U_{i-1}, U_{i+1}$:	$t_i = g^{r_i}$
	Then U_i calculates	$X_i = (t_{i+1}/t_{i-1})^{r_i}$,
		$Z_{i-1,i} = t_{i-1}^{r_i}$
Phase 2	U_i broadcasts	X_i
	U_i calculates	$K = (Z_{i-1,i})^m X_i^{m-1} X_{i+1}^{m-2} \cdots X_{i-2}$

A straightforward calculation shows that every user can compute the same secret key:

Burmester-Desmedt Protocol II

$$\begin{aligned}
K &= (Z_{i-1,i})^m X_i^{m-1} X_{i+1}^{m-2} \cdots X_{i-2} \\
&= (Z_{i-1,i})^m \cdot \left(\frac{t_{i+1}}{t_{i-1}}\right)^{(m-1)r_i} \cdot \left(\frac{t_{i+1}}{t_i}\right)^{(m-2)r_{i+1}} \cdots \left(\frac{t_{i-1}}{t_{i-3}}\right)^{r_{i-2}} \\
&= (Z_{i-1,i})^m \left(\frac{Z_{i,i+1}}{Z_{i-1,i}}\right)^{m-1} \cdot \left(\frac{Z_{i+1,i+2}}{Z_{i,i+1}}\right)^{m-2} \cdots \left(\frac{Z_{i-2,i-1}}{Z_{i-3,i-2}}\right) \\
&= Z_{i-1,i} Z_{i,i+1} Z_{i+1,i+2} \cdots Z_{i-2,i-1} \\
&= g^{r_1 r_2 + r_2 r_3 + \cdots + r_m r_1}.
\end{aligned}$$

Burmester-Desmedt without Broadcasts I

Broadcasts can be expensive. Burmester and Desmedt also proposed a protocol version that uses only communication between adjacent principals. The first phase of this version is the same as in the broadcast version. In the following algorithm,

$$b_0 = c_0 = 1.$$

Phase 1	$U_i \longrightarrow U_{i-1}, U_{i+1}$:	$t_i = g^{r_i}$
	Then U_i calculates	$X_i = (t_{i+1}/t_{i-1})^{r_i}$,
		$Z_{i-1,i} = t_{i-1}^{r_i}$
Phase 2	$U_i \longrightarrow U_{i+1}$:	b_i, c_i, where recursively
		$b_i = X_i b_{i-1}$, $c_i = b_{i-1} c_{i-1}$.
	$U_i \longrightarrow U_{i+1}$:	b_i, c_i
Phase 3	$U_i \longrightarrow U_{i+1}$:	d_i, where recursively
		$d_i = d_{i-1}/X_i^m$
		and U_i calculates $Z = d_i \cdot Z_{i,i+1}^m$

Burmester-Desmedt without Broadcasts II

When Phase 2 is complete, U_1 has received the value

$$c_m = X_1^{m-1} X_2^{m-2} \cdots X_{m-1}$$

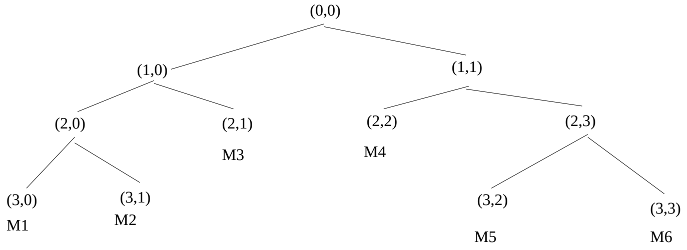
from U_m and sets this value to d_0. It is left to exercises that every user can compute the same secret key. In the case of U_1, this key is

$$K = d_0 \cdot Z_{0,1}^m$$

.

TGDH Protocol I

This protocol was invented by Kim, Perrig and Tsudik in 2004. It is based on Diffie-Hellman, but in a different way than our earlier conference protocols. It uses key trees. In the next figure, there is an example of a key tree:



Users or members are attached to leaves: $<3,0>$: M_1, $<3,1>$: M_2, $\cdots$ $<3,3>$: M_6.
General notions and notations:

TGDH Protocol II

- The root is located at level 0 and the lowest leaves are at level h (in figures $h = 3$). Since we use binary trees, every node is either a leaf or a parent of two nodes.
- The nodes are denoted (l, v), where $0 \leq v \leq 2^l - 1$, since each level l hosts at most $2^l - 1$ nodes.
- Each node (l, v) is associated with the key $K_{(l,v)}$ and the blinded key (bkey, public key) $BK_{(l,v)} = f\left(K_{(l,v)}\right)$, where the function f is modular exponentiation in prime order groups, that is

  $$f(k) = \alpha^k \mod p,$$

  where α is a primitive root.
- Members are associated to leaves only. The other nodes in the tree are called internal nodes. The tree is logical, i.e. in reality it is not formed, but the leaves manage the logical structure of the tree.

TGDH Protocol III

- The member M_i at node (l, v) knows every key along the path from (l, v) to $(0, 0)$, referred to as the key-path and denoted KEY_i^*. For example, in the figure M_2 knows every key $K_{(3,1)}, K_{(2,0)}, K_{(1,0)}, K_{(0,0)}$ in $KEY_2^* = \{< 3, 1 >, < 2, 0 >, < 1, 0 >, < 0, 0 >\}$.
- Every member knows every blinded key. Every member at leave (l, v) has a secret key $K_{(l,v)}$.
- Every other key (not at leave) is computed recursively as follows:

$$\begin{aligned} K_{(l,v)} &= \left(BK_{(l+1,2v+1)}\right)^{K_{(l+1,2v)}} \mod p \\ &= \left(BK_{(l+1,2v)}\right)^{K_{(l+1,2v+1)}} \mod p \\ &= \alpha^{K_{(l+1,2v)}K_{(l+1,2v+1)}} \mod p \\ &= f\left(K_{(l+1,2v)}K_{(l+1,2v+1)}\right. \end{aligned}$$

TGDH Protocol IV

- Members are only in leaves, while the other nodes in the tree are called internal or intermediate nodes. The tree is logical, i.e. in reality it is not constructed completely, but the keys in nodes must be calculated.
- As we have seen, computing a key at (l, v) requires the knowledge of the key of one of the two child nodes and the bkey of the other child node.
- $K_{(0,0)}$ at the root is the group secret shared by all members. This value is never used directly as an encryption or authentication key. Instead, special-purpose sub-keys are derived from the group key, for example by setting $K_{\mathrm{group}} = h(K(0,0))$, where h is a hash function.

Example I

- Consider our example tree with six members. Assume that the keys of the members are $r_1, r_2, \cdots, r_6$.
- The corresponding blinded keys are

$$BK_{3,0} = \alpha^{r_1}, \;\; BK_{3,1} = \alpha^{r_2}, \cdots, BK_{3,3} = \alpha^{r_6}.$$

- Let us show how the keys for internal nodes $(2,0)$ and $(1,0)$ are calculated by M_2:
-

$$\begin{aligned} K_{2,0} &= BK_{3,0}^{K_{3,1}} = (\alpha^{r_1})^{r_2} = \alpha^{r_1 r_2} \\ BK_{2,0} &= \alpha^{\alpha^{r_1 r_2}} \\ BK_{2,1} &= \alpha^{r_3} \\ K_{1,0} &= BK_{2,1}^{K_{2,0}} = (\alpha^{r_3})^{\alpha^{r_1 r_2}} \end{aligned}$$

Example II

- Notice that calculating the key $K_{0,0}$ demands knowledge of $BK_{1,1}$. So also the right branch of the tree must be calculated in order to get the shared secret for all.

To simplify subsequent protocol description, we introduce the term co-path, denoted as CO_i^*, which is the set of siblings of each node in the key-path of member M_i.

For example, the co-path of M_2 in the figure is

$$< 3,0 >, < 2,1 >, < 1,1 > .$$

Every member M_i at leaf node (l, v) can derive the group secret $K_{(,0)}$ from all bkeys on the co-path CO_i^* and its session random key $K_{(l,v)}$.

TGDH Operations I

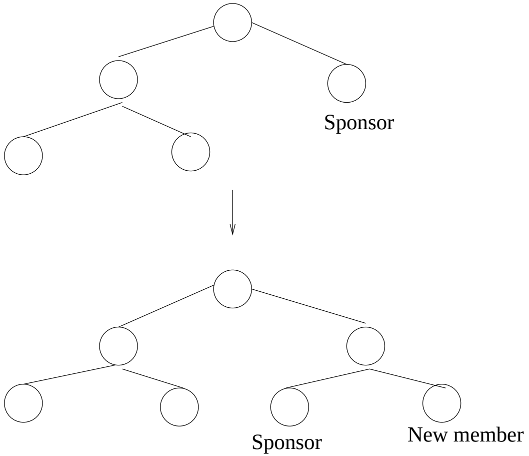
TGDH protocols consists of five protocols:

- Join: a new member is added to the group.
- Leave: a member is removed from the group.
- Merge: a group is merged with the current group.
- Partition: a subset of members are split from the group.
- Key refresh: the group key is updated.

We check only the first two. Some of the others are left to exercises.

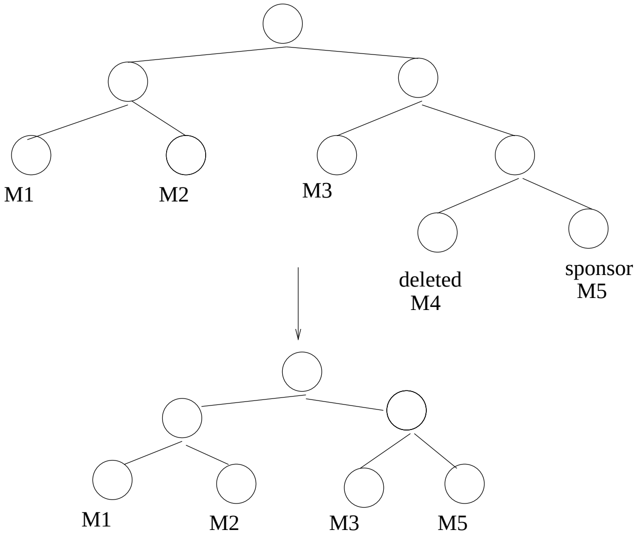
TGDH Join I

- We assume the group has n members $M_1, \cdots, M_n$.
- The new member M_{n+1} initiates the protocol by broadcasting a join request message that contains its own bkey.
- Each current member receives this message and determines the sponsor point in the tree. The sponsor is the rightmost leaf with the least depth in the tree.
- Next, sponsor creates a new intermediate node and a new member node, and promotes the new intermediate node to be the parent of both the sponsor node and the new member node. After updating the tree, all members, except the sponsor, block.
- The sponsor proceeds to update its share and computes the new group key; it can do this since it knows all the necessary bkeys. Next, the sponsor broadcasts the new tree that contains all bkeys.
- All other members update their trees accordingly and compute the new group key.


Sponsor
Sponsor
New member

TGDH Leave I

- Suppose there are n members in the group and M_d leaves.
- The sponsor is now the rightmost leaf node of the subtree rooted at the leaving member's sibling node.
- First, each member updates its key tree by deleting the leaf node of M_d.
- The former sibling of M_d is promoted to replace M_d's parent node.
- The sponsor generates a new key share, computes all (key,bkey) pairs on the key-path up to the root, and broadcasts the new set of bkeys. This allows all members to compute the new group key.
- Note that M_d cannot compute the group key, though it knows all the bkeys, because its share is no longer part of the group key.


M1
M2
M3
deleted M4
sponsor M5
M1
M2
M3
M5

Authentication in Conference Protocols I

- In many cases, the issue of key authentication have been ignored. Or it has been simply said that authentication using the digital signatures may be added.
- When there are no long-term keys providing authentication, forward secrecy is meaningless.
- Burmester and Desmedt proposed that their generalized Diffie-Hellman protocol could provide key authentication if each t_i value was authenticated by any chosen means.
- However, Just and Vaudenay (Authenticated multi-party key agreement, in Kim et al., editors, Advances in Cryptology – Asiacrypt '96, pp. 36-49) pointed out that authenticating messages is not sufficient, since it does not show that the party authenticating knows the random input r_i and consequently unknown key-share attacks are possible.

Authentication in Conference Protocols II

- Just and Vaudenay also propose a generalized form of the Burmester-Desmedt protocol, using their two party key agreement protocol as the building block, but it still provides weak key authentication.
- In the case of TGDH protocol, the key authentication was not assumed to be part of group key management. All communication channels are thus considered public but authentic.
- The latter means that all messages are digitally signed by the sender with some sufficiently strong public key signature method such as DSA or RSA. All receivers are required to verify signatures on all received messages and check the aforementioned fields.
- GDH.2 has authenticated versions, and we check two of them in more detail.

Authenticated GDH.2

- In the first authenticated version, only the final broadcast message is changed. It is assumed that the distinguished principal U_m shares a secret K_i with each U_i.
- It is suggested that this secret should be calculated from the static DH key
$$S_{(i,m)} = g^{x_i x_m},$$
shared between U_i and U_m.
- Then the first phase is the same as in the original protocol.
- In the second phase U_m sends $Z^{r_i^{-1} K_i}$ to U_i. On receipt of this message U_i can find the shared secret
$$Z = g^{r_1 r_2 \cdots r_m}$$
by raising the received message to the power $r_i K_i^{-1}$.

Authenticated GDH.2 Algorithm

Phase 1	$U_{i-1} \longrightarrow U_i$:	$p_{i-1}, p_{i-1}^{r_1^{-1}}, p_{i-1}^{r_2^{-1}}, \cdots, p_{i-1}^{r_{i-1}^{-1}}$, where $p_{i-1} = g^{r_1 r_2 \cdots r_{i-1}}$.
	$U_i \longrightarrow U_{i+1}$:	$p_i, p_i^{r_1^{-1}}, p_i^{r_2^{-1}}, \cdots, p_i^{r_i^{-1}}$
Phase 2	U_m broadcasts	$Z^{r_1^{-1}K_1}, Z^{r_2^{-1}K_2}, \cdots, Z^{r_m^{-1}K_m}$,
	U_i calculates	$Z = \left(Z^{r_i^{-1}K_i}\right)^{r_i K_i^{-1}}$

Analysis of Authenticated GDH.2 I

- The use of the shared secret K_i values means that principals can be sure that the value they calculate for Z will be known only to those that actually participate in the protocol with U_m.
- This is on the assunption that U_m follows the protocol faithfully. In this sense the protocol provides implicit key authentication.
- However, in common with many multi-party protocols, principals have no direct assurance of which other principals are participating in the protocol. This means that the principals really know which other parties have the shared secret only if the group is fixed or assured by some other means.
- Pereira and Quisquater conducted an analysis of the security of the protocol. They showed that strong key authentication can fail if the group membership varies. They showed the following attack.
  - $m = 4$.

Analysis of Authenticated GDH.2 II

- The adversary takes part in the protocol as U_3 but alters the message sent from U_3 to U_4 by replacing $g^{r_1 r_3}$ with $g^{r_1 r_2}$.
- As a consequence, U_4 will include $g^{r_1 r_2 r_4 K_2}$ in the broadcast part intended for U_2, instead of the correct value $g^{r_1 r_3 r_4 K_2}$.
- The adversary also obtains $g^{r_1 r_2 r_4}$ from the last broadcast part, since it knows K_3. The adversary records the values exchanged in this run.
- The adversary observes a new protocol run involving only the other three principals from the first run.
- In the new run suppose that new values r_1', r_2' and r_3' are chosen by U_1, U_2 and U_3.
- The adversary replaces the message from U_1 by $g^{r_1 r_2 r_4}$ obtained from the first run. Then U_2 will send $g^{r_1 r_2 r_4 r_2'}$ as part of its message to U_4.

Analysis of Authenticated GDH.2 III

- Finally, the adversary can replace the part of the broadcast message to U_2 by $g^{r_1 r_2 r_4 K_2}$ recorded from the first run. This means that U_2 will calculate

$$Z = g^{r_1 r_2 r_4 r_2'}$$

which was sent by U_2 in the second message and so is known to the adversary.

A Second Authenticated GDH.2 I

- The general idea is to allow each principal to mutually authenticate each other principal through use of a lon-term shared secret.
- Each pair of principals U_i and U_j shares two keys $K_{i,j}$ and $K_{j,i}$. An obvious way to calculate them is to use two different derivations of the long-term static static Diffie-Hellman key $S_{i,j}$.
- There are m users $U_1, \cdots, U_m$ as before. The first phase of the algorithm starts at U_1 and then proceeds through $U_2, \cdots, U_m$ in that order.

A Second Authenticated GDH.2 II

Phase 1 Rounds i, $0 < i < m$: U_i receives m values V_k, $1 \leq k \leq m$, where

$$V_k = \begin{cases} g^{\frac{r_1 \cdots r_{i-1}}{r_k} K_{k1} \cdots K_{k(i-1)}}, & \text{k} \leq i-1 \\ g^{(r_1 \cdots r_{i-1} K_{k1} \cdots K_{k(i-1)}}, & \text{k >i-1} \end{cases}$$

U_i updates each V_k as follows:

$$V_k = \begin{cases} V_k^{K_{ik} r_i}, & \text{k<i,} \\ V_k^{K_{ik} r_i}, & \text{k>i,} \\ V_k & k = i \end{cases}$$

In the initial round U_1 sets $V_1 = g$.

A Second Authenticated GDH.2 III

Phase 2 (round m): U_m broadcasts a set of all V_k values to the group. On receipt, each U_i selects the appropriate V_i, where

$$V_i = g^{\frac{r_1 \cdots r_m}{r_i}(K_{1i} \cdots K_{mi})}$$

U_i proceeds to compute

$$V_i^{r_i(K_{1i}^{-1} \cdots K_{mi}^{-1})}.$$

- Notice that the exponent $K_{1i}^{-1} \cdots K_{mi}^{-1}$ can be calculated by calculating a single inverse of $K_{1i} \cdots K_{mi}$, because $(K_{1i} \cdots K_{mi})^{-1} = K_{1i}^{-1} \cdots K_{mi}^{-1}$.
- The advantage of this protocol is that a stronger form of implicit key authentication is achieved. Each U_i knows that only principals that possess one of the shared keys K_{ij} are able to calculate the same value of the key calculated by U_i.

A Second Authenticated GDH.2 IV

- There is still no confirmation that any other principal does actually possess that key and again U_i must know the other group members' identities by some external means.
- Furthermore, the computational cost for each principal, with the exception of U_m, is greatly increased.

On the Performance I

In the article Amir, Kim, Nita-Rotaru, Tsudik: On the Performance of Group Key Agreement Protocols, ACM Transactions and Information System Security, Vol. 7, No. 3, August 2004, pp. 457-488, group key agreement protocols have been compared for perfomances. These protocols were GDH, TGDH, BD, STR and CKD. Of these, CDK is a simple group key management scheme and STR is an extreme version of TGDH with the underlying tree completely unbalanced.

General conclusions:

- Experiments show that communication cost for group-oriented cryptographic protocols over long delay networks can dominate the computational cost.
- When designing group-oriented protocols, most cryptographers focused on computational overhead and number of rounds. However, simultaneous *n* broadcast message for relatively large *n* is also very expensive in practice, and it, therefore, is recommended to be avoided.

On the Performance II

- The cost of BD roughly doubles as the group size grows in increment of the total number of machines and degrades significantly when the group size hits the number of processors.
- The best and worst case costs for TGDH can be theoretically analysed. The results show that TGDH is the best overall protocol in practice, if only one protocol has to be selected.

Application areas:

- *Peer groups of long-running servers*:
  - This group usually connects replicated servers that provide a service, as if they were one logical server. The entire group of servers can reside on one LAN, or they may be spread across WAN. These servers join the group upon startup, and never voluntarily leave the group.
  - The most prevalent membership events in such a group are partitions and merges. There may also be a limited number of server shutdowns and startups, usually for maintenance reasons.

On the Performance III

- When small number of servers is considered, BD would fit best. However, when the number of servers increases, TGDH would perform much better.
- Especially in a WAN setting, the self-clustering effect of TGDH would play a major role in reducing its inefficient partition cost. When servers are distributed over a high-delay WAN, STR can also be considered, as it has the most efficient communication cost.

- *Conferencing*:
  - In this group type, membership is built over time as participants join the conference, with an occasional participant joining or leaving. The group usually dissolves when most participants leave roughly at the same time, although the time to complete the mass leave event is not very important to the participants.
  - Again, for small number of participants, BD would fit best. Since most hosts in this application type are expected to run a single member process, BD would not show stepping behaviour.

On the Performance IV

- However, STR would fit better for large number of participants. Note that the cost of a subtractive event does not matter much in this case, since the group dissolves almost at the same time.

- *One-to-many broadcast*:
    - In this case there is one source of multicasting to many receivers. The group has no value without the presence of the source while receivers join and leave at will.
    - It is obvious that group key distribution protocols are most appropriate for this application class. In these protocols, one single member creates the group key and delivers it to other members. However, CKD would fit better for applications that require strong security properties.

- *Distributed logging*:
    - This case has several logging servers that accept updates from many participants, which may frequently join or leave the group.

On the Performance V

- For example, there are such systems with hundreds of participants, each of which with a lifespan of several minutes. This translates into several join or leave operations per second globally.
- This model requires the most scalable and efficient solution; therefore, TGDH would fit best.

- *Mobile state transfer*
  - This group type has up to a few tens of participants that share soft state. From time to time, participants join the group, exchange state, stay connected for a while, and temporarily leave the group.
  - In such a setting, there are no long-term participants in the group, but the group changes are usually less frequent than in the distributed logging case.
  - However, this also requires relatively frequent joins and leaves. Therefore, once again, TGDH would work best.